Stressant Documentation

Release ???

Antoine Beaupre

Feb 09, 2023

Contents

Stressant is a simple yet complete stress-testing tool that forces a computer to perform a series of test using well-known Linux software in order to detect possible design or construction failures.

Stressant builds on top of existing software and tries to cover most components of you system (currently disk, CPU and processor). It is part of the Grml Live Linux project and also packaged for Debian.

CHAPTER 1

Features

- disk testing (with *smartctl(8)* and *fio()*)
- CPU testing (with *stress-ng(1)*)
- network testing (with *iperf3(1)*)
- designed to be ran automatically
- supports sending reports by email or saving to disk
- basic benchmarks (with *hdparm(8)* and *dd(1)*)

See also the *Design* document for other planned features and remaining work.

Installation instructions are in the *Installation* document.

Contributions are welcome, see the *Contributor's guide* document.

CHAPTER 2

Demo

```
                        gather basic information (default: True)
  --disk, --no-disk     run disk tests (default: True)
  --no-smart, --smart   run SMART tests (default: False)
  --diskDevice DISKDEVICE
                        device to benchmark (default: /dev/sda)
  --overwrite           actually destroy the given device (default: False)
  --diskPercent DISKPERCENT
                        how much of the disk to trash (default: 0%)
  --fileSize FILESIZE   file size for I/O benchmarks (default: 100M)
  --cpu, --no-cpu       run CPU tests (default: True)
  --cpuBurnTime CPUBURNTIME
                        timeout for CPU burn-in (default: 1m)
  --network, --no-network
                        run network tests (default: True)
  --iperfServer IPERFSERVER
                        iperf server to use (default: iperf.he.net)
  --iperfTime IPERFTIME
                        timeout for iperf test, in seconds (default: 60)

Stressant is a simple yet complete stress-testing tool that forces a computer
to perform a series of test using well-known Linux software in order to detect
possible design or construction failures.
[1001]anarcat@curie:stressant$ ./stressant --cpuBurnTime 1s --fileSize 1M --iper
fTim
```

CHAPTER 3

Acknowledgements

Thanks to jrollins & dkg for writing debirf which was so useful in creating the first Stressant prototypes, and especially to dkg for help with qemu and other oddities. Thanks also to taggart for spurring this idea forward. Finally, thanks to the Grml team for being providing useful feedback on the toolset and for welcoming Stressant contributions in Grml itself.

3.1 Installation

Stressant is a Python program that can be installed in the usual way from PyPI:

```
pip install stressant
```

It can also be installed from source:

```
./setup.py install
```

If you are running Debian "buster" (10) or later, you can also install the Debian package:

```
apt install stressant
```

3.1.1 Downloading images

Stressant is also designed to boot from a live CD or USB stick. It can also boot from the network with a proper configuration.

Pre-built images are available in the Grml Linux Linux distribution, in the "FULL" images. In particular, see the daily images for the latest.

3.1.2 Installing the image

You can deploy the ISO on a USB stick with the following command on most UNIX systems, where `/dev/sdX` is the device for your USB key:

```
sudo cp grml96-full_sid_latest.iso /dev/sdX
```

To get better progress information, you can use the fantastic `pv` package:

```
pv grml96-full_sid_latest.iso | sudo dd of=/dev/sdX conv=fdatasync
```

For other operating systems, you may try those different options:

- For *Debian*: see the official Debian documentation
- For *Ubuntu*: try the Ubuntu Startup Disk Creator
- For *Fedora* and derivatives: try the Media Writer or follow those instructions
- For *Microsoft Windows*: use the win32diskimager software
- For all platforms: maybe also try Etcher if the above is too complicated

  Note: it seems that Mac OS is constantly changing how this works. There used to be support for burning ISO images directly in the Disk Utility, but that feature was removed. You may want to try this approach if Etcher doesn't work for you.

  Note: this should be moved in the main Grml documentation.

3.1.3 Booting images using PXE

Inside the ISO file, Grml is build as a squashfs and one can use the `fetch=URL` argument to fetch that squashfs over the network. Those german instructions show how to extract the squashfs from the image and set it up in a PXE menu, but basically, what you need is:

```
kernel grml/2008.11/small-linux26
append initrd=grml/2008.11/small-minirt26.gz boot=live fetch=http://grml.example.com/
↪grml/grml-small.squashfs
```

The `small-*` kernel and initrd come from the grml-terminalserver project.

Netbooting images is very useful in all sorts of situations. While Grml's grml-terminalserver tool helps in automatically configure a PXE server, you may want to configure your own, more complete setup. See for now the Koumbit wiki for PXE configuration documentation.

I have done xperimentes to boot ISO images over the network but those have been generally unsuccessful.

Note: this should be moved in the main Grml documentation.

Also note that you can directly boot into Grml and/or Stressant using a tool like netboot.xyz and iPXE.

3.1.4 Testing images in qemu

A good way to test Stressant without having to reboot your computer is with an emulator like Qemu. This will load the 64 bit image in a KVM-enabled 512 MB image running only on the serial console:

```
qemu-system-x86_64 -m 512 -enable-kvm \
    -serial mon:stdio -display none \
    grml96-full-stressant.iso
```

See also the Qemu cheat sheet in the Koumbit wiki.

3.2 stressant manual page

3.2.1 Synopsis

stressant [-h] [–version] [–log [PATH]] [–email EMAIL] [–smtpserver HOST] [–smtpuser USERNAME] [–smtppass PASSWORD] [–no-information] [–no-disk] [–smart] [–diskDevice PATH] [–jobFile PATH] [–overwrite] [–writeSize SIZE] [–directory PATH] [–diskRuntime DISKRUNTIME] [–no-cpu] [–cpuBurnTime TIME] [–no-network] [–iperfServer HOST] [–iperfTime TIME]

3.2.2 Description

Stressant is a simple yet complete stress-testing tool that forces a computer to perform a series of test using well-known Linux software in order to detect possible design or construction failures.

3.2.3 Options

-h, --help show this help message and exit

--version show program's version number and exit

--logfile <PATH> write reports to the given logfile (default: stressant-$HOSTNAME.log)

--email EMAIL send report by email to given address

--smtpserver HOST SMTP server to use, use a colon to specify the port number if non-default (25). willl attempt to use STARTTLS to secure the connexion and fail if unsupported (default: deliver using the –mta command)

--smtpuser USERNAME username for the SMTP server (default: no user)

--smtppass PASSWORD password for the SMTP server (default: prompted, if –smtpuser is specified)

--no-information, --information gather basic information (default: True)

--no-disk, --disk run disk tests (default: True)

--smart, --no-smart run SMART tests (default: False)

--diskDevice PATH device to benchmark (default: /dev/sda)

--jobFile PATH path to the fio job file to use (default: /usr/share/doc/fio/examples/basic-verify.fio)

--overwrite actually destroy the given device (default: False)

--writeSize SIZE size to write to disk, bytes or percentage (default: 100M)

--directory PATH directory to perform file tests in, created if missing (default: None)

--diskRuntime DISKRUNTIME upper limit for disk benchmark (default: 1m)

--no-cpu, --cpu run CPU tests (default: True)

--cpuBurnTime TIME timeout for CPU burn-in (default: 1m)

--no-network, --network run network tests (default: True)

--iperfServer HOST iperf server to use (default: iperf.he.net)

--iperfTime TIME timeout for iperf test, in seconds (default: 60)

3.2.4 Examples

Small run load with defaults:

```
stressant
```

Very fast test, useful to run if you are worried about crashing the machine:

```
stressant --writeSize 1M --cpuBurnTime 1s --iperfTime 1
```

Extensive test with complete disk wipe and SMART long test:

```
sudo stressant --writeSize 100% --overwrite --cpuBurnTime 24h --smart
# wait for the prescribed time, then show the SMART test results:
sudo smartctl -l selftest
```

Network test only on dedicated server:

```
stressant --no-information --no-cpu --no-disk --iperfServer iperf.example.net
```

Send test results by email:

```
stressant --email person@example.com
```

If the mail server refuses mail from your location, you can use another relay (password will be prompted):

```
stressant --email person@example.com --smtpserver submission.example.net --smtpuser␣
↪person --smtppassword
```

The `stressant-meta` package also depends on other tools that are not directly called by the automated script above, but are documented below. The meta-package also suggests many more useful tools.

Wiping disks

Danger: Wiping disks, just in case it's not totally obvious, will **DELETE DATA** on the given file or device. **DO NOT** run *ANY* command in this section unless you are sure you are writing to the **CORRECT DEVICE** and that you **REALLY** want to **DESTROY DATA**.

As mentioned above, the `stressant` commandline tool can be used to directly wipe a disk with the *`fio(1)`* command which is actually a disk-testing command that is abused for that purpose. You may not have *`fio(1)`* installed on your machine, however, so you may also use the venerable *`badblocks(8)`* command to test disks, without wiping them:

```
badblocks -nsv /dev/sdc
```

You can also wipe disks with the `-w` flag:

```
badblocks -wsv /dev/sdc
```

Be aware, however, that the effect of this will vary according to the physical medium. For example, data may be recovered old spinning hard drives (HDD) if only the above technique is used. For that purpose, you should use a tool like *nwipe(1)* that erases disks using multiple passes and patterns:

```
nwipe --autonuke --nogui --method=random --verify=off --logfile=nwipe.log /dev/sdc
```

Those tools are also ineffective on solid state drives (SSD) as they have a more complex logic layer and different layout semantics. For this, you need to use a "ATA secure erase" procedure using the *hdparm(8)* command:

```
hdparm --user-master u --security-set-pass Eins /dev/sdc
time hdparm --user-master u --security-erase Eins /dev/sdc
```

More information about this procedure is available in the ATA wiki.

Note: The "secure erase" procedure basically delegates the task of erasing the data to the disk controler. Nothing garantees the destruction of that data, short of physical destruction of the drive. See this discussion for more information.

Benchmarking disks

A good way to test disks is to wipe them, as above, but that's obviously destructive. Sometimes you might want to just test the disk's performance by hand, without wiping anything. Stressant ships with *fio(1)* and *bonnie++(1)* for that purpose. The latter is probably the simplest to use:

```
bonnie++ -s 4G -d /mnt/disk/ -n 1024
```

Make sure the file size (`-s`) is at least twice the main memory (see `free -h`). The `/mnt/disk` directory should be writable by the current user as well.

Stressant itself, when disk tests are enabled, will run the following commands:

```
dd bs=1M count=512 conv=fdatasync if=/dev/zero of=/mnt/disk/testfile
dd bs=1M count=512 conv=fdatasync if=/mnt/disk/testfile of=/dev/null
hdparm -Tt /dev/disk
smartctl -t long /dev/disk
```

Those provide a quick overview of basic disk statistics as well.

More elaborate workloads can be done with `fio`. A simple benchmark could be:

```
fio --name=stressant --group_reporting --directory=/mnt/disk --size=100M
```

That is a basic read test. The result here, on a *Western Digital Blue M.2 500GB Internal SSD (WDS500G1B0B)* with LUKS encryption, LVM and ext4, looks like:

```
Run status group 0 (all jobs):
   READ: bw=267MiB/s (280MB/s), 267MiB/s-267MiB/s (280MB/s-280MB/s), io=100MiB␣
↪(105MB), run=374-374msec

Disk stats (read/write):
    dm-3: ios=323/0, merge=0/0, ticks=484/0, in_queue=484, util=70.99%, aggrios=511/0,
↪ aggrmerge=0/0, aggrticks=764/0, aggrin_queue=764, aggrutil=76.86%
```

(continues on next page)

(continued from previous page)

```
    dm-0: ios=511/0, merge=0/0, ticks=764/0, in_queue=764, util=76.86%, aggrios=511/0,
↪ aggrmerge=0/0, aggrticks=547/0, aggrin_queue=576, aggrutil=73.55%
  sdb: ios=511/0, merge=0/0, ticks=547/0, in_queue=576, util=73.55%
```

A more realistic workload will ignore the cache (`--direct=1`), include random (`--readwrite=randrw`) or sequential writes (`--readwrite=readwrite`), and parallelize the test to put more pressure on the disk (`--numjobs=4`):

```
$ fio --name=stressant --group_reporting --directory=test --size=100M --
↪readwrite=randrw --direct=1 --numjobs=4
Run status group 0 (all jobs):
   READ: bw=45.8MiB/s (48.0MB/s), 45.8MiB/s-45.8MiB/s (48.0MB/s-48.0MB/s), io=199MiB␣
↪(209MB), run=4346-4346msec
  WRITE: bw=46.2MiB/s (48.5MB/s), 46.2MiB/s-46.2MiB/s (48.5MB/s-48.5MB/s), io=201MiB␣
↪(211MB), run=4346-4346msec

Disk stats (read/write):
    dm-3: ios=49674/50087, merge=0/0, ticks=10028/3912, in_queue=13972, util=97.22%,␣
↪aggrios=50982/51423, aggrmerge=0/0, aggrticks=10204/3852, aggrin_queue=14092,␣
↪aggrutil=96.62%
    dm-0: ios=50982/51423, merge=0/0, ticks=10204/3852, in_queue=14092, util=96.62%,␣
↪aggrios=50982/51423, aggrmerge=0/0, aggrticks=9042/2598, aggrin_queue=11224,␣
↪aggrutil=92.54%
  sdb: ios=50982/51423, merge=0/0, ticks=9042/2598, in_queue=11224, util=92.54%
```

There is, of course, way more information shown by the default fio output, including latency distribution, but those are the numbers people first look for.

Parameters can be stored in a job file, passed as an argument to fio. Examples are available in `/usr/share/doc/fio/examples`.

Stressant itself actually runs the equivalent of this:

```
fio --name=stressant --group_reporting --runtime=1m <(sed /^filename=/d /usr/share/
↪doc/fio/examples/basic-verify.fio ; echo size=100m) --filename=test
```

Note: There are many other ways to test disks, obviously. In particular, simple tools like disk-filltest might be considered for inclusion in the future, provided they enter Debian.

Testing disks

The above actually tests disks in the sense that it looks at its performance, but it's more a benchmark than a "test". For tests, stressant will do a `smartctl` run if the `--smart` argument is provided. What it actually does is:

```
smartctl -t long /dev/sdX
```

Then `smartctl -l selftest /dev/sdX` can be used to track progress.

But this doesn't work for all drives. For example, it may fail for external USB enclosures.

smartmontools's NVMe support is particularly limited. nvme-cli might be able to deal with those drives better. In *theory*, it should support running tests with:

```
nvme device-self-test /dev/nvme0
```

But in practice, that often fails because the devices sometimes do not support self-test at all. You can look at the `smart-log` instead:

```
nvme smart-log /dev/nvme0
```

If the `num_err_log_entries` entry is non-zero, you can look at the actual log:

```
nvme error-log /dev/nvme0
```

Note: The above doesn't detail how to interpret the output of those commands, and probably should. Sorry.

Testing flash memory

Flash memory cards are known to sometimes be "fake", that is, they misreport the actual capacity of the card or the bandwith available. The stressant distribution therefore recommends a tool called f3 which allows you to perform tests on the memory card. For example, this is a probe on a honest memory card:

```
$ sudo f3probe --destructive --time-ops /dev/sdb
F3 probe 6.0
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.

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

Device geometry:
             *Usable* size: 30.00 GB (62916608 blocks)
            Announced size: 30.00 GB (62916608 blocks)
                   Module: 32.00 GB (2^35 Bytes)
    Approximate cache size: 0.00 Byte (0 blocks), need-reset=no
       Physical block size: 512.00 Byte (2^9 Bytes)

Probe time: 4'57"
 Operation: total time / count = avg time
      Read: 3.07s / 4815 = 637us
     Write: 4'51" / 4192321 = 69us
     Reset: 324.5ms / 1 = 324.5ms
```

Warning: As the `--destructive` flag hints, this will *destroy* the data on the card, so backup the data elsewhere before doing those tests.

Note that older versions of *f3probe(1)* (6.0 or earlier) will have trouble doing its job unless the card is connected through a USB reader. Newer versions can deal with normal block devices, provided that you pass the magic `--reset-type=2` argument. Here's such an example, on a fake MicroSD card that is labeled and announced as 32GB but is actually closer to 16GB:

```
root@curie:/home/anarcat/backup# ~anarcat/dist/f3/f3probe --destructive --time-ops --
↪reset-type=2  /dev/mmcblk0
F3 probe 6.0
Copyright (C) 2010 Digirati Internet LTDA.
```

(continues on next page)

(continued from previous page)

```
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/mmcblk0' is a counterfeit of type limbo

You can "fix" this device using the following command:
f3fix --last-sec=30983327 /dev/mmcblk0

Device geometry:
             *Usable* size: 14.77 GB (30983328 blocks)
            Announced size: 31.25 GB (65536000 blocks)
                   Module: 32.00 GB (2^35 Bytes)
    Approximate cache size: 7.00 MB (14336 blocks), need-reset=no
       Physical block size: 512.00 Byte (2^9 Bytes)

Probe time: 2'29"
 Operation: total time / count = avg time
      Read: 1.57s / 32937 = 47us
     Write: 2'27" / 200814 = 736us
     Reset: 2us / 2 = 1us
```

To repair the device, you can repartition it quickly with the *f3fix(1)* command, as recommended in the output:

```
f3fix --last-sec=30983327 /dev/mmcblk0
```

You will also need to reformat the partition so the new size is taken into account, for example if this is a FAT32 filesystem:

```
mkfs.fat /dev/mmcblk0p1
```

You can also perform bandwidth tests with *f3read(1)* and *f3write(1)*:

```
pmount /dev/sdb1
f3write /media/sdb1
f3read /media/sdb1
```

This allows you to detect hidden caches and fake sizes directly as well.

Network performance testing

The `--iperfServer` option of stressant runs a bandwidth test against a predefined (or specified) server. You can, of course, call iPerf directly to run your own client/server tests to find issues in specific routes on the network. The `iperf3` package was chosen over the older `iperf` because public servers are available for the test to work automatically. `iperf3` also has interesting performance features like `--zerocopy` and `--file`, see *iperf3(1)* for details.

To run a test, start a server:

```
iperf3 --server
```

On another machine, connect to the server:

```
iperf3 --client 192.0.2.1
```

This runs a TCP test. You can specify UDP test on the client and disable bandwidth limitations (otherwise UDP tests are limited to 1 Mbit/s):

```
iperf3 -c 192.0.2.1 --udp --bandwidth 0
```

To simulate a DDOS condition, you can try multiple clients and run the test for a longer period:

```
iperf3 -c 192.0.2.1 -u -b 0 --parallel 50 --time 30
```

3.2.5 See also

hdparm(8), *smartctl(8)*, *dd(1)*, *fio()*, *stress-ng(1)*, *iperf3(1)*

3.3 Design

Stressant is shipped in the Debian GNU/Linux distribution. It is also part of the Grml project since August 2017, so it benefits from its extensive list of utilities, which cover most of the rescue systems out there (e.g. Debian Live and Debirf, see below for a more thorough comparison).

The Grml distribution is an ISO image that can be burned on CD/DVD or copied to a USB drive, or net-bootable images. Grml can perform:

- memory tests with `memtest86`
- hardware detection and inventory with HDT

There are also many more options, for example loading to RAM or setting the system to be read-only, see the cheatcodes list for more details.

3.3.1 The stressant tool

Stressant itself is a Python program that calls other UNIX utilities, collects their output on the screen, in a logfile and/or sends it over email.

The objective of this software is to automate a basic stress-testing suite that, once started, will go through a basic CPU/memory/disk/network test framework and report any errors and failures.

This is done through the `stressant` script, which performs the following tests:

- `lshw` and `smartctl` for hardware inventory
- `dd`, `hdparm`, `fio` and `smartctl` for disk testing - `fio` can also overwrite disk drives with the proper options (`--overwrite` and `--size=100%`)
- `stress-ng` for CPU testing
- `iperf3` for network testing

Here is an example test run:

```
$ sudo ./stressant --email anarcat@anarc.at --writeSize 1M --cpuBurnTime 1s --
↪iperfTime 1
INFO: Starting tests
INFO: CPU cores: 4
INFO: Memory: 16 GiB (16715816960 bytes)
INFO: Hardware inventory
```

(continues on next page)

(continued from previous page)

```
DEBUG: Calling lshw -short
OUTPUT: H/W path             Device     Class          Description
OUTPUT: ============================================================
OUTPUT: system         Desktop Computer
OUTPUT: /0                              bus            NUC6i3SYB
OUTPUT: /0/0                            memory         64KiB BIOS
OUTPUT: /0/22                           memory         64KiB L1 cache
OUTPUT: /0/23                           memory         64KiB L1 cache
OUTPUT: /0/24                           memory         512KiB L2 cache
OUTPUT: /0/25                           memory         3MiB L3 cache
OUTPUT: /0/26                           processor      Intel(R) Core(TM) i3-6100U CPU␣
↪@ 2.30GHz
OUTPUT: /0/27                           memory         16GiB System Memory
OUTPUT: /0/27/0                         memory         16GiB SODIMM DDR4 Synchronous␣
↪2133 MHz (0.5 ns)
OUTPUT: /0/27/1                         memory         [empty]
OUTPUT: /0/100                          bridge         Skylake Host Bridge/DRAM␣
↪Registers
OUTPUT: /0/100/2                        display        HD Graphics 520
OUTPUT: /0/100/14                       bus            Sunrise Point-LP USB 3.0 xHCI␣
↪Controller
OUTPUT: /0/100/14/0          usb1       bus            xHCI Host Controller
OUTPUT: /0/100/14/0/1        scsi3      storage        USB to ATA/ATAPI Bridge
OUTPUT: /0/100/14/0/1/0.0.0  /dev/sdb   disk           500GB 00ABYS-01TNA0
OUTPUT: /0/100/14/0/1/0.0.1  /dev/sdc   disk           500GB 00ABYS-01TNA0
OUTPUT: /0/100/14/0/3                   input          Dell USB Keyboard
OUTPUT: /0/100/14/0/4                   input          Kensington Expert Mouse
OUTPUT: /0/100/14/0/7                   communication  Bluetooth wireless interface
OUTPUT: /0/100/14/1          usb2       bus            xHCI Host Controller
OUTPUT: /0/100/14.2                     generic        Sunrise Point-LP Thermal␣
↪subsystem
OUTPUT: /0/100/16                       communication  Sunrise Point-LP CSME HECI #1
OUTPUT: /0/100/17                       storage        Sunrise Point-LP SATA␣
↪Controller [AHCI mode]
OUTPUT: /0/100/1c                       bridge         Sunrise Point-LP PCI Express␣
↪Root Port #5
OUTPUT: /0/100/1c/0                     network        Wireless 8260
OUTPUT: /0/100/1e                       generic        Sunrise Point-LP Serial IO␣
↪UART Controller #0
OUTPUT: /0/100/1e.6                     generic        Sunrise Point-LP Secure␣
↪Digital IO Controller
OUTPUT: /0/100/1f                       bridge         Sunrise Point-LP LPC Controller
OUTPUT: /0/100/1f.2                     memory         Memory controller
OUTPUT: /0/100/1f.3                     multimedia     Sunrise Point-LP HD Audio
OUTPUT: /0/100/1f.4                     bus            Sunrise Point-LP SMBus
OUTPUT: /0/100/1f.6          eno1       network        Ethernet Connection I219-V
OUTPUT: /0/1                 scsi2      storage
OUTPUT: /0/1/0.0.0           /dev/sda   disk           500GB WDC WDS500G1B0B-
OUTPUT: /0/1/0.0.0/1         /dev/sda1  volume         511MiB Windows FAT volume
OUTPUT: /0/1/0.0.0/2         /dev/sda2  volume         244MiB EFI partition
OUTPUT: /0/1/0.0.0/3         /dev/sda3  volume         465GiB EFI partition
INFO: SMART information for /dev/sda
DEBUG: Calling smartctl -i /dev/sda
OUTPUT: smartctl 6.6 2016-05-31 r4324 [x86_64-linux-4.9.0-1-amd64] (local build)
OUTPUT: Copyright (C) 2002-16, Bruce Allen, Christian Franke, www.smartmontools.org
OUTPUT:
OUTPUT: === START OF INFORMATION SECTION ===
```

(continues on next page)

(continued from previous page)

```
OUTPUT: Device Model:     WDC WDS500G1B0B-00AS40
OUTPUT: Serial Number:    XXXXXXXXXXXX
OUTPUT: LU WWN Device Id: XXXXXXXXXXXX
OUTPUT: Firmware Version: XXXXXXXXXXXX
OUTPUT: User Capacity:    500,107,862,016 bytes [500 GB]
OUTPUT: Sector Size:      512 bytes logical/physical
OUTPUT: Rotation Rate:    Solid State Device
OUTPUT: Form Factor:      M.2
OUTPUT: Device is:        Not in smartctl database [for details use: -P showall]
OUTPUT: ATA Version is:   ACS-2 T13/2015-D revision 3
OUTPUT: SATA Version is:  SATA 3.2, 6.0 Gb/s (current: 6.0 Gb/s)
OUTPUT: Local Time is:    Fri Mar 17 10:24:52 2017 EDT
OUTPUT: SMART support is: Available - device has SMART capability.
OUTPUT: SMART support is: Enabled
OUTPUT:
INFO: Basic disk bandwidth tests
INFO: Writing 1MB file
DEBUG: Calling dd bs=1M count=512 conv=fdatasync if=/dev/zero of=test
OUTPUT: 512+0 records in
OUTPUT: 512+0 records out
OUTPUT: 536870912 bytes (537 MB, 512 MiB) copied, 1.39591 s, 385 MB/s
INFO: Reading 1MB file
DEBUG: Calling dd bs=1M count=512 of=/dev/null if=test
OUTPUT: 512+0 records in
OUTPUT: 512+0 records out
OUTPUT: 536870912 bytes (537 MB, 512 MiB) copied, 0.0848588 s, 6.3 GB/s
INFO: Hdparm test
DEBUG: Calling hdparm -Tt /dev/sda
OUTPUT:
OUTPUT: /dev/sda:
OUTPUT: Timing cached reads:   12406 MB in  2.00 seconds = 6207.39 MB/sec
OUTPUT: Timing buffered disk reads: 1504 MB in  3.00 seconds = 501.13 MB/sec
INFO: Disk stress test
DEBUG: Calling fio --name=stressant --readwrite=randrw --numjob=4 --sync=1 --direct=1␣
↪--group_reporting --size=1M --output=/tmp/tmpo2QJnR
OUTPUT: stressant: (g=0): rw=randrw, bs=4K-4K/4K-4K/4K-4K, ioengine=psync, iodepth=1
OUTPUT: ...
OUTPUT: fio-2.16
OUTPUT: Starting 4 processes
OUTPUT:
OUTPUT: stressant: (groupid=0, jobs=4): err= 0: pid=978: Fri Mar 17 10:25:07 2017
OUTPUT:   read : io=2160.0KB, bw=5669.3KB/s, iops=1417, runt=   381msec
OUTPUT:     clat (usec): min=141, max=2197, avg=486.59, stdev=344.70
OUTPUT:      lat (usec): min=141, max=2198, avg=486.96, stdev=344.71
OUTPUT:     clat percentiles (usec):
OUTPUT:      |  1.00th=[  145],  5.00th=[  153], 10.00th=[  161], 20.00th=[  175],
OUTPUT:      | 30.00th=[  189], 40.00th=[  217], 50.00th=[  278], 60.00th=[  676],
OUTPUT:      | 70.00th=[  748], 80.00th=[  828], 90.00th=[  916], 95.00th=[  980],
OUTPUT:      | 99.00th=[ 1384], 99.50th=[ 1800], 99.90th=[ 2192], 99.95th=[ 2192],
OUTPUT:      | 99.99th=[ 2192]
OUTPUT:   write: io=1936.0KB, bw=5081.4KB/s, iops=1270, runt=   381msec
OUTPUT:     clat (usec): min=618, max=6602, avg=2566.13, stdev=1029.39
OUTPUT:      lat (usec): min=619, max=6602, avg=2566.71, stdev=1029.39
OUTPUT:     clat percentiles (usec):
OUTPUT:      |  1.00th=[  732],  5.00th=[  900], 10.00th=[  964], 20.00th=[ 1672],
OUTPUT:      | 30.00th=[ 1976], 40.00th=[ 2384], 50.00th=[ 2640], 60.00th=[ 3152],
OUTPUT:      | 70.00th=[ 3312], 80.00th=[ 3440], 90.00th=[ 3568], 95.00th=[ 3856],
```

(continues on next page)

(continued from previous page)

```
OUTPUT:       | 99.00th=[ 4704], 99.50th=[ 4960], 99.90th=[ 6624], 99.95th=[ 6624],
OUTPUT:       | 99.99th=[ 6624]
OUTPUT:      lat (usec) : 250=24.41%, 500=4.88%, 750=7.91%, 1000=19.34%
OUTPUT:      lat (msec) : 2=10.84%, 4=30.66%, 10=1.95%
OUTPUT:    cpu          : usr=0.80%, sys=2.39%, ctx=1945, majf=0, minf=35
OUTPUT:    IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
OUTPUT:       submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0
 ↪%
OUTPUT:       complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0
 ↪%
OUTPUT:       issued    : total=r=540/w=484/d=0, short=r=0/w=0/d=0, drop=r=0/w=0/d=0
OUTPUT:       latency   : target=0, window=0, percentile=100.00%, depth=1
OUTPUT:
OUTPUT: Run status group 0 (all jobs):
OUTPUT:    READ: io=2160KB, aggrb=5669KB/s, minb=5669KB/s, maxb=5669KB/s,␣
 ↪mint=381msec, maxt=381msec
OUTPUT:   WRITE: io=1936KB, aggrb=5081KB/s, minb=5081KB/s, maxb=5081KB/s,␣
 ↪mint=381msec, maxt=381msec
OUTPUT:
OUTPUT: Disk stats (read/write):
OUTPUT:     dm-3: ios=207/493, merge=0/0, ticks=120/296, in_queue=416, util=58.78%,␣
 ↪aggrios=540/1527, aggrmerge=0/0, aggrticks=288/752, aggrin_queue=1040, aggrutil=75.
 ↪30%
OUTPUT:     dm-0: ios=540/1527, merge=0/0, ticks=288/752, in_queue=1040, util=75.30%,␣
 ↪aggrios=540/1326, aggrmerge=0/201, aggrticks=264/704, aggrin_queue=968, aggrutil=74.
 ↪49%
OUTPUT:   sda: ios=540/1326, merge=0/201, ticks=264/704, in_queue=968, util=74.49%
INFO: CPU stress test for 1s
DEBUG: Calling stress-ng --timeout 1s --cpu 0 --ignite-cpu --metrics-brief --log-
 ↪brief --tz --times --aggressive
OUTPUT: dispatching hogs: 4 cpu
OUTPUT: cache allocate: default cache size: 3072K
OUTPUT: successful run completed in 1.05s
OUTPUT: stressor       bogo ops real time  usr time  sys time   bogo ops/s   bogo ops/
 ↪s
OUTPUT: (secs)    (secs)    (secs)   (real time) (usr+sys time)
OUTPUT: cpu                453      1.04      2.80      0.00       437.62       161.
 ↪79
OUTPUT: cpu:
OUTPUT: acpitz   27.80 °C
OUTPUT: pch_skylake   32.77 °C
OUTPUT: acpitz   31.78 °C
OUTPUT: x86_pkg_temp   34.40 °C
OUTPUT: for a 1.05s run time:
OUTPUT: 4.22s available CPU time
OUTPUT: 2.81s user time   ( 66.64%)
OUTPUT: 0.01s system time (  0.24%)
OUTPUT: 2.82s total time  ( 66.87%)
OUTPUT: load average: 0.34 0.58 2.52
INFO: Running network benchmark
DEBUG: Calling iperf3 -c iperf.he.net -t 1
OUTPUT: iperf3: error - the server is busy running a test. try again later
ERROR: Command failed: Command 'iperf3 -c iperf.he.net -t 1' returned non-zero exit␣
 ↪status 1
INFO: all done
INFO: sent email to ['anarcat@anarc.at'] using anarc.at
```

Note that there are nice colors in an actual console, the above is just a dump of the logfile.

We currently use the `iperf.he.net` server from Hurricane Electric as a default server for our tests, but users are encouraged to change that to a local server using the `--iperfServer` argument to get more accurate results. Notice how that performance test failed, above, because the HE server wasn't available: this is just another hint that you should use your own server.

A number of public iPerf servers are available, here are a few lists:

- <https://iperf.fr/iperf-servers.php>
- <https://github.com/R0GGER/public-iperf3-servers>
- <https://proof.ovh.ca/>

3.3.2 Background

This project emanates from a packaging effort of a custom Linux distribution called 'breakin'. It turned into a simple Python program that reuses existing stress-testing programs packaged in Debian.

Stressant *used* to be built as a standalone Debian Derivative, a Pure blend based on Debirf. But in 2017, the project was rearchitectured to be based on Grml and to focus on developing a standalone stress-testing tool. While Stressant could still become its own Debian derivative, it seems futile for now to make yet another Debian derivative. Instead, we focus our energy into contributing to the Grml project without needing to create the heavy infrastructure for another Linux distribution.

The Is your Computer Stable? post from Jeff Atwood was a motivation to get back into the project. It outlines a few basic tools to use to make sure your computer is stable:

- memtest86 - shipped with Grml
- install Ubuntu - we assume you'll do that anyways
- MPrime to stress the CPU - not free software, [stress-ng][] was chosen instead and gives similar results here
- badblocks test (with `-sv`) - this is covered by the `fio` test
- smartctl -i/-a/-t to identify and test harddrives
- `dd` and `hdparm` to get quick stats - done
- bonnie++ for more extensive benchmarks - Grml people suggested we use fio instead
- iperf for network testing - this assumes a local server, we instead use iperf3 and public servers
- furmark for testing the GPU - Windows-only, no Linux equivalent, the Phoronix test suite uses ffmpeg tests for that purpose

The idea was to regroup this in a single tool that would perform all those tests, without reinventing the wheel of course.

Stressant was also highly coupled with Koumbit's infrastructure as this is where the Debirf recipes were originally developed. It needed a CI system to build the images, which was originally done with Jenkins. This was then done with Gitlab CI, but failed to build images because of issues with debirf and, ultimately, docker itself. This is why why Grml was used as a basis for future development.

3.3.3 Remaining work

Stressant could run in a tmux or screen session that would show the current task on one pane and syslog (or journalctl) in another. This would allow for more information to be crammed in a single display while at the same time making remote access (e.g. through SSH) easier to switch to.

Note: Parallism is discussed as part of a larger redesign in issue #3.

Finally, we need clear and better documentation on various testing tools there are out there, a bit like TAILS is doing. For example, we used to ship with `diskscan` but i didn't even remember that and I am not sure what to use it for or when to use it. A summary description of the available tools, maybe through a menu system or at least a set of HTML files, would be useful. I use Sphinx and RST for this because of the simplicity and availability of tools like readthedocs.org and the ease of creationg for offline documentation (PDF and ePUB). A rendered copy of the documentation is available on stressant.readthedocs.io and in the `stressant-doc` package. The metapackage (`stressant-meta`) lists the relevant recovery tools and some of those are documented in the *stressant manual page*.

3.3.4 Similar software

In the meantime, here's a list of software that's similar to stressant or that could be used by stressant.

Test suites

Those are fairly similar to stressant in that they perform multiple benchmarks:

- Breakin - stress-test and hardware diagnostics tool
- Checkbox - Ubuntu's certification tool, shipped with Debian stretch but removed because upstream switched to snaps, new RFP
- Inquisitor - hardware testing suite
- OpenBenchmarking.org - a good source of benchmarking tools
- PerfKit Benchmarker - GCP's benchmarking tool
- Phoronix test suite - far-ranging benchmarking suite
- Stressapptest - Stressful Application Test, userspace memory and IO test - similar to stressant
- bench-scripts - a review of many benchmarking scripts that provide a nice and simple interface for basic benchmarks
- sys_basher - another stress-testing tool
- Ars Technica - has a interesting post detailing a few key fio commands that should be ran

Purpose-specific tools

- chipsec - framework for analyzing the security of PC platforms including hardware, system firmware (BIOS/UEFI), and platform components
- FWTS - Ubuntu's Firmware Test Suite - performs sanity checks on Intel/AMD PC firmware. It is intended to identify BIOS and ACPI errors and if appropriate it will try to explain the errors and give advice to help workaround or fix firmware bugs
- Power Stress and Shaping Tool (PSST) - a "controlled power tool for Intel SoC components such as CPU and GPU. PSST enables very fine control of stress function without its own process overhead", packaged in Debian as psst
- The stress terminal (s-tui) - mostly for testing CPU, temperature and power usage, now included in the metapackage

- tinymembench - memory bandwidth userland tester
- stressdisk - "Stress test your disks / memory cards / USB sticks before trusting your valuable data to them"

3.3.5 Building images by hand

Note: Starting from August 2017, stressant is part of Grml, and it's usually superfluous to build your own image unless you're into that kind of kinky stuff. Those notes are kept mostly for historical purposes.

There is a handy `build-iso.sh` script that will setup APT repositories and run all the right commands to build a Grml Stressant ISO image on a recent Debian release (tested on Jessie and stretch). Note that you can pass extra flags to the grml-live command with the `$GRML_LIVE_FLAGS` environment variable. What follows is basically a description of what that script does.

To build an image by hand, you will need to first install the `grml-live` package which is responsible for building Grml images. For this, you will need to add the Grml Debian repository to your `sources.list` file. Instructions for doing so are available in the files section of the Grml site.

Once this is done, you should be able to build an image using:

```
sudo grml-live -c DEBORPHAN,GRMLBASE,GRML_FULL,RELEASE,AMD64,IGNORE,STRESSANT \
   -s unstable -a amd64 \
  -o $PWD/grml -U $USER \
  -v $(date +%Y.%m) -r gossage -g grml64-full-stressant
```

This will build a "full" Grml release (`-c`) based on Debian unstable on a 64 bit architecture (`-a`) in the `./grml` subdirectory (`-o`). The files will be owned (`-U`) by the current user (`$USER`). The version number (`-v`), the release name (`-r`) and flavor (`-g`) are just cargo-culted from the upstream official release. See the grml-live for further options, but do note the `-u` option that can be used to rerun the builds if you want to only update the image to the latest release, for example.

The resulting ISO will be in `./grml/grml_isos/grml64-full_$(date +%Y.%m%d).iso`. To make a multi-arch ISO, you should use the `grml2iso` command. For example, this is how upstream builds the `96` ISO which features the 32 bits and 64 bits architectures:

```
grml2iso -o grml96-small_2014.11.iso grml64-small_2014.11.iso grml32-small_2014.11.iso
```

3.3.6 Build system review

The following is a summary evaluation of the different options considered by the Stressant project to build live images. This problem space is currently in flux: at the time of writing, the tools used to build the Debian Live images are changing and the future of the project was uncertain. Keep this in mind when you read this in the future. Here are the options that were considered, with a detailed evaluation below:

- *DebIRF*
- *FAI*
- *vmdebootstrap*
- *live-build and live-wrapper*
- *Grml*

- *mkosi*

The Debian cloud team also considered a few tools to generate their cloud images, and some are relevant here (FAI and vmdebootstrap), see this post for details. There's also this more exhaustive list of tools to build Debian systems.

DebIRF

Debirf stands for Debian InitRamFs and builds the live image into the `initrd` file. It was originally used by Stressant because it was simple and easy to modify. It also allowed booting from the network easily, as we only had to load the kernel and didn't have to bother with ISO images loading or NFS, like other options.

In the end, however, Debirf proved to be too limited for our needs: it doesn't provide a way to embed boot-level, arbitrary binaries like memtest86 because it is too tightly coupled with the Linux kernel. Furthermore, we were having serious issues building debirf images in newer releases, either in Debian 8 (bug #806377) or 9 (bug #848834).

FAI

I have tried to use FAI to follow the lead of the Debian cloud team. Unforunately, I stumbled upon a few bugs. First, fai-diskimage would fail to build an image if the host uses LVM. This was fixed in FAI 5.3.3 (or maybe 5.3.4?). Also, FAI seems to fetch base files from a cleartext URL, which seems like a dubious security choice.

After finding this tutorial, I figured I would give it a try again. Unfortunately, after asking on IRC (`#debian-cloud` on OFTC), I was told (by Noah!) that "*fai-diskimage is probably not what you want for an iso image*" and they suggested I use `fai-cd`. Unfortunately, `fai-cd` works completely differently: it doesn't support the `--class` system that `fai-diskimage` was built with, so we can't reuse those already mysterious recipes. `fai-cd` seems to be built towards creating install medium and not live images.

All this seems to make FAI mostly unusable for the task at hand, although it *should* be noted that grml-live uses FAI to build their images...

vmdebootstrap

vmdebootstrap is a minimal image building tool, written in Python, used for Debian live images. It requires root (for loop filesystems creation), and doesn't support shipping the Debian installer anymore.

We have had good results with `vmdebootstrap`, but the fact that it requires a loop device has made it difficult to use Gitlab's CI system. Docker has a bug that makes it impossible to use loop devices and `kpartx` commands in it. So to build images through Gitlab's CI would require full virtualization instead of just Docker, something that's not provided by Gitlab.com right now. This problem is probably shared by all image building tools, however.

Worse: VirtualBox did not make it to stretch at all which makes it difficult to deploy new builders for it.

live-build and live-wrapper

live-build is a set of tools used by Debian live and other blends (e.g. PGP cleanroom uses live-build). It used to be a set of shell scripts, but it now uses live-wrapper which uses vmdebootstrap in the end.

It is unclear if I am better off using live build or vmdebootstrap directly. The PGP cleanroom build uses live build, so maybe I should do that as well...

Notes: live-wrapper is where the idea of using HDT comes from. Unfortunately, it looks like the boot menus don't actually work yet (bug #813527). Furthermore, live-wrapper doesn't support initrd-style netboot, so we would need documentation on how to boot from ISO files over PXE.

Grml

Grml is a project quite similar to the original goal of stressant:

- based on Debian
- provides rescue tools
- live CD/USB image
- also provides support for netboot through grml-terminalserver, which can use remote squashfs

The project is really interesting and we have therefore switched the focus of Stressant towards creating an integrated stress-test tool on *top* of grml instead of trying to fix all the issues those guys are already struggling with... We use grml-debootstrap, a tool similar to vmdebootstrap, to build stressant images.

Grml has most of the packages we had in our dependencies, except those:

```
blktool
bonnie++
chntpw
diskscan
e2tools
fatresize
foremost
hfsplus
i7z
lm-sensors
mtd-utils
scrub
smp-utils
stress-ng
tofrodos
u-boot-tools
wodim
```

Of those, only `stress-ng` is *actually* required by `stressant`.

The remaining issues with Grml integration are:

1. add stressant to the Grml build (pull request #34 - done!)
2. review the above packages we collected from various rescue modes and see if they are relevant
3. hook stressant in the magic Grml menu to start directly from the boot menu - we can use the `scripts=path-name` argument for this, it looks in the "`DCS` dir", which is `/` or whatever `myconfig=` points at
4. figure out how to chain into memtest86 to complete the test suite

Because `stressant` has been accepted into Debian, we should not need to setup our own build system, unless Grml refuses to integrate the package directly. In any case, we *may* want to setup our own Continuous Integration (CI) system to build feature branches and similar. The proper way to do this seems to be to add the `.deb` file directly at the root (`/`) of the live filesystem with the install local files technique and the `debs` boot-time argument.

mkosi

mkosi is another tool from the systemd folks which we may want to consider.

3.4 Contributor's guide

This document outlines how to contribute to this project. It details a code of conduct, how to submit issues, bug reports and patches.

3.4.1 Contributor Covenant Code of Conduct

Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting one of the persons listed below. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project maintainers is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

Project maintainers are encouraged to follow the spirit of the Django Code of Conduct Enforcement Manual when receiving reports.

Contacts

The following people have volunteered to be available to respond to Code of Conduct reports. They have reviewed existing literature and agree to follow the aforementioned process in good faith. They also accept OpenPGP-encrypted email:

- Antoine Beaupré anarcat@debian.org

Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at http://contributor-covenant.org/version/1/4

Changes

The Code of Conduct was modified to refer to *project maintainers* instead of *project team* and small paragraph was added to refer to the Django enforcement manual.

> Note: We have so far determined that writing an explicit enforcement policy is not necessary, considering the available literature already available online and the relatively small size of the community. This may change in the future if the community grows larger.

3.4.2 Patches

Patches can be submitted through merge requests on the GitLab project.

Some guidelines for patches:

- A patch should be a minimal and accurate answer to exactly one identified and agreed problem.
- A patch must compile cleanly and pass project self-tests on all target platforms.
- A patch commit message must consist of a single short (less than 50 characters) line stating a summary of the change, followed by a blank line and then a description of the problem being solved and its solution, or a reason for the change. Write more information, not less, in the commit log.
- Patches should be reviewed by at least one maintainer before being merged.

Project maintainers should merge their own patches only when they have been approved by other maintainers, unless there is no response within a reasonable timeframe (roughly one week) or there is an urgent change to be done (e.g. security or data loss issue).

As an exception to this rule, this specific document cannot be changed without the consensus of all administrators of the project.

> Note: Those guidelines were inspired by the Collective Code Construct Contract. The document was found to be a little too complex and hard to read and wasn't adopted in its entirety. See this discussion for more information.

Patch triage

You can also review existing pull requests, by cloning the contributor's repository and testing it. If the tests do not pass (either locally or in the online Continuous Integration (CI) system), if the patch is incomplete or otherwise does not respect the above guidelines, submit a review with "changes requested" with reasoning.

3.4.3 Documentation

We love documentation!

The documentation mostly in the README file and can be edited online once you register.

3.4.4 Issues and bug reports

We want you to report issuess you find in the software. It is a recognized and important part of contributing to this project. All issues will be read and replied to politely and professionnally. Issues and bug reports should be filed on the issue tracker.

3.4.5 Issue triage

Issue triage is a useful contribution as well. You can review the issues in the GitHub project and, for each issue:

- try to reproduce the issue, if it is not reproducible, label it with `more-info` and explain the steps taken to reproduce
- if information is missing, label it with `more-info` and request specific information
- if the feature request is not within the scope of the project or should be refused for other reasons, use the `wontfix` label and close the issue
- mark feature requests with the `enhancement` label, bugs with `bug`, duplicates with `duplicate` and so on...

Note that some of those operations are available only to project maintainers, see below for the different statuses.

3.4.6 Membership

There are three levels of membership in the project, Administrator (also known as "Owner" in GitHub or GitLab), Maintainer (also known as "Member" on GitHub or "Developer" on GitLab), or regular users (everyone with or without an account). Anyone is welcome to contribute to the project within the guidelines outlined in this document, regardless of their status, and that includes regular users.

Maintainers can:

- do everything regular users can
- review, push and merge pull requests
- edit and close issues

Administrators can:

- do everything maintainers can

- add new maintainers
- promote maintainers to administrators

Regular users can be promoted to maintainers if they contribute to the project, either by participating in issues, documentation or pull requests.

Maintainers can be promoted to administrators when they have given significant contributions for a sustained timeframe, by consensus of the current administrators. This process should be open and decided as any other issue.

3.4.7 Release process

To make a release:

1. generate release notes with:

   ```
   gbp dch
   ```

   the file header will need to be moved back up to the beginning of the file. also make sure to add a summary and choose a proper version according to Semantic Versioning

2. tag the release according to Semantic Versioning rules:

   ```
   git tag -s x.y.z
   ```

3. build and test the Python package:

   ```
   python3 setup.py bdist_wheel
   sudo pip3 install dist/*.whl
   stressant --version
   sudo stressant --email anarcat@anarc.at --logfile test.log --writeSize 1M --
   ↪cpuBurnTime 1s --iperfTime 1
   # check your emails and the logfile
   sudo pip3 uninstall stressant
   ```

4. build and test the debian package:

   ```
   git-buildpackage
   sudo dpkg -i ../stressant_*.deb
   stressant --version
   sudo stressant --email anarcat@anarc.at --logfile test.log --writeSize 1M --
   ↪cpuBurnTime 1s --iperfTime 1
   sudo dpkg --remove stressant
   ```

5. push changes:

   ```
   git push
   git push --tags
   twine upload dist/*
   dput ../stressant*.changes
   ```

6. edit the tag on Gitlab, copy-paste the changelog entry and attach the signed binaries

3.5 History

```
stressant (0.7.0) unstable; urgency=medium

  [ Antoine Beaupré ]
  * documentation updates:
    * add ars fio commands
    * mention nvme drives
    * move README in the right place
    * move similar software to design
    * split related projects in groups
    * add tinymembench
    * show how we actuall run a basic bench
    * checkbox is another alternative to stressant
  * new features:
    * make a basic fio test bench, only shipped in source
    * fix syntax error in docs
    * start listing iperf server lists
    * add fqdn to email subject and default logfile
    * add timestamps to email and logs
    * NegateAction: allow caller to override default
  * fixes:
    * reverse logic of default NegateAction argument
    * properly flush changes to disk before calling fio job
  * code cleanups:
    * fix branch name in gbp
    * lint: fix long line wrap

  [ Debian Janitor ]
  * Use secure copyright file specification URI.
  * Use canonical URL in Vcs-Git.
  * Update standards version to 4.5.1, no changes needed.

 -- Antoine Beaupré <anarcat@debian.org>  Thu, 09 Feb 2023 11:37:48 -0500

stressant (0.6.0) unstable; urgency=medium

  * documentation updates:
    * mention disk-filltest
    * do not verify disk wipes and use the "random" method
    * "stressant.py" was renamed to "stressant"
    * factor in NMU from plugwash, thanks!
  * code changes:
    * remove bare except, assuming we mean to catch other SMTP exceptions
    * cosmetic: harmonize fio argument order between file and dir
    * cosmetic: remove cargo-culted comments referring to previous code
  * API-breaking changes:
    * merge the two disk sizes arguments (--diskPercent and --fileSize
      merge into --writeSize)
    * deprecate the ISO build functionality, we made it into grml
  * new features:
    * cap disk tests to one minute by default (--diskRuntime)
    * bump Standards-Version to 4.5.0, no change
    * improve error messages in SMTP backoff mechanism
    * backport Python 3.7 improvements to SMTP buffering handler
  * bugfixes:
    * fix unicode decode error in Python 3.7
```

(continues on next page)

(continued from previous page)

```
    * create directory earlier, so that dd can work
    * fix job file argument so it supports jobs with filename=
    * fix error handling in SMTP handler
    * fix infinite loop in SMTP handler

 -- Antoine Beaupré <anarcat@debian.org>  Thu, 20 Feb 2020 09:43:58 -0500

stressant (0.5.0+nmu1) unstable; urgency=medium

  * Non-maintainer upload.
  * Re-upload source-only to allow migration to testing (Closes: 943867).
  * Fix clean target.

 -- Peter Michael Green <plugwash@debian.org>  Sat, 09 Nov 2019 07:50:54 +0000

stressant (0.5.0) unstable; urgency=medium

  * ship new stressant-meta package
  * add mkosi to the list of iso generators
  * update and clarify live build images section
  * add more detailed iperf tests
  * add warning and example of fake card and repair
  * tell user how to see smart results
  * run smart tests after other disk benchmarks
  * use job files instead of static fio configuration
  * rename jobfile argument to abstract away fio
  * link to debian manpages
  * use proper smartmontools dependency (Closes: #859698)
  * add more related projects
  * mention more simple disk tests
  * mention that wiping disks is a fairly good way to test them
  * more generic title for flash memory
  * include s-tui in meta
  * link to the bench.sh scripts
  * port to python3 (Closes: #938573)
  * fix --version usage
  * some linting

 -- Antoine Beaupré <anarcat@debian.org>  Sun, 27 Oct 2019 16:29:43 -0400

stressant (0.4.1) unstable; urgency=medium

  * remove traces of monkeysign
  * hide serial numbers from smartctl output
  * python packages are arch:all, not any
  * split docs in a separate package

 -- Antoine Beaupré <anarcat@debian.org>  Tue, 27 Jun 2017 22:59:36 -0400

stressant (0.4.0) unstable; urgency=medium

  * can now test in arbitrary directories with --directory option
  * fix "edit" links in docs
  * fix version parsing
  * fix version detection in sphinx build
  * fix sphinx build-dep
  * add examples to manpage
```

(continues on next page)

(continued from previous page)

```
  * explain how the usage and manpages are generated
  * deal with older colorlog modules
  * fix changelog generation with gbp

 -- Antoine Beaupré <anarcat@debian.org>  Sun, 23 Apr 2017 20:03:41 -0400

stressant (0.3.2) unstable; urgency=medium

  * hook grml build into Gitlab CI
  * various documentation updates, including:
    * review builders section
    * credit grml in acknowledgements
    * add asciinema screncast
    * convert documentation to Sphinx/RST
    * add manpage

 -- Antoine Beaupré <anarcat@debian.org>  Sun, 02 Apr 2017 21:50:04 -0400

stressant (0.3.1) unstable; urgency=medium

  * fix copyright file

 -- Antoine Beaupré <anarcat@debian.org>  Fri, 17 Mar 2017 12:02:16 -0400

stressant (0.3.0) unstable; urgency=medium

  * rebuild the whole project using Grml
  * rewrite and complete prototype in Python
  * email, logfile and color support
  * network tests with iPerf3
  * disk tests with fio
  * CPU tests with stress-ng
  * basic information gathered with lshw and smart

 -- Antoine Beaupré <anarcat@debian.org>  Fri, 17 Mar 2017 11:10:21 -0400

stressant (0.2) experimental; urgency=medium

  * switch to using vmdebootstrap to build ISO images
  * add first prototype of test run, not packaged
  * switch to native packaging
  * Initial release (Closes: #707178)

 -- Antoine Beaupré <anarcat@debian.org>  Thu, 05 Jan 2017 02:50:03 -0500
```