How to get the GPU info?

Asked 13 years, 7 months ago Modified 1 year ago Viewed 2.0m times

391

I'm looking for a command that would give me the same info as:

```
cat /proc/cpuinfo
```

Except for the GPU (type of the chip and memory, frequency).

command-line **graphics** **gpu**

Share Edit Follow Flag

edited Dec 15, 2017 at 1:36
muru
199k 55 493 747

asked Oct 9, 2010 at 17:14
user2413
14.4k 17 54 89

what does: `cat /proc/cpuinfo` do? what info are you looking for? – Charlie Parker Mar 5, 2018 at 17:00

5 @CharlieParker it outputs information of the cpu... – Emobe Sep 11, 2019 at 8:46

2 I personally use: `python -c "import torch; print(torch.cuda.get_device_name(0));"` – Charlie Parker Nov 7, 2022 at 22:44

25 Answers

Sorted by: Highest score (default)

408

I do not know of a direct equivalent, but **lshw** should give you the info you want, try:

```
sudo lshw -C display
```

(it also works without `sudo` but the info may be less complete/accurate)

You can also install the package `lshw-gtk` to get a GUI.

Share Edit Follow Flag

answered Oct 9, 2010 at 17:31
Marcel Stimberg
42k 8 47 45

3 Had to put gksu before the command in the menu to get lshw-gtk to work. – robin0800 Feb 15, 2011 at 10:55

Any updates? I'm a fan of the command but the only clock rate (frequency) it seems to provide for me is the base bus clock 33MHz. I'm attempting to bring this Q&A up to date. Thank you! – Elder Geek Dec 14, 2017 at 23:31

3 Apologies, new to Deep Learning. What should it say if I have a GPU? It says `product: 2nd Generation Core Processor Family Integrated Graphics Controller` – Nathan majicvr.com Apr 17, 2018 at 2:13

2 @Nathan That means that you have a GPU....probably a very weak GPU. Your GPU says `integrated graphics` , which means that it's integrated into the CPU. Your CPU has it own component which functions as a graphics card and probably (to save on costs) uses the ordinary RAM to store its buffers. You do not have a separate independent removable graphics card. – Jack G May 13, 2020 at 0:03

This doesn't (always?) give the memory for discrete NVidia cards on certain laptops (like the HP G7 Firefly) – AntonOfTheWoods Apr 4, 2023 at 1:32

175

That type of information is non-standard, and the tools you will use to gather it vary widely.

The command `glxinfo` will give you all available OpenGL information for the graphics processor, including its vendor name, if the drivers are correctly installed.

To get clock speed information, there is no standard tool.

- For ATI/AMD GPUs running the old Catalyst driver, `aticonfig --odgc` should fetch the clock rates, and `aticonfig --odgt` should fetch the temperature data. I'm not familiar with AMDGPU-Pro, but a similar tool should exist.
- For NVIDIA GPUs, the `nvidia-smi` tool will show all of the information you could want, including clock speeds and usage statistics.

I am not aware of an equivalent tool for the open source drivers or for Intel or other GPUs, but other information on the hardware can be fetched from the `lspci` and `lshw` tools.

Share Edit Follow Flag

edited Jun 2, 2020 at 18:17

answered Oct 9, 2010 at 17:30

greyfade
2,505 2 17 7

3 How to install glxinfo? – stiv Mar 13, 2015 at 13:40

12 @stiv: It's part of the Mesa library, and comes with the package `mesa-utils` on Ubuntu. – greyfade Mar 13, 2015 at 18:20

4 `aticonfig` doesn't appear to be available since the retirement of fglrx. `nvclock` also appears to have been abandoned since the last version was for trusty. Do you have any updated solutions? Here's what I have so far.. – Elder Geek Dec 14, 2017 at 23:16

8 `glxinfo | grep "Device"` worked well enough for me on an Intel GPU – John Hamilton May 26, 2018 at 14:10

7 I use: `glxinfo | egrep -i 'device|memory'` – Melroy van den Berg Jan 2, 2019 at 20:37

|

150

A blog post focusing on work done on the command-line is here:

http://www.cyberciti.biz/faq/howto-find-linux-vga-video-card-ram/

Find out the device ID:

```
 lspci | grep ' VGA ' | cut -d" " -f 1
03:00.0
```

You can then use this output with `lspci` again, forming **two nested commands**

```
lspci  -v -s  $(lspci | grep ' VGA ' | cut -d" " -f 1)
```

If you have more than 1 GPU card, try this equivalent command instead:

```
lspci | grep ' VGA ' | cut -d" " -f 1 | xargs -i lspci -v -s {}
```

Output from my system:

```
03:00.0 VGA compatible controller: NVIDIA Corporation G98 [Quadro NVS 295] (rev a1) (prog-if 00 [VGA controller])
    Subsystem: NVIDIA Corporation Device 062e
    Flags: bus master, fast devsel, latency 0, IRQ 24
    Memory at f6000000 (32-bit, non-prefetchable) [size=16M]
    Memory at ec000000 (64-bit, prefetchable) [size=64M]
    Memory at f4000000 (64-bit, non-prefetchable) [size=32M]
    I/O ports at dc80 [size=128]
    [virtual] Expansion ROM at f7e00000 [disabled] [size=128K]
    Capabilities: <access denied>
    Kernel driver in use: nvidia
```

EDIT: You can avoid the `<access denied>` by launching with `sudo`

So, `(prefetchable) [size=64M)` indicates that I have a 64-MB NVIDIA card. However, I don't, it's rather 256 MB. Why? See below.

To see how to get the most info and performance out of it, read an extremely comprehensive article on the Arch-Linux Wiki

https://wiki.archlinux.org/index.php/NVIDIA

For **nvidia users**, start with

```
nvidia-smi
```

(This works with the Nvidia drivers installed,but not with systems running the open-source 'nouveau' driver).

Output

```
Thu Dec 19 10:54:18 2013
+------------------------------------------------------+
| NVIDIA-SMI 5.319.60   Driver Version: 319.60         |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Quadro NVS 295      Off  | 0000:03:00.0     N/A |                  N/A |
| N/A   73C  N/A     N/A /  N/A |      252MB /   255MB |     N/A      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Compute processes:                                                  GPU Memory |
|  GPU       PID  Process name                                        Usage      |
|=============================================================================|
|    0            Not Supported                                                  |
+-----------------------------------------------------------------------------+
```

This indicates that I have a 256 MB GDDR3 Graphics card.

At this time, I don't know how to get this for Intel and AMD/ATI GPUs.

Share Edit Follow Flag

edited Jul 30, 2019 at 23:28
Nathan majicvr.com
153 2 12

answered Dec 19, 2013 at 9:50
knb
5,014 4 34 42

30 +1 for `nvidia-smi` (that should be highlighted a bit in my opinion) – Martin Thoma Sep 7, 2014 at 15:23

1 If anyone have any idea: nvidia-smi hangs indefinitely: what could be the issue? – Franck Dernoncourt Aug 9, 2016 at 15:50

5 `nvidia-smi` not showing me full GPU name. – mrgloom Nov 24, 2018 at 15:15

12 `nvidia-smi -q` , as suggested by @Quanlong uses more sensible output format. – Nickolay Oct 27, 2019 at 9:47

1 You can also use `$ nvidia-smi --query-gpu=gpu_name --format=csv` to just get the GPU name – Eric Wiener May 14, 2021 at 11:40

92

Run `google-chrome` and navigate to the URL `about:gpu` . If chrome has figured out how to use OpenGL, you will get extremely detailing information about your GPU.

Share Edit Follow Flag

edited Aug 25, 2015 at 19:19
David Foerster
36.4k 56 94 148

answered Nov 15, 2012 at 7:35
Chris Uhlik
921 6 2

8 This also works in Chromium (`chromium-browser`). – Eliah Kagan Jul 2, 2017 at 13:02

1 Clever. Along these lines I additionally went to chromeexperiments.com to see the performance there. Smooth as butter - I'm definitely on gpu – Jacksonkr Jul 29, 2018 at 17:49

66

Because you specified a command like `cat` for CPU's this is therefore the equivalent for GPU's. Specifically for Nvidia cards. It requires no software except the Nvidia device driver to be loaded.

The path here works for the cards I have. But yours may differ as others have pointed out in the comments.

1st GPU

```
> cat /proc/driver/nvidia/gpus/0/information
Model:       GeForce GTX 680
IRQ:         24
GPU UUID:    GPU-71541068-cded-8a1b-1d7e-a093a09e9842
Video BIOS:      80.04.09.00.01
Bus Type:    PCIe
```

```
DMA Size:    40 bits
DMA Mask:    0xffffffffff
Bus Location:    0000:01.00.0
```

2nd GPU

```
> cat /proc/driver/nvidia/gpus/1/information
Model:       GeForce GTX 580
IRQ:         33
GPU UUID:    GPU-64b1235c-51fc-d6f1-0f0e-fa70320f7a47
Video BIOS:      70.10.20.00.01
Bus Type:    PCIe
DMA Size:    40 bits
DMA Mask:    0xffffffffff
Bus Location:    0000:08.00.0
```

Share Edit Follow Flag

edited Sep 26, 2019 at 3:33

answered Apr 1, 2015 at 21:22

hookenz
2,599 6 32 40

19 Thanks! (though `cat /proc/driver/nvidia/gpus/0000\:01\:00.0/information` for me) – matt wilkie Nov 24, 2015 at 3:54

5 This is the only correct answer in on-demand cloud/HPC cluster environment on which glxinfo or lspci both fail (the former because there's no OpenGL and display, the latter because the nVidia graphics card is abstracted by a graphics controller like Matrox G200eW3). The folder name under gpus is `0000:3b:00.0` or `0000:d8:00.0` for me, so we should type: `cat /proc/driver/nvidia/gpus/0000:3b:00.0/information` . The lovely `Tesla V100-PCIE-16GB` model shows that the `qsub` job limit is satisfied as desired. – user5280911 Oct 6, 2018 at 7:40

34

For Nvidia cards, type

```
nvidia-smi -q
```

Share Edit Follow Flag

answered Jun 26, 2017 at 5:48

Quanlong
491 5 5

This got me the UUID of the Nvidia GPU on Windows. Thanks! Really useful! – rd51 Jul 5, 2023 at 10:28

28

clinfo

```
sudo apt-get install clinfo
clinfo
```

is the analogue of `glxinfo` but for OpenCL, my GPU setup is described at: https://stackoverflow.com/questions/7542808/how-to-compile-opencl-on-ubuntu/33483311#33483311 The output contains my GPU model among other things:

```
Number of devices                                 1
  Device Name                                     Quadro M1200
```

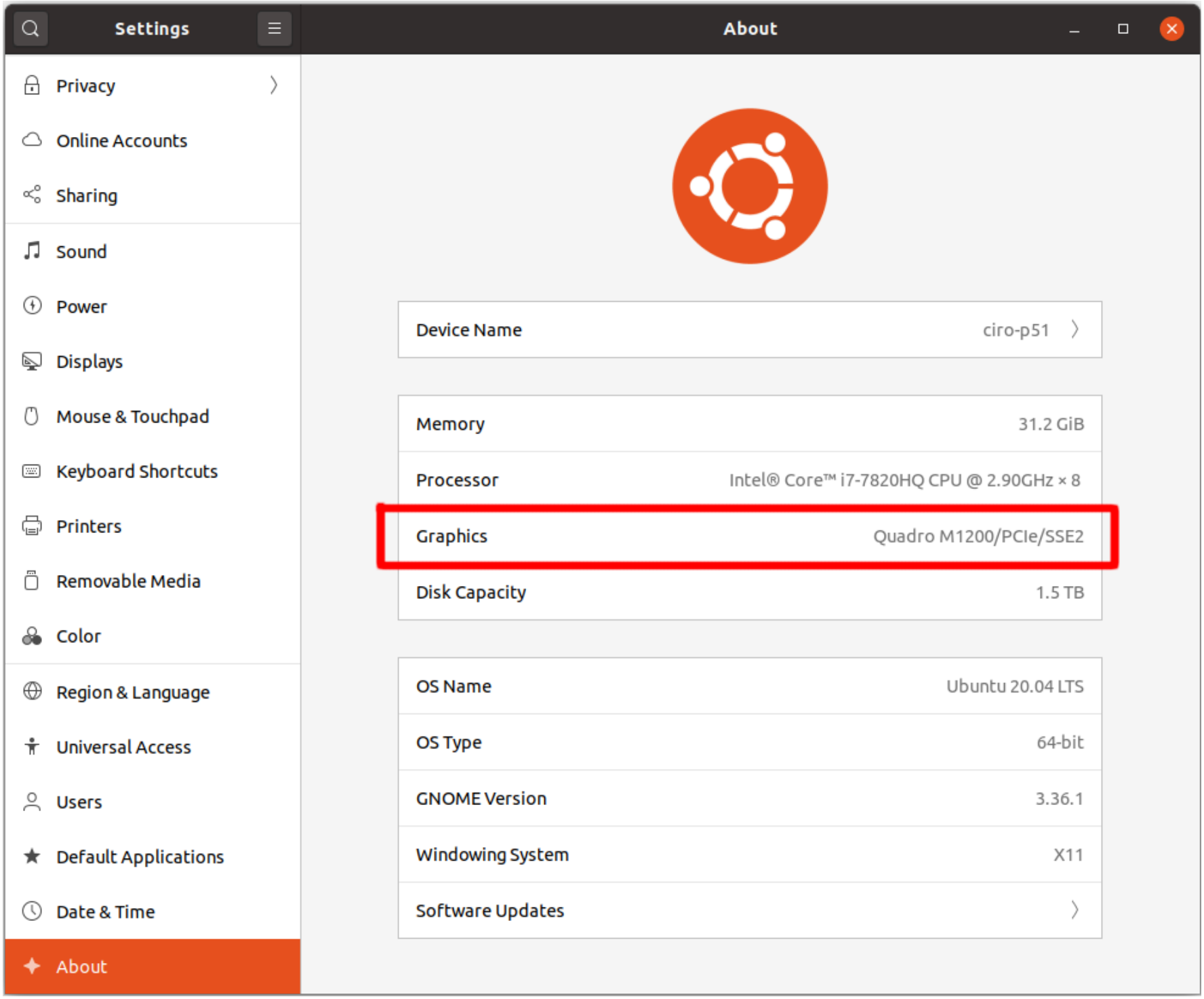
Ubuntu 20.04 Settings -> About

You can either open settings by clicking on top right menu, or you can just do:

- Super key (AKA Windows key)
- Type "about" and select the entry

So under "Graphics" I can see that my GPU model is

> Quadro M1200/PCIe/SSE2

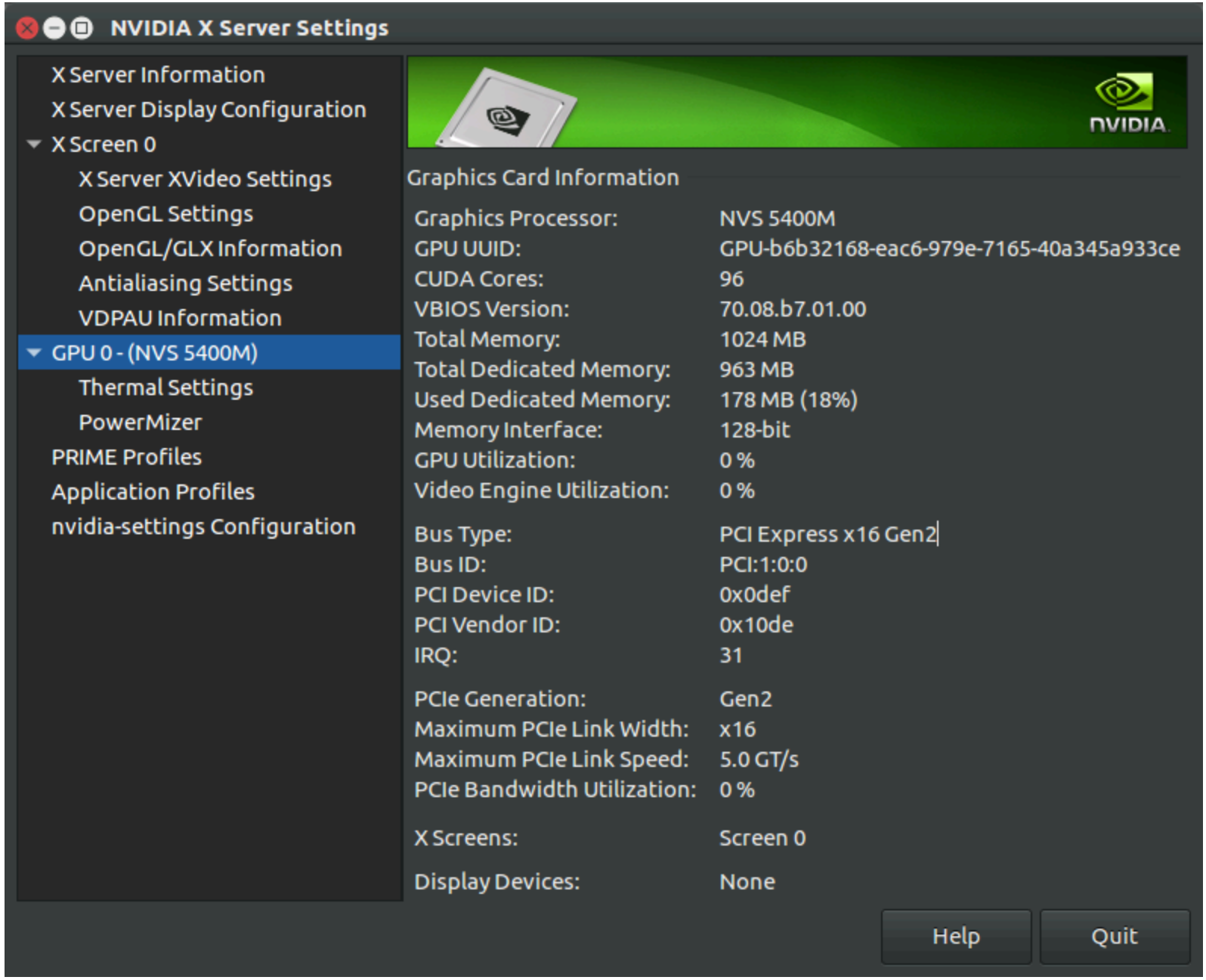

Settings
About
Privacy
Online Accounts
Sharing
Sound
Power
Displays
Mouse & Touchpad
Keyboard Shortcuts
Printers
Removable Media
Color
Region & Language
Universal Access
Users
Default Applications
Date & Time
About
Device Name
ciro-p51
Memory
31.2 GiB
Processor
Intel® Core™ i7-7820HQ CPU @ 2.90GHz × 8
Graphics
Quadro M1200/PCIe/SSE2
Disk Capacity
1.5 TB
OS Name
Ubuntu 20.04 LTS
OS Type
64-bit
GNOME Version
3.36.1
Windowing System
X11
Software Updates

Some other things it can show:

- "Software Rendering" (Ubuntu 23.10): graphics card not working at all
- "NV117": I think this means it is using Nouveau

nvidia-settings

Mixes runtime with some static info.



More details: How do I check if Ubuntu is using my NVIDIA graphics card?

Share Edit Follow Flag edited May 25, 2023 at 9:04 answered Nov 2, 2015 at 10:14

Ciro Santilli
OurBigBook.com
28.9k 14 119 113

1 This is perfect -- this answer deserved more upvotes. I think the other answers were more catered towards lower-level technical information about the GPU, whereas this option is much more consumer-friendly (since it resembles the Nvidia Control Panel program everyone on Windows is familiar with). – Raleigh L. Aug 2, 2022 at 23:11

14

I do believe the best option for this is neofetch.

```
# Get neofetch
sudo add-apt-repository ppa:dawidd0811/neofetch
sudo apt update
sudo apt install neofetch
# Run neofetch
neofetch
```

This gives an output like this:

```
haozeke ~ neofetch
                    -`                    haozeke@myi7archbox
                   .o+`                   ------------------
                  `ooo/                   OS: Arch Linux x86_64
                 `+oooo:                  Host: HP 406 G1 MT
                `+oooooo:                 Kernel: 4.14.3-1-zen
                -+oooooo+:                Uptime: 4 days, 11 hours, 21 mins
              `/:-:++oooo+:               Packages: 1925
             `/++++/+++++++:              Shell: zsh 5.4.2
            `/++++++++++++++:             Resolution: 1366x768, 1920x1200
           `/+++ooooooooooooo/`           DE: KDE
          ./ooosssso++osssssso+`          WM: LG3D
         .oossssso-`` `/ossssss+`         Theme: Breeze Dark [KDE], Arc-Dark-Grey [GTK2/3]
        -osssssso.      :ssssssso.        Icons: Breeze-dark [KDE], Papirus-Dark [GTK2/3]
       :osssssss/        osssso+++.       Terminal: tilix
      /ossssssss/        +ssssooo/-       CPU: Intel i7-4770 (8) @ 3.900GHz
    `/ossssso+/:-        -:/+osssso+-     GPU: AMD Mobility Radeon HD 4530/4570/545v
   `+sso+:-`                 `.-/+oso:    Memory: 14117MiB / 28065MiB
  `++:.                          `-/+/
  .`                                 `/
```

Share Edit Follow Flag

edited Dec 14, 2019 at 12:50
Aminu Kano
133 6

answered Dec 10, 2017 at 16:37
HaoZeke
429 3 8

5 I'm not seeing the video card frequency and memory in this answer. There are far simpler methods to obtain the model of GPU which appears to be all you are giving us. I'm not sure what this adds to the existing answers. – Elder Geek Dec 10, 2017 at 17:16

4 The `screenfetch` program does the same thing and doesn't require a PPA to install. – Braden Best Jan 30, 2018 at 19:31

1 It's a shell script. Plus I linked to its github as well so you can just use it as a script. – HaoZeke Jan 30, 2018 at 19:36

3 Here's screenfetch: github.com/KittyKatt/screenFetch – Camille Goudeseune Feb 7, 2018 at 21:31

10

If you're looking for only the names of the video cards on the machine, then simply use:

```
$ nvidia-smi --list-gpus
```

For some newer GPUs, this also lists the memory of each device.

Share Edit Follow Flag

answered Apr 15, 2019 at 15:59
kmario23
1,019 2 10 20

1 to be even shorter: `$nvidia-smi -L` – Shawyan Azdam May 18, 2020 at 21:14

10

This is really not that complex For model and memory, here's a 1 liner that works for every video card I've tested it on regardless of manufacturer (Intel, AMD, NVIDIA):

```
GPU=$(lspci | grep VGA | cut -d ":" -f3);RAM=$(cardid=$(lspci | grep VGA |cut -d " " -f1);lspci -v -s $cardid | grep " prefetchable"| cut -d "=" -f2);echo $GPU $RAM
```

GPU= All this bit does is grab the 3rd field from 'lspci' output filtered via 'grep' for VGA which corresponds to the video chip.

RAM= All this bit does is set variable `cardid` equal to the first field of output from `lspci` matching "VGA" and feeds that as a request for `-v` verbose output from `lspci` for that specific `-s` device, further filtering the output by `grep` for the string " prefetchable" as this contains the memory on the card itself (note the preceding space as we don't want to match "non-prefetchable" in our output.

For clock rate on Intel integrated graphics (Tested on I3 and I5)

execute the command `sudo find /sys -type f -name gt_cur* -print0 | xargs -0 cat` This dives into the /sys tree to locate the gt_cur_freq_mhz file which on my I3 is `/sys/devices/pci0000:00/0000:00:02.0/drm/card0/gt_cur_freq_mhz` and prints the content. which in my case under extremely light load is `350` as in 350 MHz which corresponds to the minimum frequency found in `/sys/devices/pci0000:00/0000:00:02.0/drm/card0/gt_min_freq_mhz` and when running `glxgears` and `glmark2` results in `1050` as in 1050 MHz which corresponds to the maximum frequency found in `/sys/devices/pci0000:00/0000:00:02.0/drm/card0/gt_max_freq_mhz`

For clock rates on nvidia cards:

`nvidia-smi -stats -d procClk` corresponds to the GPU clock `nvidia-smi -stats -d memClk` corresponds to the memory clock.

Note: I am unable to test the above as my trusty GeForce 210 isn't supported and this works only on Kepler or newer GPUs as indicated by `nvidia-smi -stats --help'

I do not currently have any solutions for clock rate on AMD cards and do not have the hardware available for testing. I will however say that to the best of my knowledge the `aticonfig` mentioned in the accepted answer no longer exists and it appears that `nvclock` isn't available for anything since trusty.

Share Edit Follow Flag

edited Dec 14, 2017 at 23:17

answered Dec 11, 2017 at 13:52

Elder Geek
36.1k 26 98 183

8

Conky or Terminal Splash Screen

I use two methods to automatically display nVidia GPU and Intel iGPU information:

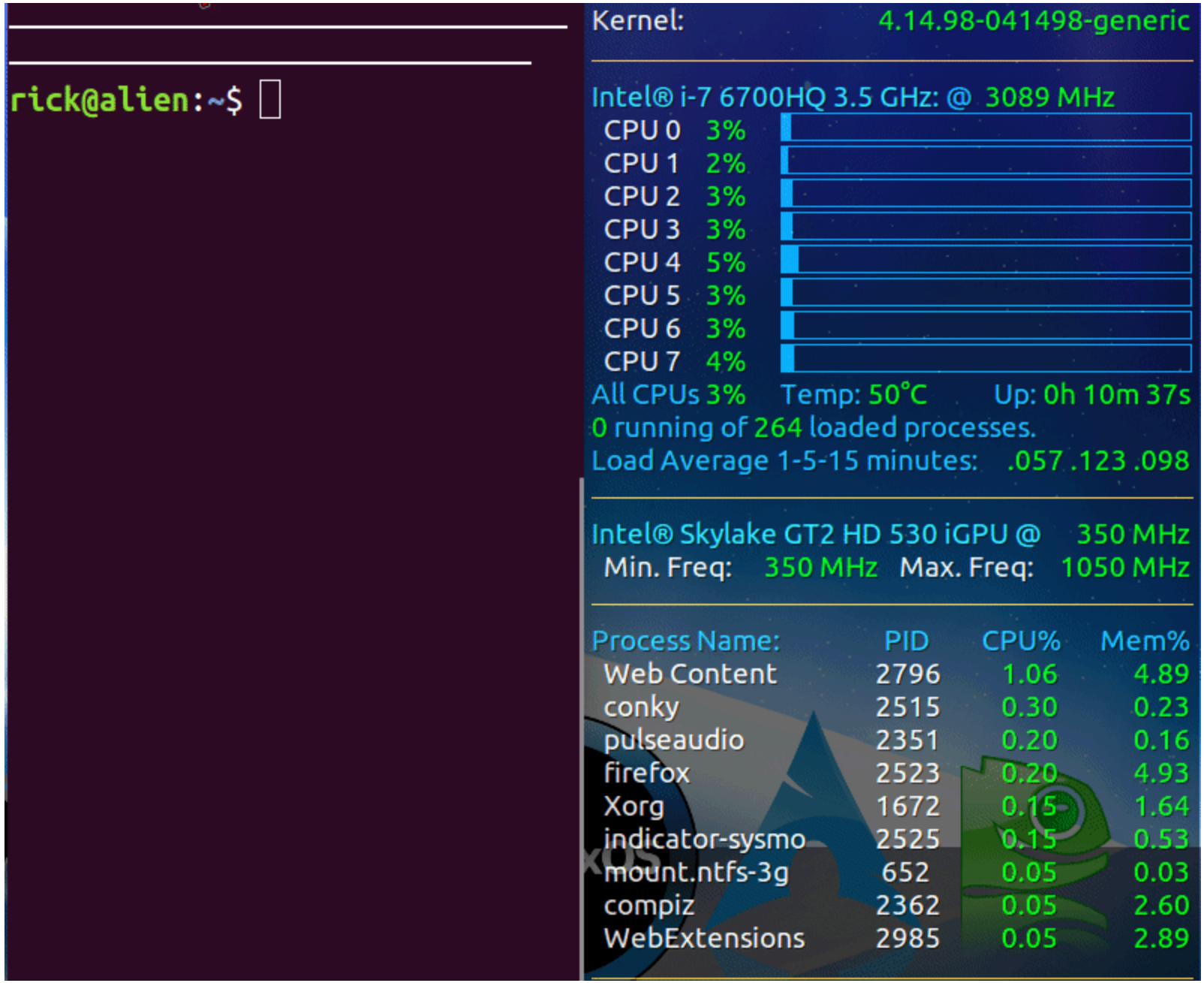
- Conky dynamically displays GPU information in real time
- `~/.bashrc` displays GPU information each time the terminal is opened

Conky real time display

This example uses Conky to display current GPU (nVidia or Intel) stats in real time. Conky is a light weight system monitor popular among many Linux enthusiasts.

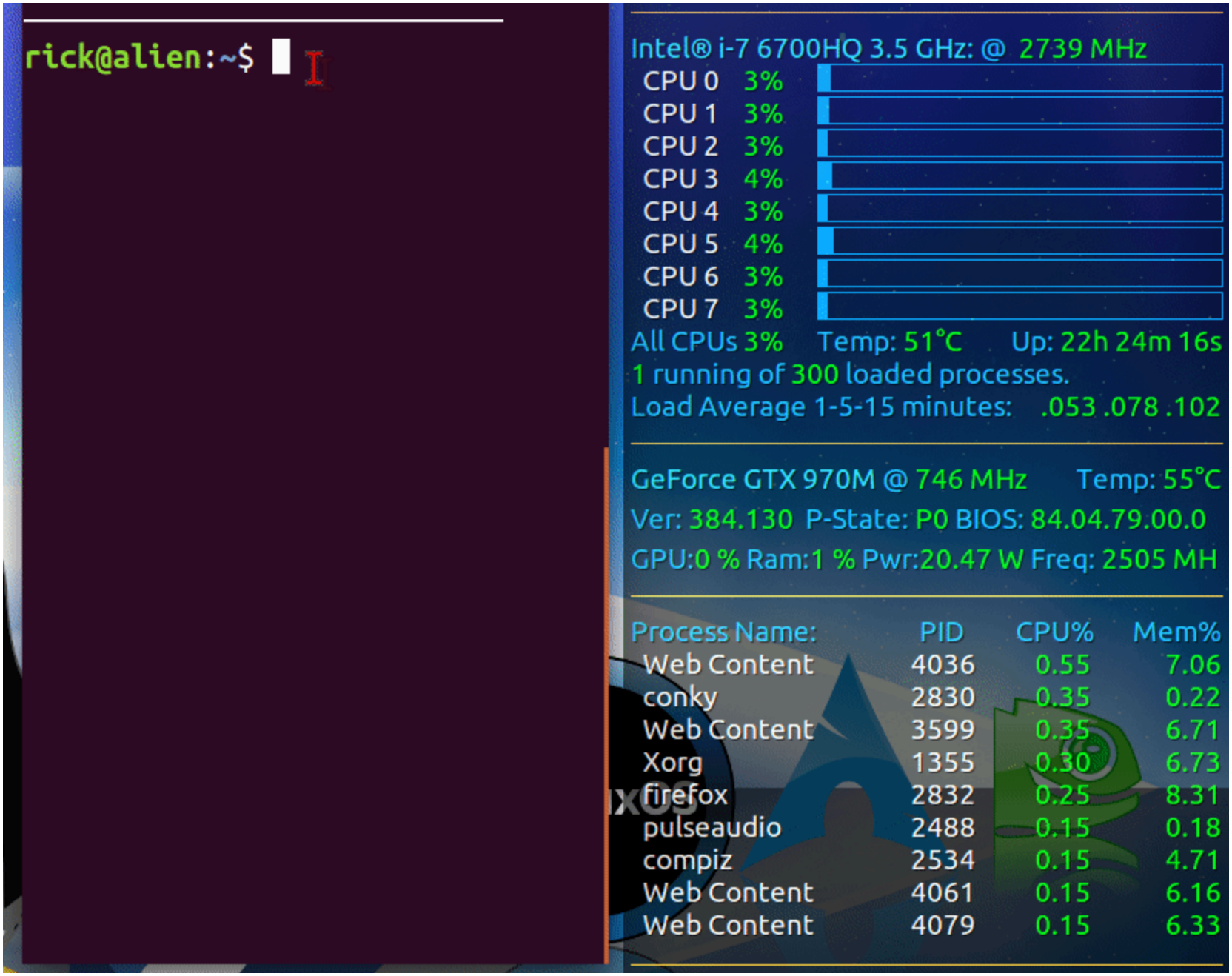
The display changes depending on if you booted after `prime-select intel` or `prime-select nvidia`.

Intel iGPU



- The Intel iGPU shows as `Skylake GT2 HD 530 iGPU` with current frequency
- The Minimum frequency is `350` MHz and the Maximum is `1050` MHz

nVidia GPU



- The nVidia GPU shows as `GeForce GTX970M` with current GPU frequency and temperature

- The Driver version, P-State and BIOS version are displayed
- The GPU load, RAM use, Power Consumption and RAM frequency is displayed

Conky Code

Here is the relevant Conky script for Intel iGPU and nVidia GPU:

```
#------------+
# Intel iGPU |
#------------+
${color orange}${hr 1}${if_match "intel" == "${execpi 99999 prime-select query}"}
${color2}${voffset 5}Intel® Skylake GT2 HD 530 iGPU @${alignr}${color green}${execpi .001 (cat
/sys/class/drm/card1/gt_cur_freq_mhz)} MHz
${color}${goto 13}Min. Freq:${goto 120}${color green}${execpi .001 (cat /sys/class/drm/card1/gt_min_freq_mhz)}
MHz${color}${goto 210}Max. Freq:${alignr}${color green}${execpi .001 (cat /sys/class/drm/card1/gt_max_freq_mhz)} MHz
${color orange}${hr 1}${else}
#------------+
# Nvidia GPU |
#------------+
${color2}${voffset 5}${execpi .001 (nvidia-smi --query-gpu=gpu_name --format=csv,noheader)} ${color1}@ ${color
green}${execpi .001 (nvidia-smi --query-gpu=clocks.sm --format=csv,noheader)} ${alignr}${color1}Temp: ${color
green}${execpi .001 (nvidia-smi --query-gpu=temperature.gpu --format=csv,noheader)}°C
${color1}${voffset 5}Ver: ${color green}${execpi .001 (nvidia-smi --query-gpu=driver_version --format=csv,noheader)}
${color1} P-State: ${color green}${execpi .001 (nvidia-smi --query-gpu=pstate --format=csv,noheader)}
${alignr}${color1}BIOS: ${color green}${execpi .001 (nvidia-smi --query-gpu=vbios_version --format=csv,noheader)}
${color1}${voffset 5}GPU:${color green}${execpi .001 (nvidia-smi --query-gpu=utilization.gpu --format=csv,noheader)}
${color1}Ram:${color green}${execpi .001 (nvidia-smi --query-gpu=utilization.memory --format=csv,noheader)}
${color1}Pwr:${color green}${execpi .001 (nvidia-smi --query-gpu=power.draw --format=csv,noheader)}
${alignr}${color1}Freq: ${color green}${execpi .001 (nvidia-smi --query-gpu=clocks.mem --format=csv,noheader)}
${color orange}${hr 1}${endif}
```

`~/.bashrc` Terminal splash screen

This example modifies `~/.bashrc` to display information on a splash screen each time the terminal is opened or whenever you type `. .bashrc` at the shell prompt.

In addition to `neofetch` answered previously, there is `screenfetch` which looks a lot nicer (IMO). Plus another answer mentions he doesn't know how to get iGPU listed and this does it:

```
rick@alien: ~

Weather report: Edmonton                    April 2018
                                         Su Mo Tu We Th Fr Sa
    \  /       Partly cloudy              1  2  3  4  5  6  7
  _ /""._.     7 °C                       8  9 10 11 12 13 14
    \_(   ).   ↓ 4 km/h                  15 16 17 18 19 20 21
    /(___(__)  14 km                     22 23 24 25 26 27 28
               0.0 mm                    29 30

                                                  05:03 PM

                        ./+o+-       rick@alien
                yyyyy- -yyyyyy+      OS: Ubuntu 16.04 xenial
             ://+//////-yyyyyyo      Kernel: x86_64 Linux 4.14.34-041434-generic
         .++ .:/++++++/-.+sss/`      Uptime: 21m
       .:++o:  /++++++++/:--:/-      Packages: 2211
      o:+o+:++.`..```.-/oo+++++/     Shell: bash 4.3.48
     .:+o:+o/.          `+sssoo+/   Resolution: 1920x2160
 .++/+:+oo+o:`             /sssooo. DE: Unity 7.4.0
/+++//+:`oo+o               /::--:. WM: Compiz
\+/+o+++`o++o               ++////. WM Theme: Ambiance
 .++.o+++oo+:`             /dddhhh. GTK Theme: Ambiance [GTK2/3]
     .+.o+oo:.          `oddhhhh+   Icon Theme: ubuntu-mono-dark
      \+.++o+o``-```.:ohdhhhhh+     Font: Ubuntu 11
       `:o+++ `ohhhhhhhhyo++os:     CPU: Intel Core i7-6700HQ CPU @ 3.5GHz
         .o:`.syhhhhhhh/.oo++o`     GPU: Mesa DRI Intel(R) HD Graphics 530 (Skylake GT2)
             /osyyyyyyo++ooo+++/    RAM: 2522MiB / 7581MiB
                 `` `` +oo+++o\:
                        `oo++.

rick@alien:~$ _
```

For details on setup see: [Terminal splash screen with Weather, Calendar, Time & Sysinfo?]

In summary just for the bottom section with Ubuntu display containing GPU information (second last line) use:

```
sudo apt install screenfetch
screenfetch
```

You'll want to put the `screenfetch` command an the bottom of your `~/.bashrc` file to have it appear every time you open the terminal.

Share Edit Follow Flag

edited Jun 12, 2020 at 14:37
Community Bot
1

answered Apr 18, 2018 at 23:05
WinEunuuchs2Unix
103k 37 245 417

7

Just to find the basics, according to https://wiki.debian.org/NvidiaGraphicsDrivers#NVIDIA_Proprietary_Driver,

```
lspci | grep VGA
```

If you need more detail than that, see @knb's answer to this same question.

Share Edit Follow Flag

edited Sep 22, 2017 at 20:50

answered May 4, 2017 at 21:11
mightypile
1,212 1 15 16

5

For nvidia GPUs, `nvidia-smi` command is your friend. See `man nvidia-smi` if you like to.

For listing GPUs use `nvidia-smi -L` (`nvidia-smi --list-gpus`), `nvidia-smi -q` give information about the gpu and the running processes.

Share Edit Follow Flag

answered Jan 21, 2019 at 15:19
Memin
171 1 2

5

If you have a AMD Radeon Card, you may want to run the following commands

```
sudo update-pciids #optional command, requires internet
lspci -nn | grep -E 'VGA|Display'
```

It should report something like this

```
00:01.0 VGA compatible controller [0300]: Advanced Micro Devices, Inc. [AMD/ATI] Wani [Radeon R5/R6/R7 Graphics]
[1002:9874] (rev c5)
03:00.0 Display controller [0380]: Advanced Micro Devices, Inc. [AMD/ATI] Sun XT [Radeon HD 8670A/8670M/8690M / R5 M330 /
M430 / R7 M520] [1002:6660] (rev ff)
```

Share Edit Follow Flag

answered May 27, 2018 at 11:04
Abbas Gadhia
491 1 5 10

This was also an issue solver for me with my NVIDIA card. Found this in the official CUDA installation guide: docs.nvidia.com/cuda/cuda-installation-guide-linux/... – boomkin Mar 31, 2020 at 8:41

The best answer! I have AMD Ryzen 7 6800u (onexplayer), Every tool shows just AMD/ATI Rembrandt. After this tip it shows Radeon 680M additionally. Kudos! – x'ES Mar 22 at 20:52

5

If you would like to have simple information, you could try gpustat. It is very good and simple.

```
❯❯❯ gpustat -cp
dali.vision  Thu Jun  2 23:46:16 2016
[0] GeForce GTX TITAN X | 77'C,  96 % | 11848 / 12287 MB | python/52046(11821M)
[1] GeForce GTX TITAN X | 75'C,   9 % | 11848 / 12287 MB | python/52046(11821M)
[2] GeForce GTX TITAN X | 75'C,  39 % | 12015 / 12287 MB | python/52046(11821M) python/128424(165M)
```

The author gives the following installation instructions:

> Install from PyPI:
>
> ```
> pip install gpustat
> ```
>
> To install the latest version (master branch) via pip:

```
pip install git+https://github.com/wookayin/gpustat.git@master
```

If you don't have root privilege, please try to install on user namespace: `pip install --user` . Note that from v0.4, `gpustat.py` is no more a zero-dependency executable. However, in rare cases you'll need a single executable script (legacy), you can also try:

```
wget https://raw.githubusercontent.com/wookayin/gpustat/v0.3.2/gpustat.py -O ~/.local/bin/gpustat chmod +x
~/.local/bin/gpustat    # Assuming ~/.local/bin is in your $PATH
```

Share Edit Follow Flag

edited Mar 28, 2019 at 1:10

answered Apr 24, 2018 at 1:43

Nufa
51 1 5

Only NVIDIA supported. – Protect children of Donbas2014 Nov 21, 2020 at 17:31

4

Well, this answer assumes you have a server with NVIDIA-GPUs. You have three ways:

1. To get just a short gist: `nvidia-smi`
2. To get a detailed one : `nvidia-smi -q` . You'll get multiple screens of detailed info if you more than 1 gpu.
3. Do a `ls /proc/driver/nvidia/gpus/` . It'll display the GPU-bus location as folders. Now, run the following command for each of the gpu bus locations. Fill `<gpu-id>` with bus-location: `cat /proc/driver/nvidia/gpus/<gpu_id>/information`

Share Edit Follow Flag

answered Jun 13, 2019 at 2:38

S Chakraborty
41 3

For AMD based graphics card(GPU), you can use `radeon-profile` application to get detailed information about the cards. It provides temperature, clock, Vram usage etc. (Github repository link)

2

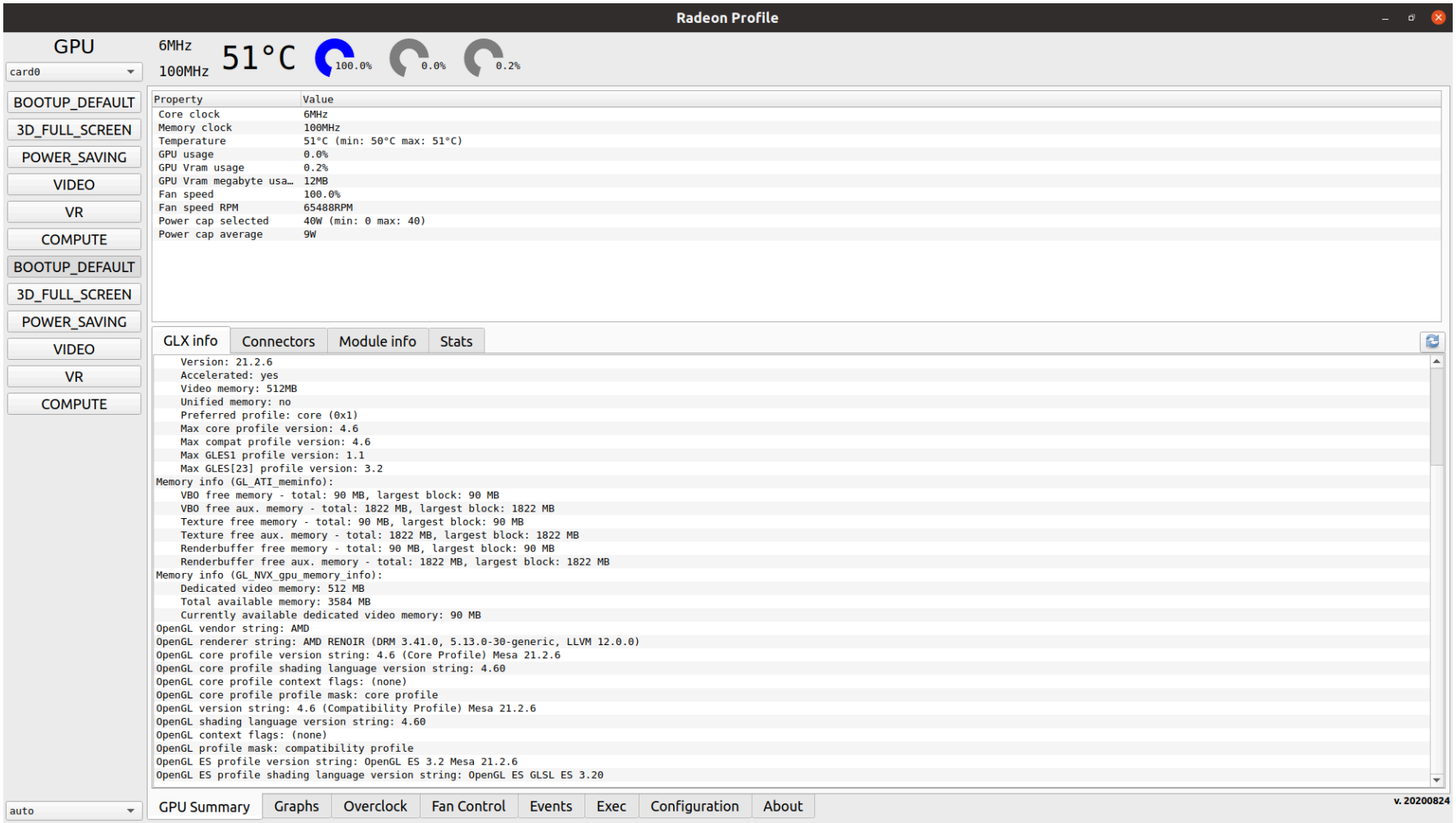
Step1: `sudo add-apt-repository ppa:radeon-profile/stable`

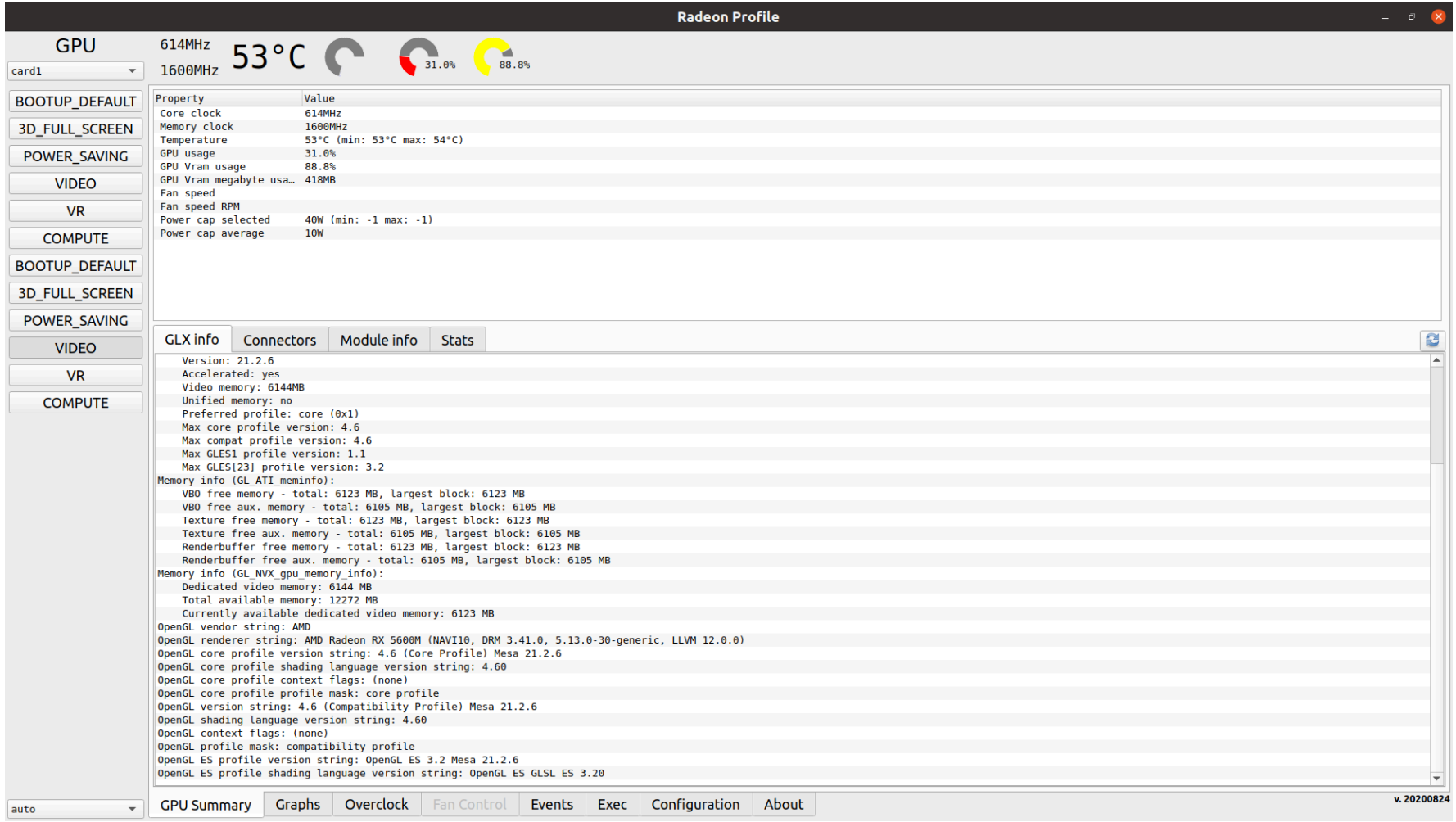
Step 2:

```
sudo apt update
sudo apt install radeon-profile
```

Step 3: run `radeon-profile`

Output screenshots (I have two GPU cards, card0(integrated) and card1(discrete)):





System specification:

- Ubuntu 20.04 LTS
- CPU: AMD® Ryzen 5 4600h
- GPU: AMD® Radeon rx 5600m

Source

Share Edit Follow Flag

answered Feb 23, 2022 at 15:47

Kevin Patel
152 8

2

If you're running Ubuntu on a Chromebook with crouton, the only one of the answers that will work is going to `chrome://gpu` in the Chrome browser.

Share Edit Follow Flag

edited Aug 25, 2015 at 19:19
David Foerster
36.4k 56 94 148

answered Aug 25, 2015 at 17:55
k26dr
191 1 1 5

1

For checking the utilization of Intel GPUs there is `intel_gpu_top` available since xenial (https://packages.ubuntu.com/bionic/intel-gpu-tools).

```
apt-get install intel-gpu-tools
```

There are just a few options available - see `man intel_gpu_top`.

Share Edit Follow Flag

answered Nov 15, 2020 at 20:17
wdp
179 1 11

`$ intel_gpu_top -d pci Failed to detect engines! (No such file or directory) (Kernel 4.16 or newer is required for i915 PMU support.) Segmentation fault (core dumped)` for me on a ChromeOS Flex NUC – paul_h Mar 21 at 11:17

1

In order to get all the information about the graphics processor, you can use the following command as specified by @greyfade.

```
> glxinfo
```

However, if the program `glxinfo` is currently not installed, you can install it by typing:

```
> sudo apt install mesa-utils
```

You will also have to enable the component called `universe`. Once this is done, `glxinfo` will list all the specifications related to the graphics processor in that environment.

Share Edit Follow Flag

answered Mar 26, 2019 at 5:58

Nayantara Jeyaraj
121 4

1

Tested for GTX 1080Ti - For clock rates on nvidia card -

```
nvidia-smi stats -d procClk corresponds to the Core clock
nvidia-smi stats -d memClk corresponds to the Memory clock
```

For monitoring every second -

```
watch -n 1 nvidia-smi stats -d procClk -c 1
```

For monitoring a subset of the devices every second -

```
watch -n 1 nvidia-smi stats -d procClk -c 1 -i 0,1
```

Based off on Elder Geek's answer with modifications to print every second to check for thermal throttling. If your GPU is being throttled, the procClk will drop significantly.

Share Edit Follow Flag

answered Dec 8, 2019 at 1:53

MonsieurBeilto
111 4

1

Use `lspci` , `lspci -v` to get basic info see here.

In my case for ex once I run `lspci` and I have got :

```
dina@dina-X450LA:~$ lspci
00:02.0 VGA compatible controller: Intel Corporation Haswell-ULT Integrated Graphics Controller (rev 0b)
00:03.0 Audio device: Intel Corporation Haswell-ULT HD Audio Controller (rev 0b)
00:14.0 USB controller: Intel Corporation 8 Series USB xHCI HC (rev 04)
00:16.0 Communication controller: Intel Corporation 8 Series HECI #0 (rev 04)
00:1b.0 Audio device: Intel Corporation 8 Series HD Audio Controller (rev 04)
00:1c.0 PCI bridge: Intel Corporation 8 Series PCI Express Root Port 1 (rev e4)
00:1c.2 PCI bridge: Intel Corporation 8 Series PCI Express Root Port 3 (rev e4)
00:1c.3 PCI bridge: Intel Corporation 8 Series PCI Express Root Port 4 (rev e4)
00:1d.0 USB controller: Intel Corporation 8 Series USB EHCI #1 (rev 04)
00:1f.0 ISA bridge: Intel Corporation 8 Series LPC Controller (rev 04)
00:1f.2 SATA controller: Intel Corporation 8 Series SATA Controller 1 [AHCI mode] (rev 04)
00:1f.3 SMBus: Intel Corporation 8 Series SMBus Controller (rev 04)
02:00.0 Ethernet controller: Qualcomm Atheros QCA8171 Gigabit Ethernet (rev 10)
03:00.0 Network controller: Ralink corp. RT3290 Wireless 802.11n 1T/1R PCIe
03:00.1 Bluetooth: Ralink corp. RT3290 Bluetooth
```

Share Edit Follow Flag

answered Feb 9, 2019 at 18:49

DINA TAKLIT
113 5

1

For my setup this is the cleanest:

```
python -c "import torch; print(torch.cuda.get_device_name(0));"
```

or

```
python -c "import uutils; gpu_name_otherwise_cpu(print_to_stdout=True);"
```

code:

```
def gpu_name_otherwise_cpu(print_to_stdout: bool = False):
    gpu_name_or_cpu = None
    try:
        gpu_name_or_cpu = torch.cuda.get_device_name(0)
    except:
        device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
        gpu_name_or_cpu = device
    print(f'{gpu_name_or_cpu=}') if print_to_stdout else None
    return gpu_name_or_cpu
```

Share Edit Follow Flag

answered Nov 7, 2022 at 22:43

Charlie Parker
435 1 5 11

1

For the Intel GMA950 (comes with EeePC in particular) you can run:

```
setpci -s 00:02.0 f0.b
```

which will return '00' for 200MHz, '01' for 250MHz or '03' for 400MHz. You may be able to apply the same principle to other Intel cards.

Share Edit Follow Flag

answered Feb 15, 2011 at 9:24

user10872

I'm getting 04. My GPU is 00:02.0 VGA compatible controller: Intel Corporation Mobile GM965/GL960 Integrated Graphics Controller (primary) (rev 03) (prog-if 00 [VGA controller]) – johny why Sep 19, 2019 at 19:29

1

Like in Chromium, if you have Firefox: about:support

Graphics

Features	
Compositing	Basic
Asynchronous Pan/Zoom	wheel input enabled; scrollbar drag enabled; keyboard enabled; autoscroll enabled; smooth pinch-zoom enabled
WebGL 1 Driver WSI Info	GLX 1.4 GLX_VENDOR(client): Mesa Project and SGI GLX_VENDOR(server): SGI Extensions: GLX_ARB_create_context GLX_ARB_create_context_no_error GLX_ARB_create_context_profile GLX_ARB_create_context_robustness GLX_ARB_fbconfig_float GLX_ARB_framebuffer_sRGB GLX_ARB_get_proc_address GLX_ARB_multisample GLX_EXT_buffer_age GLX_EXT_create_context_es2_profile GLX_EXT_create_context_es_profile GLX_EXT_fbconfig_packed_float GLX_EXT_framebuffer_sRGB GLX_EXT_import_context GLX_EXT_texture_from_pixmap GLX_EXT_visual_info GLX_EXT_visual_rating GLX_INTEL_swap_event GLX_MESA_copy_sub_buffer GLX_MESA_query_renderer GLX_MESA_swap_control GLX_OML_swap_method GLX_OML_sync_control GLX_SGIS_multisample GLX_SGIX_fbconfig GLX_SGIX_pbuffer GLX_SGIX_visual_select_group GLX_SGI_make_current_read GLX_SGI_swap_control GLX_SGI_video_sync IsWebglOutOfProcessEnabled: 0
WebGL 1 Driver Renderer	Intel Open Source Technology Center -- Mesa DRI Intel(R) HD Graphics 4000 (IVB GT2)
WebGL 1 Driver Version	3.0 Mesa 20.2.6
WebGL 1 Driver Extensions	GL_ARB_multisample GL_EXT_abgr GL_EXT_bgra GL_EXT_blend_color GL_EXT_blend_minmax GL_EXT_blend_subtract GL_EXT_copy_texture GL_EXT_subtexture GL_EXT_texture_object GL_EXT_vertex_array GL_EXT_compiled_vertex_array GL_EXT_texture GL_EXT_texture3D GL_IBM_rasterpos_clip GL_ARB_point_parameters GL_EXT_draw_range_elements GL_EXT_packed_pixels GL_EXT_point_parameters GL_EXT_rescale_normal GL_EXT_separate_specular_color GL_EXT_texture_edge_clamp GL_SGIS_generate_mipmap GL_SGIS_texture_border_clamp GL_SGIS_texture_edge_clamp GL_SGIS_texture_lod GL_ARB_framebuffer_sRGB GL_ARB_multitexture GL_EXT_framebuffer_sRGB GL_IBM_multimode_draw_arrays GL_IBM_texture_mirrored_repeat GL_3DFX_texture_compression_FXT1 GL_ARB_texture_cube_map GL_ARB_texture_env_add GL_ARB_transpose_matrix GL_EXT_blend_func_separate GL_EXT_fog_coord GL_EXT_multi_draw_arrays GL_EXT_secondary_color GL_EXT_texture_env_add GL_EXT_texture_filter_anisotropic GL_EXT_texture_lod_bias GL_INGR_blend_func_separate GL_NV_blend_square GL_NV_light_max_exponent GL_NV_texgen_reflection GL_NV_texture_env_combine4 GL_S3_s3tc GL_SUN_multi_draw_arrays GL_ARB_texture_border_clamp GL_ARB_texture_compression GL_EXT_framebuffer_object GL_EXT_texture_compression_s3tc GL_EXT_texture_env_combine GL_EXT_texture_env_dot3 GL_MESA_window_pos GL_NV_packed_depth_stencil GL_NV_texture_rectangle GL_ARB_depth_texture GL_ARB_occlusion_query GL_ARB_shadow GL_ARB_texture_env_combine GL_ARB_texture_env_crossbar GL_ARB_texture_env_dot3 GL_ARB_texture_mirrored_repeat GL_ARB_window_pos GL_EXT_stencil_two_side GL_EXT_texture_cube_map GL_NV_depth_clamp GL_NV_fog_distance GL_APPLE_packed_pixels GL_ARB_draw_buffers GL_ARB_fragment_program GL_ARB_fragment_shader GL_ARB_shader_objects GL_ARB_vertex_program GL_ARB_vertex_shader GL_ATI_draw_buffers GL_ATI_texture_env_combine3 GL_ATI_texture_float GL_EXT_shadow_funcs GL_EXT_stencil_wrap GL_MESA_pack_invert GL_NV_primitive_restart GL_ARB_depth_clamp GL_ARB_fragment_program_shadow GL_ARB_half_float_pixel GL_ARB_occlusion_query2 GL_ARB_point_sprite GL_ARB_shading_language_100 GL_ARB_sync GL_ARB_texture_non_power_of_two GL_ARB_vertex_buffer_object GL_ATI_blend_equation_separate GL_EXT_blend_equation_separate GL_OES_read_format GL_ARB_color_buffer_float GL_ARB_pixel_buffer_object GL_ARB_texture_compression_rgtc GL_ARB_texture_float GL_ARB_texture_rectangle GL_EXT_packed_float GL_EXT_pixel_buffer_object GL_EXT_texture_compression_dxt1 GL_EXT_texture_compression_rgtc GL_EXT_texture_rectangle GL_EXT_texture_sRGB GL_EXT_texture_shared_exponent GL_ARB_framebuffer_object GL_EXT_framebuffer_blit GL_EXT_framebuffer_multisample GL_EXT_packed_depth_stencil GL_APPLE_object_purgeable GL_ARB_vertex_array_object GL_ATI_separate_stencil GL_EXT_draw_buffers2 GL_EXT_draw_instanced GL_EXT_gpu_program_parameters GL_EXT_gpu_shader4 GL_EXT_texture_array GL_EXT_texture_integer GL_EXT_texture_sRGB_decode GL_EXT_timer_query GL_OES_EGL_image GL_AMD_texture_texture4 GL_ARB_copy_buffer GL_ARB_depth_buffer_float GL_ARB_draw_instanced GL_ARB_half_float_vertex GL_ARB_instanced_arrays GL_ARB_map_buffer_range GL_ARB_texture_rg GL_ARB_texture_swizzle GL_ARB_vertex_array_bgra GL_EXT_texture_swizzle GL_EXT_vertex_array_bgra GL_NV_conditional_render GL_AMD_conservative_depth GL_AMD_draw_buffers_blend GL_AMD_seamless_cubemap_per_texture GL_ARB_ES2_compatibility GL_ARB_blend_func_extended GL_ARB_debug_output GL_ARB_draw_buffers_blend GL_ARB_draw_elements_base_vertex GL_ARB_explicit_attrib_location GL_ARB_fragment_coord_conventions GL_ARB_provoking_vertex GL_ARB_sample_shading GL_ARB_sampler_objects GL_ARB_seamless_cube_map GL_ARB_shader_texture_lod GL_ARB_texture_cube_map_array GL_ARB_texture_gather GL_ARB_texture_multisample GL_ARB_texture_query_lod GL_ARB_texture_rgb10_a2ui GL_ARB_uniform_buffer_object GL_ARB_vertex_type_2_10_10_10_rev GL_EXT_provoking_vertex GL_EXT_texture_snorm GL_MESA_texture_signed_rgba GL_NV_texture_barrier GL_ARB_draw_indirect GL_ARB_get_program_binary GL_ARB_robustness GL_ARB_separate_shader_objects GL_ARB_shader_bit_encoding GL_ARB_shader_precision GL_ARB_texture_compression_bptc GL_ARB_timer_query GL_ARB_transform_feedback2 GL_ARB_transform_feedback3 GL_EXT_direct_state_access GL_AMD_multi_draw_indirect GL_ANGLE_texture_compression_dxt3 GL_ANGLE_texture_compression_dxt5 GL_ARB_compressed_texture_pixel_storage GL_ARB_conservative_depth GL_ARB_internalformat_query GL_ARB_map_buffer_alignment GL_ARB_shader_atomic_counters GL_ARB_shader_image_load_store GL_ARB_shading_language_420pack GL_ARB_shading_language_packing GL_ARB_texture_storage GL_ARB_transform_feedback_instanced GL_EXT_framebuffer_multisample_blit_scaled GL_EXT_transform_feedback GL_AMD_shader_trinary_minmax GL_ARB_ES3_compatibility GL_ARB_arrays_of_arrays GL_ARB_clear_buffer_object GL_ARB_compute_shader GL_ARB_copy_image GL_ARB_explicit_uniform_location GL_ARB_fragment_layer_viewport GL_ARB_framebuffer_no_attachments GL_ARB_invalidate_subdata GL_ARB_multi_draw_indirect GL_ARB_program_interface_query GL_ARB_shader_image_size GL_ARB_shader_storage_buffer_object GL_ARB_texture_query_levels GL_ARB_texture_storage_multisample GL_ARB_texture_view GL_ARB_vertex_attrib_binding GL_KHR_debug GL_KHR_robustness GL_ARB_buffer_storage GL_ARB_clear_texture GL_ARB_compute_variable_group_size GL_ARB_indirect_parameters GL_ARB_internalformat_query2 GL_ARB_multi_bind GL_ARB_seamless_cubemap_per_texture GL_ARB_shader_draw_parameters GL_ARB_shader_group_vote GL_ARB_shading_language_include GL_ARB_texture_mirror_clamp_to_edge GL_ARB_vertex_type_10f_11f_11f_rev GL_EXT_shader_integer_mix GL_INTEL_performance_query GL_ARB_clip_control GL_ARB_conditional_render_inverted GL_ARB_cull_distance GL_ARB_derivative_control GL_ARB_get_texture_sub_image GL_ARB_pipeline_statistics_query GL_ARB_shader_texture_image_samples GL_ARB_texture_barrier GL_ARB_transform_feedback_overflow_query GL_EXT_polygon_offset_clamp GL_KHR_blend_equation_advanced GL_KHR_context_flush_control GL_ARB_parallel_shader_compile GL_ARB_shader_atomic_counter_ops GL_ARB_shader_clock GL_EXT_shader_samples_identical GL_KHR_no_error GL_ARB_gl_spirv GL_ARB_spirv_extensions GL_MESA_shader_integer_functions GL_ARB_polygon_offset_clamp GL_ARB_texture_filter_anisotropic GL_KHR_parallel_shader_compile GL_EXT_EGL_image_storage GL_EXT_shader_framebuffer_fetch_non_coherent GL_EXT_texture_sRGB_R8 GL_EXT_texture_shadow_lod GL_INTEL_blackhole_render GL_NV_compute_shader_derivatives GL_EXT_EGL_sync GL_EXT_demote_to_helper_invocation
WebGL 1 Extensions	ANGLE_instanced_arrays EXT_blend_minmax EXT_color_buffer_half_float EXT_float_blend EXT_frag_depth EXT_shader_texture_lod EXT_sRGB EXT_texture_compression_bptc EXT_texture_compression_rgtc EXT_texture_filter_anisotropic MOZ_debug OES_element_index_uint OES_fbo_render_mipmap OES_standard_derivatives OES_texture_float OES_texture_float_linear OES_texture_half_float OES_texture_half_float_linear OES_vertex_array_object WEBGL_color_buffer_float WEBGL_compressed_texture_etc WEBGL_compressed_texture_s3tc WEBGL_compressed_texture_s3tc_srgb WEBGL_debug_renderer_info WEBGL_debug_shaders WEBGL_depth_texture WEBGL_draw_buffers WEBGL_lose_context
WebGL 2 Driver WSI Info	GLX 1.4 GLX_VENDOR(client): Mesa Project and SGI GLX_VENDOR(server): SGI Extensions: GLX_ARB_create_context GLX_ARB_create_context_no_error GLX_ARB_create_context_profile GLX_ARB_create_context_robustness GLX_ARB_fbconfig_float GLX_ARB_framebuffer_sRGB GLX_ARB_get_proc_address GLX_ARB_multisample GLX_EXT_buffer_age GLX_EXT_create_context_es2_profile GLX_EXT_create_context_es_profile GLX_EXT_fbconfig_packed_float GLX_EXT_framebuffer_sRGB GLX_EXT_import_context GLX_EXT_texture_from_pixmap GLX_EXT_visual_info GLX_EXT_visual_rating GLX_INTEL_swap_event GLX_MESA_copy_sub_buffer GLX_MESA_query_renderer GLX_MESA_swap_control GLX_OML_swap_method GLX_OML_sync_control GLX_SGIS_multisample GLX_SGIX_fbconfig GLX_SGIX_pbuffer GLX_SGIX_visual_select_group GLX_SGI_make_current_read GLX_SGI_swap_control GLX_SGI_video_sync IsWebglOutOfProcessEnabled: 0
WebGL 2 Driver Renderer	Intel Open Source Technology Center -- Mesa DRI Intel(R) HD Graphics 4000 (IVB GT2)
WebGL 2 Driver Version	4.2 (Core Profile) Mesa 20.2.6

Share Edit Follow Flag

edited Mar 4, 2021 at 0:53

answered Mar 4, 2021 at 0:35

Vitaly Zdanevich
1,114 5 19 40

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.