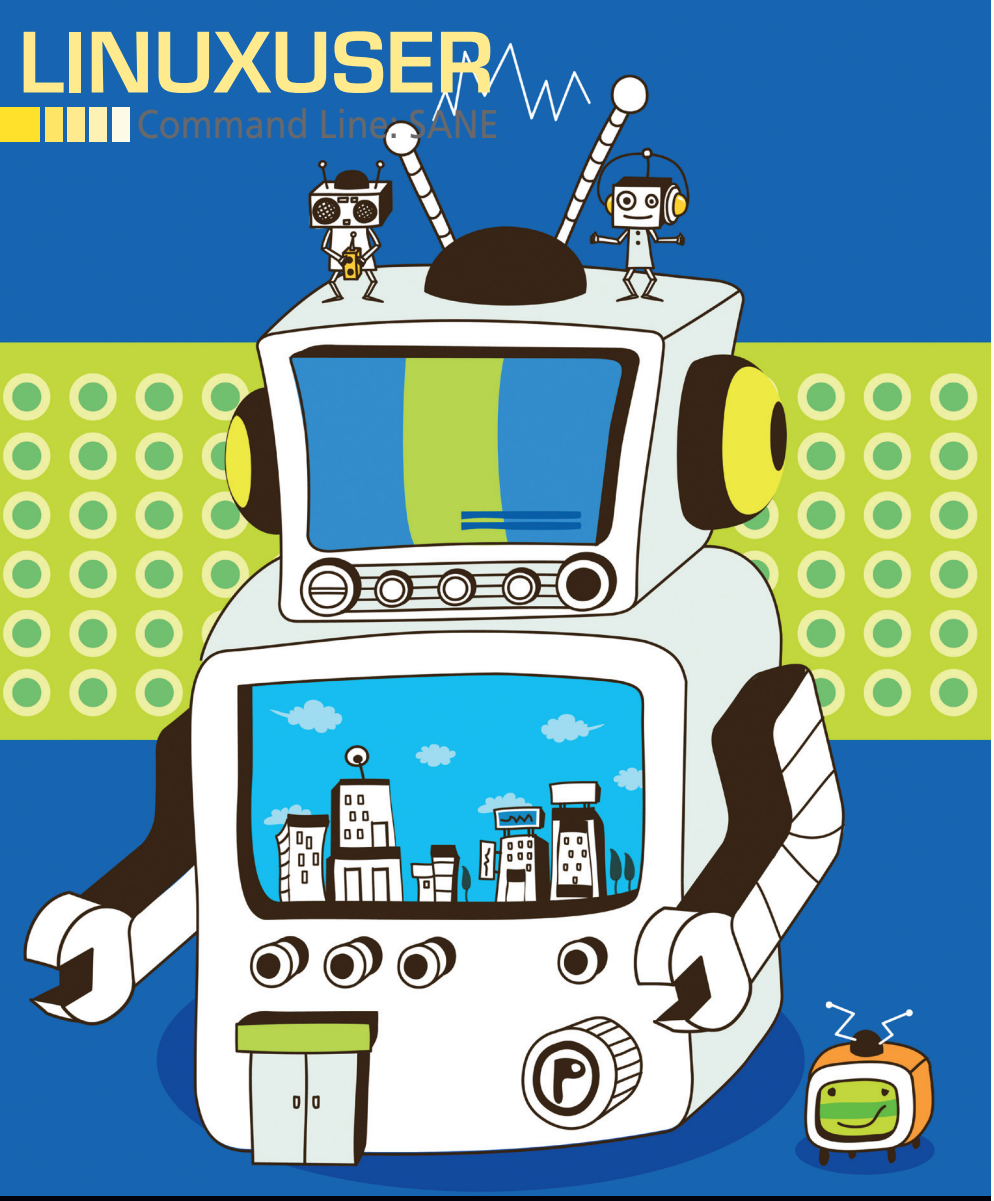

Control your scanner with SANE

Sanity Check

Running your scanner from the command line offers greater control of tasks. We show you how to get started.

By Bruce Byfield

If you have a scanner attached to your computer, then you've probably heard of SANE [1]. Short for Scanner Access Now Easy, SANE is a free software API for interacting with scanners, cameras, and frame-grabbers. SANE generally operates in the background if you are working from the desktop, but, if you open a command line, you can control your scanner with much greater precision.

If you think the acronym sounds contrived, you're right. SANE was originally written in direct response to TWAIN [2], the standard protocol for Windows and OS X computers interacting with scanners and cameras. Specifically, SANE is designed as an improvement over TWAIN – as the project home page says, "TWAIN simply isn't SANE."

SANE vs. TWAIN

This claim is not an empty boast: SANE improves on TWAIN in at least two ways. First, SANE allows Unix-like operating systems to communicate with scanners and cameras, something that TWAIN has shown little interest in.

Second, SANE is entirely concerned with communication with the peripheral devices, whereas TWAIN requires both a specific user interface and the ability to communicate with devices. In other words, TWAIN is like an application from the days of DOS, when each application required its own device driver, and SANE is like a sub-system of a modern desktop or operating system that any application can hook into.

This difference means that SANE development requires less effort: Two different applications require two TWAIN drivers, but only one SANE driver. SANE also frees developers to focus on application drivers, which is one reason why GNU/Linux has so many desktop applications for scanning – everything from XSane [3] to Simple Scan [4]. You can add scanner functionality relatively easily to an application, such as OpenOffice.org or GIMP, or create a dedicated program like gscan2pdf [5].

Generally, SANE is more flexible than TWAIN. It works more easily over networks and is easier to port to different operating systems. For such reasons, the SANE project regards itself as what TWAIN should be, but isn't.

Depending on your distribution, you might need to add users to the scanner group before you start using a scanner. Usually, this group will be called *saned*, after the SANE daemon, or perhaps *scanner*. You might also need to change the permissions on the port that the scanner users, although in most modern distributions that will likely be unnecessary.

Otherwise, you have four tools for dealing with SANE as a user: the SANE daemon, `sane-find-scanner` for detecting scanners, and `scanimage` and `scanadf` for using a scanner from the command line.

Usually, `sane-find-image`, `scanimage`, and `scanadf` are not included by a distro automatically, but you can find them in most distributions' repositories, sometimes as a single package of utilities.

Finding a Scanner

To make sure your system is detecting any installed scanner, you can use the dedicated utility `sane-find-scanner`. This command detects USB or SCSI scanners and will also detect Mustek parallel scanners. Other parallel scanners might also be detected, but the man page makes very clear that you shouldn't expect too much. However, parallel port scanners are rare these days and are confined mostly to older systems, so this limitation should affect very few users.

To locate USB scanners, `sane-fine-scanner` searches for scanner device files under `/dev/usb`. If no scanners are detected, the utility scans USB ports. Similarly, SCSI scanners are detected by probing the standard SCSI device files, `/dev/sg0` and `/dev/scanner`. These probes will normally find any USB or SCSI scanners,

```
bruce@nanday:~$ sane-find-scanner

  # sane-find-scanner will now attempt to detect your scanner. If the
  # result is different from what you expected, first make sure your
  # scanner is powered up and properly connected to your computer.

  # No SCSI scanners found. If you expected something different, make sure that
  # you have loaded a kernel SCSI driver for your SCSI adapter.
  # Also you need support for SCSI Generic (sg) in your operating system.
  # If using Linux, try "modprobe sg".

found USB scanner (vendor=0x03f0 [HP], product=0x5611 [Photosmart C3100 series])
 at libusb:001:005
  # Your USB scanner was (probably) detected. It may or may not be supported by
  # SANE. Try scanimage -L and read the backend's manpage.

  # Not checking for parallel port scanners.

  # Most Scanners connected to the parallel port or other proprietary ports
  # can't be detected by this program.

  # You may want to run this program as root to find all devices. Once you
  # found the scanner devices, be sure to adjust access permissions as
  # necessary.
```

Figure 1: By default, `sane-fine-scanner` gives you negative and positive results, as well as suggestions about how to troubleshoot.

```
bruce@nanday:~$ sane-find-scanner -q
found USB scanner (vendor=0x03f0 [HP], product=0x5611 [Photosmart C3100 series])
 at libusb:001:005
```

Figure 2: If you are only interested in positive results, add the `-q` option to run `sane-find-scanner` in quiet mode.

By default, `sane-find-scanner` runs in verbose mode, giving you results and some detailed comments about what to do if a scanner is undetected (Figure 1). However, if you are only interested in positive results, you can add the `-q` option to the command (Figure 2). If you want all the information that can be picked up from the installed scanners, type `sane-find-scanner -v -v | less`. You'll need to pipe to less, because this command returns several screens of information (Figure 3).

If a scanner is not detected and you think it should be, you have several options. For any scanner, the `-f` option will try to force detection, and, for a parallel scanner, you can use `-p`. Also, if you know the port to which a device is attached, you can use the port as a option. For example, `sane-find-scanner libusb:002` will probe the second USB port, whereas `sane-find-scanner libusb:001:005` will detect a scanner attached to the fifth slot of a USB hub.

These options, however, use the same language as the full report while being more limited in their search. So, if you specify libusb:002 as the port to probe and a scanner is attached to libusb:003, then the command will report no scanner.

The SANE Daemon and Device Drivers

One reason scanning is not traditionally enabled automatically for all users is because the SANE daemon, `saned`, is not secure. If you read the man page for saned, you will find repeated warnings only to use it behind a firewall and not to use it as root.

These warnings are definitely still worth taking into account, even though some distributions have opted for convenience over security and enabled scanning for all users.

The `/etc/sane.d` directory determines how saned operates. To gain a measure of security,

```
This is sane-find-scanner from sane-backends 1.0.20

  # sane-find-scanner will now attempt to detect your scanner. If the
  # result is different from what you expected, first make sure your
  # scanner is powered up and properly connected to your computer.

searching for SCSI scanners:
checking /dev/scanner... failed to open (Invalid argument)
checking /dev/sg0... failed to open (Invalid argument)
checking /dev/sg1... failed to open (Invalid argument)
checking /dev/sg2... failed to open (Invalid argument)
checking /dev/sg3... failed to open (Invalid argument)
checking /dev/sg4... failed to open (Invalid argument)
checking /dev/sg5... failed to open (Invalid argument)
checking /dev/sg6... failed to open (Invalid argument)
checking /dev/sg7... failed to open (Invalid argument)
checking /dev/sg8... failed to open (Invalid argument)
checking /dev/sg9... failed to open (Invalid argument)
checking /dev/sga... failed to open (Invalid argument)
checking /dev/sgb... failed to open (Invalid argument)
checking /dev/sgc... failed to open (Invalid argument)
checking /dev/sgd... failed to open (Invalid argument)
checking /dev/sge... failed to open (Invalid argument)
checking /dev/sgf... failed to open (Invalid argument)
checking /dev/sgg... failed to open (Invalid argument)
checking /dev/sgh... failed to open (Invalid argument)
checking /dev/sgi... failed to open (Invalid argument)
checking /dev/sgj... failed to open (Invalid argument)
checking /dev/sgk... failed to open (Invalid argument)
checking /dev/sgl... failed to open (Invalid argument)
:
```

Figure 3: With very verbose mode, `sane-find-scanner` gives you detailed information about the search for scanners, as well as any results.

```
# Configuration for the saned daemon

## Daemon options
# Port range for the data connection. Choose a range inside [1024 - 65535].
# Avoid specifying too large a range, for performance reasons.
#
# ONLY use this if your saned server is sitting behind a firewall. If your
# firewall is a Linux machine, we strongly recommend using the
# Netfilter nf_conntrack_sane connection tracking module instead.
#
# data_portrange = 10000 - 10100


## Access list
# A list of host names, IP addresses or IP subnets (CIDR notation) that
# are permitted to use local SANE devices. IPv6 addresses must be enclosed
# in brackets, and should always be specified in their compressed form.
#
# The hostname matching is not case-sensitive.

#scan-client.somedomain.firm
#192.168.0.1
#192.168.0.1/29
#[2001:7a8:185e::42:12]
#[2001:7a8:185e::42:12]/64

# NOTE: /etc/inetd.conf (or /etc/xinetd.conf) and
# /etc/services must also be properly configured to start
# the saned daemon as documented in saned(8), services(4)
# and inetd.conf(4) (or xinetd.conf(5)).
```

Figure 4: To make `saned` more secure, edit the ports and IP addresses with which it interacts in `/etc/sane.d/saned.conf`.

```
scsi HP
# Uncomment the following if you have "Error during device I/O" on SCSI
#   option dumb-read
#
# The usual place for a SCSI-scanner on Linux
/dev/scanner
#
# USB-scanners supported by the hp-backend
# HP ScanJet 4100C
usb 0x03f0 0x0101
# HP ScanJet 5200C
usb 0x03f0 0x0401
# HP ScanJet 62X0C
usb 0x03f0 0x0201
# HP ScanJet 63X0C
usb 0x03f0 0x0601
#
# Uncomment the following if your scanner is connected by USB,
# but you are not using libusb
# /dev/usb/scanner0
#   option connect-device
./hp.conf (END)
```

Figure 5: One way to troubleshoot a scanner is to edit the configuration file for its driver in `/etc/sane.d`.

you can use the file `saned.conf` to set the ports and IP addresses with which the daemon interacts (Figure 4).

The `/etc/sane.d` directory also contains configuration files for a variety of scanner manufacturers and models (Figure 5). Once you identify your scanner, you can edit or uncomment the various options in its configuration file if it is not working properly.

If editing any of these files does make a scanner operational, then SANE might not be working properly on your system. However, such issues are beyond the scope of this article, so check the man pages if you suspect the problem is larger than these solutions can handle.

Scanning from the Command Line

The main command for scanning from the command line is `scanimage`. This command includes the option `-L`, which reports all available scanners but lacks the detailed report of `sane-find-scanner`; it is mostly used as quick reference for other uses of the command.

The most common use of `scanimage` is to get the information needed to use the option `-d DEVICE` to specify a scanner or to ensure that the command finds it. You can also run `scanimage` with either the `-T` or `-test` option to check that the scanner is working properly (Figure 6).

However, `scanimage` mainly produces images in `.pnm` format from a scanner. Its usage can be as simple as `scanimage > [FILENAME].pnm`, or, if you prefer, `scanimage --filename FILE`. With the addition of the `-v` option, this basic command can become increasingly verbose (Figure 7). Alternatively, you might prefer to use the `-p` option to receive a running percentage of a scan being performed (see Figure 8).

This basic command can be supplemented by other options to control output. For example, GIMP and some other graphics editors can handle the `.pnm` format, but you might prefer to add `--format tiff` to produce files in a more popular format. With some scanners, you can also specify the *x* and *y* resolutions for the output, so that `-x 300 -y 300` produces an image with a resolution of 300x300dpi. The command

```
scanimage --help --device-name DEVICE
```

will show you options for controlling the brightness of the image, whether it prints in color or gray scale, and other options to improve the quality of a scan.

The `scanimage` command includes an especially useful set of options for scanning multiple pages. If your scanner has an automatic feed, the `--batch FORMAT` option lets you spec-

```
bruce@nanday:~$ scanimage -T
scanimage: scanning image of size 638x877 pixels at 24 bits/pixel
scanimage: acquiring RGB frame, 8 bits/sample
scanimage: reading one scanline, 1914 bytes...  PASS
scanimage: reading one byte...          PASS
scanimage: stepped read, 2 bytes...     PASS
scanimage: stepped read, 4 bytes...     PASS
scanimage: stepped read, 8 bytes...     PASS
scanimage: stepped read, 16 bytes...    PASS
scanimage: stepped read, 32 bytes...    PASS
scanimage: stepped read, 64 bytes...    PASS
scanimage: stepped read, 128 bytes...   PASS
```

Figure 6: To test whether your scanner works, use `scanimage`.

Anzeige
wird
separat
angeliefert

```
bruce@nanday:~$ scanimage -v > /home/bruce/test.pnm
scanimage: scanning image of size 638x877 pixels at 24 bits/pixel
scanimage: acquiring RGB frame
scanimage: min/max graylevel value = 0/255
scanimage: read 1678578 bytes in total
```

Figure 7: The `scanimage` command includes several levels of verbosity than can help you identify how your printer is working.

```
bruce@nanday:~$ scanimage -p > /home/bruce/test.pnm
Progress: 100.0%
```

Figure 8: With the `-p` option, `scanimage` gives ongoing reports about the percentage of a scan that is complete.

ify `.pnm` or `.tiff` format for the scans. With `--batch-start` *PAGE*, you can set the starting point of a scan, and `--batch-count` *NUMBER* controls the number of pages to scan. Similarly, `--batch-increment` *NUMBER* sets the gap between scans, and `batch-double` specifies that you are working from double-sided pages. If your scanner does not have an automatic feed, you can use `--batch-prompt` to delay each new scan until you press Enter.

For scanners with automatic feeders, you might prefer to use `scanadf` instead of `scanimage`. The commands are similar, in that they both use the `-L`, `-d`, and `-v` options.

Although the functionality of `scanimage` and `scanadf` is generally similar, the specific options are not. For example, `scanadf` uses `-o` or `--output` to specify the output file, not `--filename`. Also, it uses `-s` or `--start-count` instead of `--batch-start`. The `scanadf` command also uses `-e` or `--endcount` (Figure 9).

Another useful option with `scanadf` is `-S` or `--scan-script`, which specifies a Bash script that can be run after each scan. This option is convenient for converting from the default graphic format to another format, such as `.png` or `.jpg` via ImageMagick.

As you use `scanimage` and `scanadf`, you will notice a small delay before scanning begins, just as you would with a desktop scanning application. However, as shown in the screenshots for this article, both commands do not always exit cleanly when they are done. When that happens, you can press Ctrl + Z to return to the prompt.

Working Blind

If you are troubleshooting a scanner, the command line – or at least a text editor – is a logical choice. The equivalent tools mostly do not exist on the desktop.

By contrast, once you are actually using the scanner, you might feel strange to be manipulating graphics from an interface that does not allow you to observe them directly. However, this anomaly is more apparent than real.

Working with a scanner from the command line is more about file management than creativity. When you run `scanimage` or `scanadf`, the purpose is to add images to your system as quickly as possible – especially large groups of images. Although you will probably want to do a trial run, once you have the parameters set properly, running a scanner from the command line is far more efficient than any desktop scanner interface I have seen. Some might even argue that the process is more efficient partly because you are not distracted by the images, which frees you to focus on the process of scanning.

So, try running your scanner from the command line, and you might find that it's a practical alternative.

```
bruce@nanday:~$ scanadf -s 1 -e 2 >test.pnm
Scanned document image-0001
Scanned document image-0002
Scanned 2 pages
^Z
[10]+  Stopped                 scanadf -s 1 -e 2 > test.pnm
bruce@nanday:~$ ^C
bruce@nanday:~$ scanadf -v -s 1 -e 2 >test.pnm
scanadf: scanning image of size 638x877 pixels at 24 bits/pixel
scanadf: acquiring RGB frame
scanadf: min/max graylevel value = 0/255
Scanned document image-0001
scanadf: scanning image of size 638x877 pixels at 24 bits/pixel
scanadf: acquiring RGB frame
scanadf: min/max graylevel value = 0/255
Scanned document image-0002
Scanned 2 pages
```

Figure 9: If your scanner has an automatic document feed, you might prefer to use `scanadf` instead of `scanimage`.

INFO

[1] SANE: *http://www.sane-project.org*

[2] TWAIN: *http://www.twain.org/*

[3] XSane: *http://www.xsane.org/*

[4] Simple Scan: *https://launchpad.net/simple-scan*

[5] gscan2pdf: *http://gscan2pdf.sourceforge.net/*