NVIDIA/Tips and tricks

< NVIDIA

Contents [hide]

Fixing terminal resolution

Transitioning from nouveau may cause your startup terminal to display at a lower resolution.

For GRUB, see **GRUB/Tips and tricks#Setting the framebuffer resolution** for details.

For **rEFInd**, add to `esp/EFI/refind/refind.conf` and `/etc/refind.d/refind.conf` (latter file is optional but recommended):

```
use_graphics_for linux
```

A small caveat is that this will hide the kernel parameters from being shown during boot.

Using TV-out

See **Wikibooks:nVidia/TV-OUT**.

X with a TV (DFP) as the only display

The X server falls back to CRT-0 if no monitor is automatically detected. This can be a problem when using a DVI connected TV as the main display, and X is started while the TV is turned off or otherwise disconnected.

To force NVIDIA to use DFP, store a copy of the EDID somewhere in the filesystem so that X can parse the file instead of reading EDID from the TV/DFP.

To acquire the EDID, start nvidia-settings. It will show some information in tree format, ignore the rest of the settings for now and select the GPU (the corresponding entry should be titled "GPU-0" or similar), click the `DFP` section (again, `DFP-0` or similar), click on the `Acquire Edid` Button and store it somewhere, for example, `/etc/X11/dfp0.edid`.

If in the front-end mouse and keyboard are not attached, the EDID can be acquired using only the command line. Run an X server with enough verbosity to print out the EDID block:

```
$ startx -- -logverbose 6
```

After the X Server has finished initializing, close it and your log file will probably be in `/var/log/Xorg.0.log`. Extract the EDID block using nvidia-xconfig:

```
$ nvidia-xconfig --extract-edids-from-file=/var/log/Xorg.0.log --extract-edids-output-file=/etc/X11/dfp0.bin
```

Edit `xorg.conf` by adding to the `Device` section:

```
Option "ConnectedMonitor" "DFP"
Option "CustomEDID" "DFP-0:/etc/X11/dfp0.edid"
```

The `ConnectedMonitor` option forces the driver to recognize the DFP as if it were connected. The `CustomEDID` provides EDID data for the device, meaning that it will start up just as if the TV/DFP was connected during X the process.

This way, one can automatically start a display manager at boot time and still have a working and properly configured X screen by the time the TV gets powered on.

If the above changes did not work, in the `xorg.conf` under `Device` section you can try to remove the `Option "ConnectedMonitor" "DFP"` and add the following lines:

```
Option "ModeValidation" "NoDFPNativeResolutionCheck"
Option "ConnectedMonitor" "DFP-0"
```

The `NoDFPNativeResolutionCheck` prevents NVIDIA driver from disabling all the modes that do not fit in the native resolution.

Headless (no monitor) resolution

In headless mode, resolution falls back to 640x480, which is used by VNC or Steam Link. To start in a higher resolution *e.g.* 1920x1080, specify a `Virtual` entry under the `Screen` subsection in `xorg.conf`:

```
Section "Screen"
   [...]
   SubSection     "Display"
       Depth       24
       Virtual     1920 1080
   EndSubSection
EndSection
```

Tip: Using headless mode may be tricky and prone to error. For instance, in headless mode, desktop environments and **nvidia-utils** do not provide a graphical way to change resolution. To facilitate setting up resolution one can use a DP or an HDMI dummy adapter which simulates the presence of a monitor attached to that port. Then resolution change can be done normally using a remote session such as VNC or Steam Link.

Check the power source

The NVIDIA X.org driver can also be used to detect the GPU's current source of power. To see the current power source, check the 'GPUPowerSource' read-only parameter (0 - AC, 1 - battery):

```
$ nvidia-settings -q GPUPowerSource -t

1
```

Listening to ACPI events

NVIDIA drivers automatically try to connect to the **acpid** daemon and listen to ACPI events such as battery power, docking, some hotkeys, etc. If connection fails, X.org will output the following warning:

```
~/.local/share/xorg/Xorg.0.log

NVIDIA(0): ACPI: failed to connect to the ACPI event daemon; the daemon
NVIDIA(0):     may not be running or the "AcpidSocketPath" X
NVIDIA(0):     configuration option may not be set correctly.  When the
NVIDIA(0):     ACPI event daemon is available, the NVIDIA X driver will
NVIDIA(0):     try to use it to receive ACPI event notifications.  For
NVIDIA(0):     details, please see the "ConnectToAcpid" and
NVIDIA(0):     "AcpidSocketPath" X configuration options in Appendix B: X
NVIDIA(0):     Config Options in the README.
```

While completely harmless, you may get rid of this message by disabling the `ConnectToAcpid` option in your `/etc/X11/xorg.conf.d/20-nvidia.conf`:

```
Section "Device"
  ...
  Driver "nvidia"
  Option "ConnectToAcpid" "0"
  ...
EndSection
```

If you are on laptop, it might be a good idea to install and enable the **acpid** daemon instead.

Displaying GPU temperature in the shell

There are three methods to query the GPU temperature. *nvidia-settings* requires that you are using X, *nvidia-smi* or *nvclock* do not. Also note that *nvclock* currently does not work with newer NVIDIA cards such as GeForce 200 series cards as well as embedded GPUs such as the Zotac IONITX's 8800GS.

nvidia-settings

To display the GPU temp in the shell, use *nvidia-settings* as follows:

```
$ nvidia-settings -q gpucoretemp
```

This will output something similar to the following:

```
Attribute 'GPUCoreTemp' (hostname:0.0): 41.
'GPUCoreTemp' is an integer attribute.
'GPUCoreTemp' is a read-only attribute.
'GPUCoreTemp' can use the following target types: X Screen, GPU.
```

The GPU temps of this board is 41 C.

In order to get just the temperature for use in utilities such as *rrdtool* or *conky*:

```
$ nvidia-settings -q gpucoretemp -t

41
```

nvidia-smi

Use *nvidia-smi* which can read temps directly from the GPU without the need to use X at all, e.g. when running Wayland or on a headless server. To display the GPU temperature in the shell, use *nvidia-smi* as follows:

```
$ nvidia-smi
```

This should output something similar to the following:

```
$ nvidia-smi

Fri Jan  6 18:53:54 2012
+------------------------------------------------------+
| NVIDIA-SMI 2.290.10   Driver Version: 290.10         |
|-------------------------------+----------------------+----------------------+
| Nb.  Name                     | Bus Id        Disp.  | Volatile ECC SB / DB |
| Fan   Temp   Power Usage /Cap | Memory Usage         | GPU Util. Compute M. |
|===============================+======================+======================|
| 0.  GeForce 8500 GT           | 0000:01:00.0  N/A    |       N/A        N/A |
|  30%   62 C  N/A   N/A /  N/A |  17%   42MB /  255MB |  N/A      Default    |
|-------------------------------+----------------------+----------------------|
| Compute processes:                                               GPU Memory |
|  GPU  PID      Process name                                      Usage      |
|=============================================================================|
|  0.           ERROR: Not Supported                                          |
+-----------------------------------------------------------------------------+
```

Only for temperature:

```
$ nvidia-smi -q -d TEMPERATURE

====NVSMI LOG====

Timestamp                           : Sun Apr 12 08:49:10 2015
Driver Version                      : 346.59

Attached GPUs                       : 1
GPU 0000:01:00.0
    Temperature
        GPU Current Temp            : 52 C
        GPU Shutdown Temp           : N/A
        GPU Slowdown Temp           : N/A
```

In order to get just the temperature for use in utilities such as *rrdtool* or *conky*:

```
$ nvidia-smi --query-gpu=temperature.gpu --format=csv,noheader,nounits

52
```

Reference: **https://www.question-defense.com/2010/03/22/gpu-linux-shell-temp-get-nvidia-gpu-temperatures-via-linux-cli**.

nvclock

Use **nvclock**[AUR] which is available from the **AUR**.

Note: *nvclock* cannot access thermal sensors on newer NVIDIA cards such as Geforce 200 series cards.

There can be significant differences between the temperatures reported by *nvclock* and *nvidia-settings*/*nv-control*. According to **this post** by the author (thunderbird) of *nvclock*, the *nvclock* values should be more accurate.

Overclocking and cooling

Enabling overclocking

Warning: Overclocking might permanently damage your hardware. You have been warned.

Overclocking is controlled via *Coolbits* option in the `Device` section, which enables various unsupported features:

```
Option "Coolbits" "value"
```

Tip: The *Coolbits* option can be easily controlled with the *nvidia-xconfig*, which manipulates the Xorg configuration files:

```
# nvidia-xconfig --cool-bits=value
```

The *Coolbits* value is the sum of its component bits in the binary numeral system. The component bits are:

- `1` (bit 0) - Enables overclocking of older (pre-Fermi) cores on the *Clock Frequencies* page in *nvidia-settings*.
- `2` (bit 1) - When this bit is set, the driver will "attempt to initialize SLI when using GPUs with different amounts of video memory".
- `4` (bit 2) - Enables manual configuration of GPU fan speed on the *Thermal Monitor* page in *nvidia-settings*.
- `8` (bit 3) - Enables overclocking on the *PowerMizer* page in *nvidia-settings*. Available since version 337.12 for the Fermi architecture and newer.[1]
- `16` (bit 4) - Enables overvoltage using *nvidia-settings* CLI options. Available since version 346.16 for the Fermi architecture and newer.[2]

To enable multiple features, add the *Coolbits* values together. For example, to enable overclocking and overvoltage of Fermi cores, set `Option "Coolbits" "24"`.

The documentation of *Coolbits* can be found in `/usr/share/doc/nvidia/html/xconfigoptions.html` and **here**.

Note: An alternative is to edit and reflash the GPU BIOS either under DOS (preferred), or within a Win32 environment by way of **nvflash** and **NiBiTor 6.0**. The advantage of BIOS flashing is that not only can voltage limits be raised, but stability is generally improved over software overclocking methods such as Coolbits. **Fermi BIOS modification tutorial**

Setting static 2D/3D clocks

Set the following string in the `Device` section to enable PowerMizer at its maximum performance level (VSync will not work without this line):

```
Option "RegistryDwords" "PerfLevelSrc=0x2222"
```

Allow change to highest performance mode

The factual accuracy of this article or section is disputed.

Reason: This section refers to the limits for **GPU boost**, which is unrelated to overclocking discussed above. The `nvidia-smi(1)` man page says that it is "For Tesla devices from the Kepler+ family and Maxwell-based GeForce Titan." And as far as **Lahwaacz** is aware, the only GPU which supports this and does not have the default clocks equal to the maximum, is Tesla K40 [3]. Since the Pascal architecture, **Boost 3.0** handles automatic clocking even differently. (Discuss in **Talk:NVIDIA/Tips and tricks**)

Since changing performance mode and overclocking memory rate has little to no effect in *nvidia-settings*, try this:

- Setting Coolbits to 24 or 28 and remove Powermizer RegistryDwords -> Restart X
- find out max. Clock and Memory rate. (this can be LOWER than what your gfx card reports after booting!):

```
$ nvidia-smi -q -d SUPPORTED_CLOCKS
```

- set rates for GPU 0:

```
# nvidia-smi -i 0 -ac memratemax,clockratemax
```

After setting the rates the max. performance mode works in *nvidia-settings* and you can overclock graphics-clock and memory transfer rate.

Saving overclocking settings

Typically, clock and voltage offsets inserted in the *nvidia-settings* interface are not saved, being lost after a reboot. Fortunately, there are tools that offer an interface for overclocking under the proprietary driver, able to save the user's overclocking preferences and automatically applying them on boot. Some of them are:

- **gwe**[AUR] - graphical, applies settings on desktop session start
- **nvclock**[AUR] and **systemd-nvclock-unit**[AUR] - graphical, applies settings on system boot
- **nvoc**[AUR] - text based, profiles are configuration files in `/etc/nvoc.d/`, applies settings on desktop session start

Custom TDP Limit

Modern Nvidia graphics cards throttle frequency to stay in their TDP and temperature limits. To increase performance it is possible to change the TDP limit, which will result in higher temperatures and higher power consumption.

For example, to set the power limit to 160.30W:

```
# nvidia-smi -pl 160.30
```

To set the power limit on boot (without driver persistence):

```
/etc/systemd/system/nvidia-tdp.timer

[Unit]
Description=Set NVIDIA power limit on boot

[Timer]
OnBootSec=5

[Install]
WantedBy=timers.target
```

```
/etc/systemd/system/nvidia-tdp.service

[Unit]
Description=Set NVIDIA power limit

[Service]
Type=oneshot
ExecStart=/usr/bin/nvidia-smi -pl 160.30
```

Set fan speed at login

This article or section needs language, wiki syntax or style improvements. See Help:Style for reference.

Reason: Refer to #Enabling overclocking for description of *Coolbits*. (Discuss in Talk:NVIDIA/Tips and tricks)

You can adjust the fan speed on your graphics card with *nvidia-settings'* console interface. First ensure that your Xorg configuration has enabled the bit 2 in the **Coolbits** option.

Note: GeForce 400/500 series cards cannot currently set fan speeds at login using this method. This method only allows for the setting of fan speeds within the current X session by way of *nvidia-settings*.

Place the following line in your **xinitrc** file to adjust the fan when you launch Xorg. Replace `n` with the fan speed percentage you want to set.

```
nvidia-settings -a "[gpu:0]/GPUFanControlState=1" -a "[fan:0]/GPUTargetFanSpeed=n"
```

You can also configure a second GPU by incrementing the GPU and fan number.

```
nvidia-settings -a "[gpu:0]/GPUFanControlState=1" -a "[fan:0]/GPUTargetFanSpeed=n" \
                -a "[gpu:1]/GPUFanControlState=1" -a  [fan:1]/GPUTargetFanSpeed=n" &
```

If you use a login manager such as **GDM** or **SDDM**, you can create a desktop entry file to process this setting. Create `~/.config/autostart/nvidia-fan-speed.desktop` and place this text inside it. Again, change `n` to the speed percentage you want.

```
[Desktop Entry]
Type=Application
Exec=nvidia-settings -a "[gpu:0]/GPUFanControlState=1" -a "[fan:0]/GPUTargetFanSpeed=n"
X-GNOME-Autostart-enabled=true
Name=nvidia-fan-speed
```

Note: Before driver version 349.16, `GPUCurrentFanSpeed` was used instead of `GPUTargetFanSpeed` .[4]

To make it possible to adjust the fanspeed of more than one graphics card, run:

```
$ nvidia-xconfig --enable-all-gpus
$ nvidia-xconfig --cool-bits=4
```

Note: On some laptops (including the ThinkPad X1 Extreme and P51/P52), there are two fans, but neither are controlled by nvidia.

Kernel module parameters

This article or section needs language, wiki syntax or style improvements. See Help:Style for reference.

Reason: Giving advanced examples without explaining what they do is pointless. (Discuss in Talk:NVIDIA/Tips and tricks)

Some options can be set as kernel module parameters, a full list can be obtained by running `modinfo nvidia` or looking at `nv-reg.h`. See **Gentoo:NVidia/nvidia-drivers#Kernel module parameters** as well.

For example, enabling the following will turn on kernel mode setting (see above) and enable the PAT feature [5], which affects how memory is allocated. PAT was first introduced in Pentium III [6] and is supported by most newer CPUs (see **wikipedia:Page attribute table#Processors**). If your system can support this feature, it should improve performance.

```
/etc/modprobe.d/nvidia.conf

options nvidia-drm modeset=1
options nvidia NVreg_UsePageAttributeTable=1
```

On some notebooks, to enable any nvidia settings tweaking you must include this option, otherwise it responds with "Setting applications clocks is not supported" etc.

```
/etc/modprobe.d/nvidia.conf

options nvidia NVreg_RegistryDwords="OverrideMaxPerf=0x1"
```

Preserve video memory after suspend

By default the NVIDIA Linux drivers save and restore only essential video memory allocations on system suspend and resume. Quoting NVIDIA ([7], also available with the `nvidia-utils` package in **/usr/share/doc/nvidia/html/powermanagement.html**): *The resulting loss of video memory contents is partially compensated for by the user-space NVIDIA drivers, and by some applications, but can lead to failures such as rendering corruption and application crashes upon exit from power management cycles.*

The **still experimental** system enables saving all video memory (given enough space on disk or main RAM). The interface is through the **/proc/driver/nvidia/suspend** file as follows: write "suspend" (or "hibernate") to **/proc/driver/nvidia/suspend** immediately before writing to the usual Linux **/sys/power/state** file, write "resume" to **/proc/driver/nvidia/suspend** immediately after waking up, or after an unsuccessful attempt to suspend or hibernate.

The NVIDIA drivers rely on a user defined file system for storage. The chosen file system needs to support unnamed temporary files (ext4 works) and have sufficient capacity for storing the video memory allocations (e.g., at least `(sum of the memory capacities of all NVIDIA GPUs) * 1.2`). Use the command `nvidia-smi -q -d MEMORY` to list the memory capacities of all GPUs in the system.

To choose the file system used for storing video memory during system sleep (and change the default video memory save/restore strategy to save and restore all video memory allocations), it is necessary to pass two options to the "nvidia" kernel module. For example, write the following line to **/etc/modprobe.d/nvidia-power-management.conf** and reboot:

```
options nvidia NVreg_PreserveVideoMemoryAllocations=1 NVreg_TemporaryFilePath=/tmp-nvidia
```

Feel free to replace "/tmp-nvidia" in the previous line with a path within your desired file system.

The interaction with **/proc/driver/nvidia/suspend** is handled by the simple Unix shell script at **/usr/bin/nvidia-sleep.sh**, which will itself be called by a tool like **Systemd**. The Archlinux `nvidia-utils` package ships with the following relevant Systemd services (which essentially just call **nvidia-sleep.sh**): `nvidia-suspend`, `nvidia-hibernate`, `nvidia-resume`. Contrary to NVIDIA's instructions, it is currently not necessary to enable `nvidia-resume` (and it is in fact probably not a good idea to enable it), because the **/usr/lib/systemd/system-sleep/nvidia** script does the same thing as the service (but slightly earlier), and it is enabled by default (Systemd calls it after waking up from a suspend). Do enable `nvidia-suspend` and/or `nvidia-hibernate`.

Driver persistence

Nvidia has a daemon that can be optionally run at boot. In a standard single-GPU X desktop environment the persistence daemon is not needed and can actually create issues [8]. See the **Driver Persistence** section of the Nvidia documentation for more details.

To start the persistence daemon at boot, **enable** the `nvidia-persistenced.service`. For manual usage see the **upstream documentation**.

Categories: Graphics | X server