f3 Documentation

Release 8.0

Michel Machado

Oct 29, 2020

Contents

Contents:

CHAPTER 1

f3 - Fight Flash Fraud

f3 is a simple tool that tests flash cards capacity and performance to see if they live up to claimed specifications. It fills the device with pseudorandom data and then checks if it returns the same on reading.

F3 stands for Fight Flash Fraud, or Fight Fake Flash.

Table of Contents

CHAPTER 2

Examples

2.1 Testing performance with f3read/f3write

Use these two programs in this order. f3write will write large files to your mounted disk and f3read will check if the flash disk contains exactly the written files:

```
$ ./f3write /media/michel/5EBD-5C80/
$ ./f3read /media/michel/5EBD-5C80/
```

Please replace "/media/michel/5EBD-5C80/" with the appropriate path. USB devices are mounted in "/Volumes" on Macs.

If you have installed f3read and f3write, you can remove the "./" that is shown before their names.

2.2 Quick capacity tests with f3probe

f3probe is the fastest drive test and suitable for large disks because it only writes what's necessary to test the drive. It operates directly on the (unmounted) block device and needs to be run as a privileged user:

```
# ./f3probe --destructive --time-ops /dev/sdb
```

Warning: This will destroy any previously stored data on your disk!

2.3 Correcting capacity to actual size with f3fix

f3fix creates a partition that fits the actual size of the fake drive. Use f3probe's output to determine the parameters for f3fix:

```
# ./f3fix --last-sec=16477878 /dev/sdb
```

CHAPTER 3

Installation

3.1 Download and Compile

The files of the stable version of F3 are here. The following command uncompresses the files:

```
$ unzip f3-8.0.zip
```

3.2 Compile stable software on Linux or FreeBSD

To build:

```
make
```

If you want to install f3write and f3read, run the following command:

```
make install
```

3.3 Compile stable software on Windows/Cygwin

f3write and f3read can be installed on Windows, but currently f3probe, f3fix, and f3brew *require Linux*. To use them on a Windows machine, use the *Docker Installation*. For f3write and f3read, read on.

If you haven't already, install the following Cygwin packages and their dependencies:

- *gcc-core*
- *make*
- *libargp-devel*

To build, you need special flags:

```
export LDFLAGS="$LDFLAGS -Wl,--stack,4000000 -largp"
make
```

If you want to install f3write and f3read, run the following command:

```
make install
```

3.4 Compile stable software on Apple Mac

f3write and f3read can be installed on Mac, but currently f3probe, f3fix, and f3brew *require Linux*. To use them on Mac, use the *Docker Installation*. For f3write and f3read, read on.

3.4.1 Using HomeBrew

If you have Homebrew already installed in your computer, the command below will install F3:

```
brew install f3
```

3.4.2 Using MacPorts

If you use MacPorts instead, use the following command:

```
port install f3
```

3.4.3 Compiling the latest development version from the source code

Most of the f3 source code builds fine using XCode, the only dependency missing is the GNU C library "argp". You can build argp from scratch, or use the version provided by HomeBrew and MacPorts as "argp-standalone"

The following steps have been tested on OS X El Capitan 10.11.

1) Install Apple command line tools:

   ```
   xcode-select --install
   ```

See http://osxdaily.com/2014/02/12/install-command-line-tools-mac-os-x/ for details.

2) Install Homebrew or MacPorts

   HomeBrew:

   ```
   /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
   ↪master/install)"
   ```

   See https://brew.sh/ for details.

   MacPorts: https://www.macports.org/install.php

3) Install argp library:

   ```
   brew install argp-standalone
   ```

See http://brewformulas.org/ArgpStandalone and https://www.freshports.org/devel/argp-standalone/ for more information.

Or, for MacPorts:

```
port install argp-standalone
```

See https://trac.macports.org/browser/trunk/dports/sysutils/f3/Portfile for more information.

4) Build F3:

When using Homebrew, you can just run:

```
make
```

When using MacPorts, you will need to pass the location where MacPorts installed argp-standalone:

```
make ARGP=/opt/local
```

3.5 Docker

3.5.1 Quick Start

A pre-built image is available over at Docker Hub, ready to be used. With docker started, just run:

```
docker run -it --rm --device <device> peron/f3 <f3-command> [<f3-options>] <device>
```

For example, to probe a drive mounted at /dev/sdb:

```
docker run -it --rm --device /dev/sdb peron/f3 f3probe --destructive --time-ops /dev/
↪sdb
```

Optionally, you can also build your own container *if* you don't want to use the pre-built image. From this directory, run:

```
docker build -t f3:latest .
docker run -it --rm --device <device> f3:latest <f3-command> [<f3-options>] <device>
```

3.5.2 Drive Permissions / Passthrough

Getting the drive device to map into the Docker container is tricky for Mac and Windows. Passing through devices on Mac and Windows is a well-documented issue ([github] [stackexchange] [tty]) On Linux it should just work, but on Mac or Windows, Docker tends to map the drive as a normal directory rather than a mounted drive and you will get an error like `f3probe: Can't open device '/opt/usb': Is a directory`, that is if you can map it at all.

To solve this, we can use docker-machine to create a VirtualBox VM (boot2docker), in which to run the Docker container. Since VirtualBox *can* handle device pass-through, we can pass the device through to the VirtualBox VM which can then pass the device through to the Docker container. Milad Alizadeh wrote up some good instructions here which are geared towards USB devices, but it shouldn't be too hard to adapt to other drive types. Here's what I typed into my Mac terminal (probably similar for Windows, but untested):

```
docker-machine create -d virtualbox default
docker-machine stop
vboxmanage modifyvm default --usb on
docker-machine start
vboxmanage usbfilter add 0 --target default --name flashdrive --vendorid 0x0123 --
↪productid 0x4567
eval $(docker-machine env default)
```

For the usbfilter add command, note that the "name" argument is the new name you're giving the filter so you can name it whatever you want. `--vendorid` and `--productid` can be found on Mac in "System Information" under "USB". You can also try searching for the right device in `vboxmanage list usbhost`.

Alternatively, you may opt to add the device through the VirtualBox GUI application instead:

```
docker-machine create -d virtualbox default
docker-machine stop
# open VirtualBox and manually add the drive device before proceeding to the next␣
↪command
docker-machine start
eval $(docker-machine env default)
```

Once you've run the above commands, unplug and replug the flash drive and run:

```
docker-machine ssh default "lsblk"
```

to list the devices. Search for the correct drive - the "SIZE" column may be helpful in locating the device of interest. For example, `sdb` is a common mount point for a USB drive. Now you should be able to run the command from Quick Start:

```
docker run --rm -it --device /dev/sdb peron/f3 f3probe --destructive --time-ops /dev/
↪sdb
```

You may find it useful to enter a bash prompt in the Docker container to poke around the filesystem:

```
docker run --rm -it --device /dev/sdb peron/f3 bash
```

so that you can run commands like `ls /dev/*`.

3.6 The extra applications for Linux

3.6.1 Install dependencies

f3probe and f3brew require version 1 of the library libudev, and f3fix requires version 0 of the library libparted to compile. On Ubuntu, you can install these libraries with the following command:

```
sudo apt-get install libudev1 libudev-dev libparted0-dev
```

On Fedora, you can install these libraries with the following command:

```
sudo dnf install systemd-devel parted-devel
```

3.6.2 Compile the extra applications

```
make extra
```

Note:

- The extra applications are only compiled and tested on Linux platform.
- Please do not e-mail me saying that you want the extra applications to run on your platform; I already know that.
- If you want the extra applications to run on your platform, help to port them, or find someone that can port them for you. If you do port any of them, please send me a patch to help others.
- The extra applications are f3probe, f3brew, and f3fix.

If you want to install the extra applications, run the following command:

```
make install-extra
```

CHAPTER 4

Other resources

4.1 Graphical User Interfaces

Thanks to our growing community of flash fraud fighters, we have the following graphical user interfaces (GUI) available for F3:

F3 QT is a Linux GUI that uses QT. F3 QT supports `f3write`, `f3read`, `f3probe`, and `f3fix`. Author: Tianze.

Please support the above project by testing it and giving feedback to their authors. This will make their code improve as it has improved mine.

4.2 Files

```
changelog   - Change log for package maintainers
f3read.1    - Man page for f3read and f3write
              In order to read this manual page, run `man ./f3read.1`
              To install the page, run
              `install --owner=root --group=root --mode=644 f3read.1 /usr/share/man/
↪man1`
LICENSE     - License (GPLv3)
Makefile    - make(1) file
README      - This file
*.h and *.c - C code of F3
```

4.3 Bash scripts

Although the simple scripts listed in this section are ready for use, they are really meant to help you to write your own scripts. So you can personalize F3 to your specific needs:

```
f3write.h2w - Script to create files exactly like H2testw.
    Use example: `f3write.h2w /media/michel/5EBD-5C80/`

log-f3wr    - Script that runs f3write and f3read, and records
              their output into a log file.
    Use example: `log-f3wr log-filename /media/michel/5EBD-5C80/`
```

Please notice that all scripts and use examples above assume that f3write, f3read, and the scripts are in the same folder.

CHAPTER 5

Usage

Contents

5.1 How to use f3write and f3read

If you prefer watching a video than reading the text below, check out Spatry's Cup of Linux's video demo of F3 on YouTube here.

My implementation of H2testw is composed of two applications: `f3write`, and `f3read`. `f3write` fills a file system up with 1GB files named N.h2w, where N is a number. Whereas, `f3read` validates those files. If the content of all N.h2w files is valid, the drive is fine. The last file may be less than 1GB since `f3write` takes all available space for data. Below the result on my fake card:

```
$ ./f3write /media/michel/5EBD-5C80/
Free space: 28.83 GB
Creating file 1.h2w ... OK!
Creating file 2.h2w ... OK!
```

(continues on next page)

(continued from previous page)

```
Creating file 3.h2w ... OK!
Creating file 4.h2w ... OK!
Creating file 5.h2w ... OK!
Creating file 6.h2w ... OK!
Creating file 7.h2w ... OK!
Creating file 8.h2w ... OK!
Creating file 9.h2w ... OK!
Creating file 10.h2w ... OK!
Creating file 11.h2w ... OK!
Creating file 12.h2w ... OK!
Creating file 13.h2w ... OK!
Creating file 14.h2w ... OK!
Creating file 15.h2w ... OK!
Creating file 16.h2w ... OK!
Creating file 17.h2w ... OK!
Creating file 18.h2w ... OK!
Creating file 19.h2w ... OK!
Creating file 20.h2w ... OK!
Creating file 21.h2w ... OK!
Creating file 22.h2w ... OK!
Creating file 23.h2w ... OK!
Creating file 24.h2w ... OK!
Creating file 25.h2w ... OK!
Creating file 26.h2w ... OK!
Creating file 27.h2w ... OK!
Creating file 28.h2w ... OK!
Creating file 29.h2w ... OK!
Free space: 0.00 Byte
Average Writing speed: 2.60 MB/s

$ ./f3read /media/michel/5EBD-5C80/
                 SECTORS      ok/corrupted/changed/overwritten
Validating file 1.h2w ...       0/  2097152/      0/      0
Validating file 2.h2w ...       0/  2097152/      0/      0
Validating file 3.h2w ...       0/  2097152/      0/      0
Validating file 4.h2w ...       0/  2097152/      0/      0
Validating file 5.h2w ...       0/  2097152/      0/      0
Validating file 6.h2w ...       0/  2097152/      0/      0
Validating file 7.h2w ...       0/  2097152/      0/      0
Validating file 8.h2w ...       0/  2097152/      0/      0
Validating file 9.h2w ...       0/  2097152/      0/      0
Validating file 10.h2w ...       0/  2097152/      0/      0
Validating file 11.h2w ...       0/  2097152/      0/      0
Validating file 12.h2w ...       0/  2097152/      0/      0
Validating file 13.h2w ...       0/  2097152/      0/      0
Validating file 14.h2w ...       0/  2097152/      0/      0
Validating file 15.h2w ...       0/  2097152/      0/      0
Validating file 16.h2w ...       0/  2097152/      0/      0
Validating file 17.h2w ...       0/  2097152/      0/      0
Validating file 18.h2w ...       0/  2097152/      0/      0
Validating file 19.h2w ...       0/  2097152/      0/      0
Validating file 20.h2w ...       0/  2097152/      0/      0
Validating file 21.h2w ...       0/  2097152/      0/      0
Validating file 22.h2w ...       0/  2097152/      0/      0
Validating file 23.h2w ...       0/  2097152/      0/      0
Validating file 24.h2w ... 1916384/   180768/      0/      0
Validating file 25.h2w ...  186816/  1910336/      0/      0
```

(continues on next page)

(continued from previous page)

```
Validating file 26.h2w ...       0/  2097152/      0/      0
Validating file 27.h2w ...       0/  2097152/      0/      0
Validating file 28.h2w ...       0/  2097152/      0/      0
Validating file 29.h2w ...   28224/  1705280/      0/      0

  Data OK: 1.02 GB (2131424 sectors)
Data LOST: 27.81 GB (58322336 sectors)
           Corrupted: 27.81 GB (58322336 sectors)
    Slightly changed: 0.00 Byte (0 sectors)
         Overwritten: 0.00 Byte (0 sectors)
Average Reading speed: 9.54 MB/s
```

This report shows that my flash card is pretty much garbage since it can only hold 1.02GB. `f3write` only writes to free space, and will not overwrite existing files as long as they aren't named N.h2w. However, as the previous report shows, files from 1.h2w to 23.h2w were written before 24.h2w and yet had all their content destroyed. Therefore, it is not wise to test nonempty cards because if the card has a problem, it may erase the old files.

When `f3read` reads a sector (i.e. 512 bytes, the unit of communication with the card), `f3read` can check if the sector was correctly written by `f3write`, and figure out in which file the sector should be and in which position in that file the sector should be. Thus, if a sector is well formed, or with a few bits flipped, but read in an unexpected position, `f3read` counts it as overwritten. Slightly changed sectors, are sectors at right position with a fews bits flipped.

Notice that `f3write` doesn't overwrite sectors by itself, it's done by the drive as a way to difficult a user to uncover its fault. The way the drive overwrites sectors is arbitrary. From the point of view of a file system, what `f3read` sees, the way the drive wraps around seems often contrived, but, from the drive's view, it is just an address manipulation.

The last lines of the output of `f3write` and `f3read` provide good estimates of the writing and reading speeds of the tested card. This information can be used to check if the claimed class of the card is correct. Check this link out for more information about classes. Note that the speeds provided by F3 are estimates, don't take them as perfect since they suffer influence even from other processes in your machine. Also, be aware that your card reader and USB port can limit the throughput of the drive.

Later I bought a second card that works just fine; I got the following output running F3 on it:

```
$ ./f3write /media/michel/6135-3363/
Free space: 29.71 GB
Creating file 1.h2w ... OK!
Creating file 2.h2w ... OK!
Creating file 3.h2w ... OK!
Creating file 4.h2w ... OK!
Creating file 5.h2w ... OK!
Creating file 6.h2w ... OK!
Creating file 7.h2w ... OK!
Creating file 8.h2w ... OK!
Creating file 9.h2w ... OK!
Creating file 10.h2w ... OK!
Creating file 11.h2w ... OK!
Creating file 12.h2w ... OK!
Creating file 13.h2w ... OK!
Creating file 14.h2w ... OK!
Creating file 15.h2w ... OK!
Creating file 16.h2w ... OK!
Creating file 17.h2w ... OK!
Creating file 18.h2w ... OK!
Creating file 19.h2w ... OK!
Creating file 20.h2w ... OK!
```

(continues on next page)

(continued from previous page)

```
Creating file 21.h2w ... OK!
Creating file 22.h2w ... OK!
Creating file 23.h2w ... OK!
Creating file 24.h2w ... OK!
Creating file 25.h2w ... OK!
Creating file 26.h2w ... OK!
Creating file 27.h2w ... OK!
Creating file 28.h2w ... OK!
Creating file 29.h2w ... OK!
Creating file 30.h2w ... OK!
Free space: 0.00 Byte
Average Writing speed: 4.90 MB/s

$ ./f3read /media/michel/6135-3363/
                  SECTORS      ok/corrupted/changed/overwritten
Validating file 1.h2w ... 2097152/        0/      0/      0
Validating file 2.h2w ... 2097152/        0/      0/      0
Validating file 3.h2w ... 2097152/        0/      0/      0
Validating file 4.h2w ... 2097152/        0/      0/      0
Validating file 5.h2w ... 2097152/        0/      0/      0
Validating file 6.h2w ... 2097152/        0/      0/      0
Validating file 7.h2w ... 2097152/        0/      0/      0
Validating file 8.h2w ... 2097152/        0/      0/      0
Validating file 9.h2w ... 2097152/        0/      0/      0
Validating file 10.h2w ... 2097152/        0/      0/      0
Validating file 11.h2w ... 2097152/        0/      0/      0
Validating file 12.h2w ... 2097152/        0/      0/      0
Validating file 13.h2w ... 2097152/        0/      0/      0
Validating file 14.h2w ... 2097152/        0/      0/      0
Validating file 15.h2w ... 2097152/        0/      0/      0
Validating file 16.h2w ... 2097152/        0/      0/      0
Validating file 17.h2w ... 2097152/        0/      0/      0
Validating file 18.h2w ... 2097152/        0/      0/      0
Validating file 19.h2w ... 2097152/        0/      0/      0
Validating file 20.h2w ... 2097152/        0/      0/      0
Validating file 21.h2w ... 2097152/        0/      0/      0
Validating file 22.h2w ... 2097152/        0/      0/      0
Validating file 23.h2w ... 2097152/        0/      0/      0
Validating file 24.h2w ... 2097152/        0/      0/      0
Validating file 25.h2w ... 2097152/        0/      0/      0
Validating file 26.h2w ... 2097152/        0/      0/      0
Validating file 27.h2w ... 2097152/        0/      0/      0
Validating file 28.h2w ... 2097152/        0/      0/      0
Validating file 29.h2w ... 2097152/        0/      0/      0
Validating file 30.h2w ... 1491904/        0/      0/      0

  Data OK: 29.71 GB (62309312 sectors)
Data LOST: 0.00 Byte (0 sectors)
           Corrupted: 0.00 Byte (0 sectors)
    Slightly changed: 0.00 Byte (0 sectors)
         Overwritten: 0.00 Byte (0 sectors)
Average Reading speed: 9.42 MB/s
```

Since `f3write` and `f3read` are independent, `f3read` can be used as many times as one wants, although `f3write` is needed only once. This allows one to easily repeat a test of a card as long as the N.h2w files are still available.

As a final remark, if you want to run `f3write` and `f3read` with a single command, check out the shell

script `log-f3wr <https://github.com/AltraMayor/f3/blob/master/log-f3wr>`__. This script runs `f3write` and `f3read`, and records their output into a log file. Use example: `log-f3wr log-filename /media/michel/5EBD-5C80/`

5.1.1 Users' notes

Randy Champoux has brought to my attention that `f3read` could eventually read data from the system cache instead of from the flash card. Since version 2.0, F3 eliminates this possibility as long as the kernel honors the system call `posix_fadvise(2)` with advice `POSIX_FADV_DONTNEED`. Linux has and honor `posix_fadvise(2)`/`POSIX_FADV_DONTNEED`, the Mac does not have the system call, and I don't know if Windows/Cygwin, or FreeBSD honors it. In doubt about this issue, just disconnect and connect back the device after `f3write` runs and before calling `f3read`.

Notice that the issue pointed by Randy Champoux is entirely an OS matter, that is, it doesn't change if the drive being tested is fake or not. In 2014, I've run into a "smarter" fake card that tries hard to behave like a good one using its internal cache to fool the test. In practice, these newer cards can only mislead `f3read` with a limited number of blocks, but those looking for a precise, repeatable report should follow the advice of disconnecting and connecting back the device before `f3read` runs. Consider the real example of a fake drive that presents this behavior. The drive announces a size of 128GB but its real capacity is less than 8GB:

```
$ ./f3write --end-at=16 /media/michel/DISK_IMG/ && ./f3read /media/michel/DISK_IMG/
Free space: 124.97 GB
Creating file 1.h2w ... OK!
Creating file 2.h2w ... OK!
Creating file 3.h2w ... OK!
Creating file 4.h2w ... OK!
Creating file 5.h2w ... OK!
Creating file 6.h2w ... OK!
Creating file 7.h2w ... OK!
Creating file 8.h2w ... OK!
Creating file 9.h2w ... OK!
Creating file 10.h2w ... OK!
Creating file 11.h2w ... OK!
Creating file 12.h2w ... OK!
Creating file 13.h2w ... OK!
Creating file 14.h2w ... OK!
Creating file 15.h2w ... OK!
Creating file 16.h2w ... OK!
Free space: 108.97 GB
Average writing speed: 2.87 MB/s
                  SECTORS      ok/corrupted/changed/overwritten
Validating file 1.h2w ... 2097152/        0/      0/      0
Validating file 2.h2w ... 2097152/        0/      0/      0
Validating file 3.h2w ... 2097152/        0/      0/      0
Validating file 4.h2w ... 2097152/        0/      0/      0
Validating file 5.h2w ... 2097152/        0/      0/      0
Validating file 6.h2w ... 2097152/        0/      0/      0
Validating file 7.h2w ... 2097152/        0/      0/      0
Validating file 8.h2w ...  266176/  1830976/      0/      0
Validating file 9.h2w ...       0/  2097152/      0/      0
Validating file 10.h2w ...      0/  2097152/      0/      0
Validating file 11.h2w ...      0/  2097152/      0/      0
Validating file 12.h2w ...      0/  2097152/      0/      0
Validating file 13.h2w ...      0/  2097152/      0/      0
Validating file 14.h2w ...      0/  2097152/      0/      0
Validating file 15.h2w ...      0/  2097152/      0/      0
```

(continues on next page)

(continued from previous page)

```
Validating file 16.h2w ...  523920/  1573232/      0/      0

  Data OK: 7.38 GB (15470160 sectors)
Data LOST: 8.62 GB (18084272 sectors)
           Corrupted: 8.62 GB (18084272 sectors)
    Slightly changed: 0.00 Byte (0 sectors)
         Overwritten: 0.00 Byte (0 sectors)
Average reading speed: 12.73 MB/s
```

After disconnecting the drive and connecting it back, `f3read` produced the following output:

```
$ ./f3read /media/michel/DISK_IMG/
                  SECTORS      ok/corrupted/changed/overwritten
Validating file 1.h2w ... 2097152/        0/      0/      0
Validating file 2.h2w ... 2097152/        0/      0/      0
Validating file 3.h2w ... 2097152/        0/      0/      0
Validating file 4.h2w ... 2097152/        0/      0/      0
Validating file 5.h2w ... 2097152/        0/      0/      0
Validating file 6.h2w ... 2097152/        0/      0/      0
Validating file 7.h2w ... 2097152/        0/      0/      0
Validating file 8.h2w ...  266176/  1830976/      0/      0
Validating file 9.h2w ...       0/  2097152/      0/      0
Validating file 10.h2w ...      0/  2097152/      0/      0
Validating file 11.h2w ...      0/  2097152/      0/      0
Validating file 12.h2w ...      0/  2097152/      0/      0
Validating file 13.h2w ...      0/  2097152/      0/      0
Validating file 14.h2w ...      0/  2097152/      0/      0
Validating file 15.h2w ...      0/  2097152/      0/      0
Validating file 16.h2w ...      0/  2097152/      0/      0

  Data OK: 7.13 GB (14946240 sectors)
Data LOST: 8.87 GB (18608192 sectors)
           Corrupted: 8.87 GB (18608192 sectors)
    Slightly changed: 0.00 Byte (0 sectors)
         Overwritten: 0.00 Byte (0 sectors)
Average reading speed: 12.50 MB/s
```

Notice that file 16.h2w, that last file `f3write` wrote, has no longer good sectors. What happened is that the last sectors of 16.h2w were in the internal cache of the drive when `f3read` ran right after `f3write`, but were not there after the forced reset. The internal cache will fool any test that doesn't write beyond the real capacity of the drive plus the size of the internal cache, and does not hard reset the drive. One can estimate the size of this cache as follows: 523920 * 512B ~ 256MB.

Tom Metro once ran `f3write` on a 16GB flash drive formatted with ext2 file system, and obtained puzzling free space at the end of `f3write`'s output:

```
% ./f3write /media/Kodi/
Free space: 14.50 GB
Creating file 1.h2w ... OK!
Creating file 2.h2w ... OK!
Creating file 3.h2w ... OK!
Creating file 4.h2w ... OK!
Creating file 5.h2w ... OK!
Creating file 6.h2w ... OK!
Creating file 7.h2w ... OK!
Creating file 8.h2w ... OK!
```

(continues on next page)

(continued from previous page)

```
Creating file 9.h2w ... OK!
Creating file 10.h2w ... OK!
Creating file 11.h2w ... OK!
Creating file 12.h2w ... OK!
Creating file 13.h2w ... OK!
Creating file 14.h2w ... OK!
Free space: 755.80 MB
Average writing speed: 13.77 MB/s
```

This happened because ext2 and some other file systems reserve space for special purposes. So they don't allow `f3write` to use that reserved space. It's mostly safe to ignore that free space. If one wants to use all space possible, there're two options: (1) using a file system that doesn't reserve space (e.g. FAT), or (2) reducing the reserved space. How to go for the second option depends on the used file system. The page explains how to reduce the reserved space on ext2, ext3, and ext4 file systems.

Elliot Macneille has ran into an application that reports the size of one of its good flash cards as 15.71GB, whereas `f3read` only finds 14.63GB. Details on how space is accounted varies with operating system, applications, file system used to format the drive, etc. However, there is a common source for this problem that often explains most of the difference: part of the computer industry (including F3) takes 1GB as being 2^30 bytes, whereas the rest of the industry assumes that 1GB is equal to 10^9 bytes. Some people use GiB for the first definition, but its use is not universal, and some users even get confused when they see this unit. With this information in mind, the mystery is easily solved: 14.63GiB * 2^30 / 10^9 = 15.71GB.

When Art Gibbens tested a flash card hosted in a camera connected to his Linux box, at some point F3 didn't show progress, and could not be killed. After a reboot, the card was read only. Using an adapter to connect his card directly to his machine, he recreated the partition of the card, and successfully ran F3 with the card in the adapter. Thus, Art's experience is a good warning if you're testing your card in a device other than an adapter. Please, don't take it as a bug of F3. I'm aware of only two things that can make a process "survive" a kill signal: hardware failures, and/or bugs in the kernel. F3 doesn't run in kernel mode, so Art's camera is likely the root of the problem.

Darrell Johnson has reported that a flash card he got stopped working after he filled it up. This could be that the only memory chip the card had died, but it is just speculation since Darrell was not able to obtain more information. The important message here is that if you test your card with F3, or just copy files into it, and it stops working, it's not your fault because writing files to a card shouldn't damage it if it is a legitimate card.

Username Kris, here, asked what's the difference between "dosfsck -vtr /dev/sda1" and F3. dosfsck(8) makes two assumptions that F3 does not: (1) one needs write access to the device being tested, not the file system in it; (2) hardware may fail, but it won't lie. The first assumption implies that one likely needs root's rights to run dosfsck, what is just a small inconvenience for simple uses. The second assumption is troublesome because a fake card may be able to persuade dosfsck(8) to report it's fine, or not report the whole problem, or give users the illusion the memory card was fixed when it wasn't. I singled dosfsck(8) out because of the question about it, but those two assumptions are true for fsck software for other file systems and badblocks(8) as well.

Mac user Athanasios Tourtouras noticed that Spotlight of OS X, which runs in the background, also indexes the content of removable drives. Although Spotlight does not interferes with `f3write`/`f3read`, its indexing takes away around 2MB/s of bandwidth, so `f3write`/`f3read` will run slower as well as their speed measurements will underestimate the real capacity of the drive. Not to mention that you likely don't want to index test files. You can disable the indexing of removable drives including the flash drive to Spotlight's exclude list by going to System Preferences / Spotlight / Privacy.

5.1.2 On the compatibility with H2testw's file format

Starting at version 3.0, F3 generates files following H2testw's file format. This feature should help users that use both applications and/or interact with users that use the other application. However, there are two caveats explained in this section that users should be aware.

Verifying files created by H2testw with F3read. The caveat here is that H2testw only writes files whose size is a multiple of 1MB, whereas F3write fills up the memory card. Thus, verifying files created by H2testw with F3read works, but, likely, will not test approximately 1MB of space that H2testw does not write.

Verifying files created by ‘‘f3write‘‘ with H2testw. The caveat here is that H2testw requires that all files have sizes that are multiple of 1MB. When it is not the case for a single file, H2testw rejects all files, and issues the message “Please delete files *.h2w.” This problem often comes up with the last file `f3write` generates to fill up the space available in the memory card being tested. The solution is to truncate the size of the last file `f3write` generates to the closest multiple of 1MB. Assuming the last file is 30.h2w, the following command does exactly that:

```
$ truncate --size=/1M /media/michel/6135-3363/30.h2w
```

If you want to exchange files with H2testw users often, check out the shell script `` `f3write.h2w <https://github.com/AltraMayor/f3/blob/master/f3write.h2w>‘__ ``. This script calls `truncate` after `f3write` runs successfully.

5.2 f3probe - the fastest drive test

H2testw’s algorithm has been the gold standard for detecting fake flash (see here and here) because it is robust against all counterfeits. However, as drives’ capacity grows, the time to test these newer drives becomes so painful that one rarely runs H2testw’s algorithm on a whole drive, but only a fraction of it. See question “Why test only 25% or 32GB?” on this FAQ for a defense of this approach.

The problem with this approach is that drives are still getting bigger, and counterfeiters may, in the future, be able to profit with fake drives whose real capacity are large enough to fool these partial tests. This problem is not new. For example, Steve Si implemented FakeFlashTest.exe, which has successfully reduced the amount of time to test drives, and is still able to give a good estimate of the real size of fake drives. Yet, FakeFlashTest.exe’s algorithm is not a definitive answer to the problem because FakeFlashTest.exe’s algorithm still needs to write to at least all good memory of tested drives.

When a drive is fake, `f3probe` writes the size of the cache of the drive a couple times, and a small number of blocks as the example in the next section shows. From the example in the next section, the fake drive has 7.86GB (16477879 blocks) of usable memory, but advertises itself as being a 15.33GB (32155648 blocks) drive. It is worth noticing that given that 7.86GB / 15.33GB ~ 51.2%, this fake drive already violates the 25% recommendation mentioned above. `f3probe` wrote 2158 blocks to find the real size of the drive, whereas FakeFlashTest.exe would have written at least 16477879 blocks. That is, `f3probe` wrote no more than 2158 / 16477879 ~ 0.01% of all that FakeFlashTest.exe would have written. Even if FakeFlashTest.exe wrote only 1% of the real size of the drive, `f3probe` would still write only 1% of what FakeFlashTest.exe would write under this hypothetical, two-order improvement! Moreover, `f3probe` provides the exact geometry of the drive, what allows one to “fix” the drive using the highest capacity possible.

When a drive is not fake, `f3probe` writes about half of its size, or 2GB, whichever is smaller. Thus, if the previous drive were not fake, `f3probe` would’ve written 2GB, and FakeFlashTest.exe 15.33GB. While the difference 2GB / 15.33GB ~ 13.05% is much smaller, it is still large. When a fake drive has some cache, `f3probe` will slow down, but given that `f3probe` is optimized to deal with these cases it is still fast. I do not know how FakeFlashTest.exe deal with drives that have cache. If FakeFlashTest.exe simply ignores a drive’s cache, it may over estimate the size of fake drives.

This breakthrough against counterfeiters was only possible because `f3probe`’s algorithm assumes a tight model of how fake drives work. In spite of the fact that I have not found a drive, fake or not, that confuses `f3probe`, I’ve marked `f3probe` as experimental for now because this model has not been battle proven. Although there is a chance of finding out that the model is incomplete, there is also a chance that the model can be simplified if one can be sure that not all types of fake flash the model predicts really exist; the latter chance holds a promise of even higher testing speeds. Of course, efficient flash tests like the one implemented in `f3probe` may slow down as fake chips become “smarter”. For now, though, `f3probe` gives us the upper hand over counterfeiters.

Finally, thanks to `f3probe` being free software, and once `f3probe` is battle proven, `f3probe` could be embedded on smartphones, cameras, MP3 players, and other devices to stop once and for all the proliferation of fake flash.

5.2.1 How to use f3probe

Different from `f3write/f3read` that works on the file system of the drive, `f3probe` works directly over the block device that controls the drive. In practice, this means three requirements. First, one has to have root access (i.e. superuser account) on the machine that will run the test. Second, the user must know how to find the block device of the drive. Third, you must be careful on the previous requirement to avoid messing your machine up. If you don't have root access, you can't use `f3probe`; use `f3write/f3read` in this case. The use example below helps with the second requirement, but don't forget that you are the one responsible for doing it right!

The command lsblk(8) is handy to find the block device of the drive. In the example below, which I got running lsblk on my laptop, an experienced user can quickly identify that my flash drive that is mounted at "/media/michel/A902-D705" is block device "/dev/sdb". If you don't have much experience, you may want to run lsblk before connecting the drive to your computer, and to run lsblk again after connecting the drive to easily identify what was added to the output of lsblk. Checking out the content of folder "/media/michel/A902-D705" to confirm that it's the correct drive is a good idea as well. The block device "sdb" is the disk (see column "TYPE"), and "sdb1" is the first and only partition of my flash drive; your drive may have none or more partitions. You want to choose the drive, not a partition.

```
$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0 232.9G  0 disk
+-sda1   8:1    0   218G  0 part /
+-sda2   8:2    0     1K  0 part
+-sda5   8:5    0    15G  0 part [SWAP]
sdb      8:16   1  15.3G  0 disk
+-sdb1   8:17   1  15.3G  0 part /media/michel/A902-D705
sr0     11:0    1  1024M  0 rom
```

If you get confused between "sdb" and "sdb1", don't worry, `f3probe` will report the mistake and point out the proper one. However, I cannot emphasize it enough, you MUST identify the correct drive. If I had chosen "sda", `f3probe` may have messed my computer. Once the device is chosen, just prefix it with "/dev/" to obtain its full name.

Once you have carefully identified the device, you run `f3probe` like in the example below (please use the correct device!):

```
$ sudo ./f3probe --destructive --time-ops /dev/sdb
[sudo] password for michel:
F3 probe 8.0
Copyright (C) 2010 Digirati Internet LTDA.
This is free software; see the source for copying conditions.

WARNING: Probing normally takes from a few seconds to 15 minutes, but
         it can take longer. Please be patient.

Bad news: The device `/dev/sdb' is a counterfeit of type limbo

You can "fix" this device using the following command:
f3fix --last-sec=16477878 /dev/sdb

Device geometry:
             *Usable* size: 7.86 GB (16477879 blocks)
            Announced size: 15.33 GB (32155648 blocks)
                   Module: 16.00 GB (2^34 Bytes)
    Approximate cache size: 0.00 Byte (0 blocks), need-reset=yes
```

(continues on next page)

(continued from previous page)

```
       Physical block size: 512.00 Byte (2^9 Bytes)

Probe time: 1'13"
 Operation: total time / count = avg time
      Read: 472.1ms / 4198 = 112us
     Write: 55.48s / 2158 = 25.7ms
     Reset: 17.88s / 14 = 1.27s
```

There is a lot in the previous example. First, it took one minute and 13 seconds for `f3probe` to identify that this 16GB drive had only 7.86GB. Second, I used command sudo(8) to run `f3probe` as root. Third, I used option "–time-ops" to add the last four lines of the output; these lines show the time taken to read, write, and reset the drive during the test. The rest of this section covers the other aspects of this output.

The option –destructive instructs `f3probe` to disregard the content of the drive to speed up the test. Without option –destructive, one would see a line "Probe finished, recovering blocks... Done" in the previous output to let the user know that `f3probe` has recovered all blocks in the drive to their original state. While the conservative mode is very convenient, you should not rely too much on it. If `f3probe` crashes, the conservative mode won't work. Moreover, depending on the fake drive, the conservative mode may not recover the drive to its exact original state. In case you are running `f3probe` on a memory-constrained computer (e.g. an old Raspberry Pi board), you can still run it in conservative mode reducing the amount of memory needed with option "–min-memory". If you don't have memory to test a large drive even using option "–min-memory", you need to use option "–destructive". The conservative mode is the default in the hope that it will eventually save you from a mistake.

The line "Bad news: The device '/dev/sdb' is a counterfeit of type limbo" summarizes the results presented below this line. The types of drives are good, damaged (seriously failing), limbo (the most common type of fake drives), wraparound (a rare, if existing at all, type of fake drives), and chain (a rare type of fake drives). If you ever find wraparound and chain drives, please consider donating them to my collection.

The probe time of 1'13" includes the time to run the probe algorithm, take measurements, and the time to perform all operations on the drive. But it doesn't include the time to recover the saved blocks (if this feature is enabled). Therefore, the test would take roughly another 55.48s (i.e. total write time) to write all blocks back to the drive. As some will notice, the time to perform all operations on the drive is what dominates the probe time: 472.1ms + 55.48s + 17.88s = 1'13". It's worth noticing that read and write speed estimates derived from the times of these operations are not accurate because they mix sequential and random accesses.

The next example gives you the chance to practice reading `f3probe`'s outputs:

```
$ sudo ./f3probe --time-ops /dev/sdc
[sudo] password for michel:
F3 probe 8.0
Copyright (C) 2010 Digirati Internet LTDA.
This is free software; see the source for copying conditions.

WARNING: Probing normally takes from a few seconds to 15 minutes, but
         it can take longer. Please be patient.

Probe finished, recovering blocks... Done

Good news: The device `/dev/sdc' is the real thing

Device geometry:
             *Usable* size: 3.77 GB (7913472 blocks)
            Announced size: 3.77 GB (7913472 blocks)
                   Module: 4.00 GB (2^32 Bytes)
    Approximate cache size: 0.00 Byte (0 blocks), need-reset=no
       Physical block size: 512.00 Byte (2^9 Bytes)
```

(continues on next page)

(continued from previous page)

```
Probe time: 10'06"
 Operation: total time / count = avg time
      Read: 2'22" / 3724018 = 38us
     Write: 7'41" / 3719233 = 124us
     Reset: 379.7ms / 1 = 379.7ms
```

This second drive is a good one; it has all blocks necessary to hold its announced size of 3.77GB, what is roughly 4GB.

The next section shows how to fix the 16GB drive using `f3fix` as suggested by `f3probe`.

5.2.2 Users' notes

Philip de Lisle has a SD card reader on this laptop that is not backed by a USB port. So when he tries `f3probe /dev/mmcblk0`, he gets the error message: `Device `/dev/mmcblk0' is not backed by a USB device`. This happens because the current version of `f3probe` only works on devices that are mounted at a USB port; a future release of `f3probe` may lift this restriction. In the meanwhile, one can work around this restriction using an external USB card reader.

5.3 How to "fix" a fake drive

You should not easily settle down for a fake drive, fight back and get your money back! Doing so will help you and others. If you are still reading this section, you already realized that you own a fake drive, and would like to be able to use it without losing data.

As shown in the previous section, my 16GB fake drive can only hold 7.86GB. Moreover, `f3probe` suggested how I can use `f3fix` to fix my drive. `f3fix` fixes fake drives creating a partition that includes only the usable memory of the drive. `f3fix` takes a few seconds to finish.

The execution of `f3fix` on my fake drive went as follows:

```
$ sudo ./f3fix --last-sec=16477878 /dev/sdb
F3 fix 8.0
Copyright (C) 2010 Digirati Internet LTDA.
This is free software; see the source for copying conditions.

Error: Partition(s) 1 on /dev/sdc have been written, but we have been unable to␣
↪inform the kernel of the change, probably because it/they are in use.  As a result,␣
↪the old partition(s) will remain in use.  You should reboot now before making␣
↪further changes.
Drive `/dev/sdc' was successfully fixed
```

If `f3fix` reports that you need to force the kernel to reload the new partition table, as shown above, just unplug and plug the drive back. Once the new partition is available, format it:

```
$ sudo mkfs.vfat /dev/sdb1
mkfs.fat 3.0.26 (2014-03-07)
```

At this point your card should be working fine, just mount the new partition to access it. However, before using the drive, test all its blocks with `f3write/f3read`. The test of my card went as follows:

```
$ ./f3write /media/michel/8A34-CED2/
Free space: 7.84 GB
Creating file 1.h2w ... OK!
Creating file 2.h2w ... OK!
Creating file 3.h2w ... OK!
Creating file 4.h2w ... OK!
Creating file 5.h2w ... OK!
Creating file 6.h2w ... OK!
Creating file 7.h2w ... OK!
Creating file 8.h2w ... OK!
Free space: 0.00 Byte
Average writing speed: 4.64 MB/s

$ ./f3read /media/michel/8A34-CED2/
                  SECTORS      ok/corrupted/changed/overwritten
Validating file 1.h2w ... 2097152/        0/      0/      0
Validating file 2.h2w ... 2097152/        0/      0/      0
Validating file 3.h2w ... 2097152/        0/      0/      0
Validating file 4.h2w ... 2097152/        0/      0/      0
Validating file 5.h2w ... 2097152/        0/      0/      0
Validating file 6.h2w ... 2097152/        0/      0/      0
Validating file 7.h2w ... 2097152/        0/      0/      0
Validating file 8.h2w ... 1763608/        0/      0/      0

  Data OK: 7.84 GB (16443672 sectors)
Data LOST: 0.00 Byte (0 sectors)
           Corrupted: 0.00 Byte (0 sectors)
    Slightly changed: 0.00 Byte (0 sectors)
         Overwritten: 0.00 Byte (0 sectors)
Average reading speed: 16.54 MB/s
```

As reported by `f3write/f3read` above, the real memory of my fake drive is in good shape. But it may not be the case for yours. For example, the following is `f3read`'s output for another 16GB drive with real size of 7GB fixed as described in this section:

```
                  SECTORS      ok/corrupted/changed/overwritten
Validating file 1.h2w ... 2097152/        0/      0/      0
Validating file 2.h2w ... 2097152/        0/      0/      0
Validating file 3.h2w ... 2097088/       64/      0/      0
Validating file 4.h2w ... 2097152/        0/      0/      0
Validating file 5.h2w ... 2088960/     8192/      0/      0
Validating file 6.h2w ... 2097152/        0/      0/      0
Validating file 7.h2w ... 2037632/        0/      0/      0

  Data OK: 6.97 GB (14612288 sectors)
Data LOST: 4.03 MB (8256 sectors)
           Corrupted: 4.03 MB (8256 sectors)
    Slightly changed: 0.00 Byte (0 sectors)
         Overwritten: 0.00 Byte (0 sectors)
Average reading speed: 946.46 KB/s
```

If you get some sectors corrupted, repeat the `f3write/f3read` test. Some drives recover from these failures on a second full write cycle. However, if the corrupted sectors persist, the drive is junk because not only is it a fake drive, but its real memory is already failing.

Good luck!

CHAPTER 6

History

I started this project when I bought a 32GB microSDHC card for my Android phone back in 2010, and found out that this card always fails when one fills it up. Googling about this issue, I arrived at the blogs Fight Flash Fraud and SOSFakeFlash, which recommend the software H2testw (see here or here) to test flash memories.

I downloaded H2testw and found two issues with it: (1) it is for Windows only, and (2) it is not open source. However, its author, Harald Bögeholz, was kind enough to include a text file that explains what it does, and provided the pseudo random number generator used in H2testw.

F3 is my GPLv3 implementation of the algorithm of H2testw, and other tools that I have been implementing to speed up the identification of fake drives as well as making them usable: `f3probe`, `f3fix`, and `f3brew`. My implementation of H2testw, which I've broken into two applications named `f3write` and `f3read`, runs on Linux, Macs, Windows/Cygwin, and FreeBSD. `f3probe` is the fastest way to identify fake drives and their real sizes. `f3fix` enables users to use the real capacity of fake drives without losing data. `f3brew` helps developers to infer how fake drives work. `f3probe`, `f3fix`, and `f3brew` currently runs only on Linux.

6.1 Change log

Starting at version 2.0, F3 supports the platform Mac. Mac users may want to check out Thijs Kuipers' page for help.

Starting at version 3.0, F3 supports the platform Windows/Cygwin, and adopts H2testw's file format. People interested in exchanging files between F3 and H2testw should read the *section* about it to understand the caveats.

Starting at version 4.0, F3 supports the platform FreeBSD. **Mac users:** Version 4.0 does not compile on Macs. The issue has been fixed on version 5.0.

Starting at version 5.0, F3 includes `f3probe` and `f3fix` as experimental, and for Linux only.

Starting at version 6.0, F3 includes `f3brew` as experimental, and for Linux only. Linux users may want to check out Vasiliy Kaygorodov's page or Ahmed Essam's page for help.

Starting at version 7.0, `f3probe`, `f3fix`, and `f3brew` are stable. They are for Linux only.

CHAPTER 7

Contribute

7.1 Help wanted

I maintain this project in my spare time, and I no longer have been able to answer all questions and address all feedback that I receive. Nevertheless, I still think that F3's users can stop flash counterfeiters, but I need a little bit of help to keep moving forward. How can you help?

- If F3 has helped you, consider signaling this to other users staring F3 on GitHub. The larger the number of stars a project has, the more confident new users are to try it out.
- If you know a journalist, or are one, suggest to him or her writing an article about fake flash. The media has not been covering this subject, and having more users aware that fake flash exists will make counterfeiters' life harder.
- If you own fake flash, consider donating them to me. I've been working on `f3probe` to tell in a few seconds if a flash drive is fake or not. I expect that `f3probe` will be a game changer, but I lack a diverse set of fake flash samples to check my hypotheses. Before you mail me the card, e-mail me the output you got from `f3write`, `f3read`, and (if possible) `f3probe` as well as the size the card was expected to be; I'm trying to amass a diverse set of fake flash, not necessarily a large number of them.
- If you know how to use F3 well on your platform, write a blog entry about it, and let me know the link so I can post it on this page. You can also send me your name and e-mail to publish on this page as someone that can help other users of your platform.
- (New) If you have a dual-boot machine with Windows and Linux, write a blog entry that compares h2testw, f3write/f3read, FakeFlashTest.exe, and f3probe. If you do this comparison, please send me the link to publish it on this page.
- If you are able to test F3 on your platform, send me your name and e-mail, and I'll send to you a request to test new code when it is available on GitHub before I release it as stable to everyone. I only have Linux boxes, so other platforms are specially helpful.
- If you are able to code, improve F3. I want to keep the code small to easily audit it, so focus on things that will either address a real need, for example, users' requests, or to simplify the code, or fix bugs, or make F3 work on a new platform, or improve the documentation. If you want, I can list your contact on this page as well, so people can reach out to you directly.

- If you know how to code a graphic user interface, please create one for the platforms you can. This would increase the number of users that, in turn, would win ground against the counterfeiters. I'll add a link to your application on this page.
- Tell your friends about F3, teach them how to use it, publish a video about F3, find ways to help me to better organize our efforts, spread the word, ask for your money back when you buy a fake drive, etc.

I've originally implemented F3 to address a personal need, but you have turned it into a great tool. Let's work closer to bring it to the next level!

7.2 Repository

The Git repository is kept here.

7.3 Author

Michel Machado. E-mail me at michel at digirati dot com dot br.

Please try to figure out the solution for your question on your own, or ask for help from a nearby friend before you e-mail me. My spare time is very limited. For reporting bugs, requesting features, and making suggestions, open an issue on GitHub here to allow other users to help me.

7.4 Copyright and License

CHAPTER 8

Indices and tables

- genindex
- modindex
- search