# Vision

***Release 0.1***

**Pedro Salgueiro, Javier Leon**

**Sep 20, 2022**

# CONTENTS

In this page you can find documentation on how to the Vision supercomputer, including information on how to submit and manage jobs in the best possible way.

CHAPTER

ONE

# CONTENTS

## 1.1 About Vision

The Vision supercomputer is made by 2 compute nodes and a management node. Each compute node is a NVIDIA DGX A100 systems with the following specifications:

- CPU: Dual AMD Rome 7742, 128 cores total
- System Memory: 1TB
- GPUs: 8x NVIDIA A100 Tensor Core GPUs (40GB per GPU)
- GPU Memory: 320GB total

The two DGX A100 systems are interconneted by 8 x 200Gb/s HDR InfiniBand links.

### 1.1.1 Storage

The users have access to two main storage areas in Vision:

- Home folder
- External storage

#### Home folder

The home folder is located in `/home/<username>` and is shared between all nodes of the cluster (management node and compute nodes).

This folder should be used to store personal files and source code for the applications of the user.

**Warning:** There are no backups of the files found in the home folders.

#### External storage

The Vision supercomputer has an external storage for data with the capacity of 15TB. This storage is made avaialbe in the management node and in the compute nodes via an NFS share mounted in `/data`. Each user has a folder in `/data/<username>` and/or `/data/projects/<project>/<username>`. The home folders and the data stored in `/data` is shared between all nodes.

This folder should be used to store data that will be processed by your applications. The data created by your applications should also be stored in this storage area.

**Warning:** This storage uses a RAID 0 volume to increase read performance while feeding data to the GPUs, wihtout any redundancy. If any of the drives fail, all data in the external storage will be lost. There are no backups of this volume.

## 1.2 How to access Vision

To access Vision, users should use SSH connect to the frontend frontend(management node) of the Vision, which allows them to access their data, submit jobs and check job status.

To access Vision via SSH, you should use the following settings:

- hostname: `main.vision.uevora.pt`
- port: `22`
- username: provided by the user
- password: provided by Vision administrators

### 1.2.1 Example

```
$: ssh user@main.vision.uevora.pt
[user@frontend ~]$
```

**Note:** Please note that users can only access Vision from the computer with the IP address provided by the user.

## 1.3 Quick start

In this page you can find a quick start guide on how to use Vision:

### 1.3.1 1. Access Vision

First you need to access Vision. To do so, you should use ssh with your credentials:

```
$: ssh user@main.vision.uevora.pt
[user@frontend ~]$
```

You can find more information on how to access Vision in *How to access Vision*.

### 1.3.2 2. Prepare your software environment

After logging in to Vision, you should change your working directory to your project directory and prepare the environment to run/build your application. You can use:

- *Python Environments*
- *Conda*
- *Software modules*

### 1.3.3 3. Create your slurm job script

After setting up your runtime environment, you should create a slurm job script, so that you can submit your job. The following is a Slurm job script to submit a project that uses Conda for dependency management:

```
#!/bin/bash
#SBATCH --job-name=cnn                      # create a short name for your job
#SBATCH --output="slurm-cnn-conda-%j.out" # %j will be replaced by the slurm jobID
#SBATCH --nodes=1                           # node count
#SBATCH --ntasks=1                          # total number of tasks across all nodes
#SBATCH --cpus-per-task=4                   # cpu-cores per task (>1 if multi-threaded␣
↪tasks)
#SBATCH --gres=gpu:2                        # number of gpus per node

source /opt/conda/bin/activate
conda activate tf-gpu

python3 cnn.py

conda deactivate
```

The script is made of two parts: 1) specification of the resources needed as well to run the job as some general job information; and 2) specification of the taks that will be run.

In the first part of the script, we define the job name, the output file and the requested resources (4 CPUs and 2 GPUs). Then, in the second part, we define the tasks of the job. When using Conda, we should run the following:

1. Activate the Conda environment;
2. Excecute the code;
3. Deactivate Conda environment;

You can find more information and examples on how to submit jobs in *Job examples*.

### 1.3.4 4. Submit the job

To submit the job, you should run the following command:

```
$ sbatch script_conda.sh
Submitted batch job 144
```

You can check the job status using the following command:

```
$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             143       batch      cnn     user  R       0:33      1 vision2
```

## 1.4 Job management

Jobs in Vision are managed by the Slurm Workload Manager, a job scheduler which schedules jobs according to the available computational resources and the requirements of each job.

### 1.4.1 How to submit a jobs

To submit a job in Slurm you have to first create a job script, which defines the job, the required resources by your job and the tasks that will run in the nodes. In particular, you should:

1. Access Vision: check *How to access Vision* for details
2. Copy your code to Vision: use `scp` or another tool that transfer files over SSH.
3. Create a Slurm job script: check *Slurm job scripts*. and *Job examples* for details.
4. Submit the job: see bellow.

### 1.4.2 Slurm job scripts

Slurm job scripts allow users to define the jobs that will run in the cluster. These are `bash` scripts which allows specify: 1) general properties of job, such as name an output files; 2) the resources that will be allocated to job; and 3) the code that will be run.

The following example represents a simple Slurm job script:

```
#!/bin/bash
#SBATCH --job-name=cnn                    # create a short name for your job
#SBATCH --output="slurm-cnn-venv-%j.out"  # %j will be replaced by the slurm jobID
#SBATCH --nodes=1                         # node count
#SBATCH --ntasks=1                        # total number of tasks across all nodes
#SBATCH --cpus-per-task=4                 # cpu-cores per task (>1 if multi-threaded␣
↪tasks)
#SBATCH --gres=gpu:2                      # number of gpus per node

source venv/bin/activate

python3 cnn.py

deactivate
```

In this script, we start by 1) defining the job name and the output file (lines 3-3); 2) specifying the required resources (4 CPUs and 2 GPUs) (lines 4-7); and 3) defining the code that will run (lines 9-13). For more information about this and other examples, please check *Job examples*.

### 1.4.3 Accounting

The resources used by a user are allways associated with project and are logged in order to control the cluster usage. To check the resources used in a time interval, the users should use the `sreport` command:

```
$ sreport cluster AccountUtilizationByUser account=prject start=0101 -T cpu,gres/gpu
--------------------------------------------------------------------------------
Cluster/Account/User Utilization 2022-01-01T00:00:00 - 2022-05-10T23:59:59 (11232000␣
↪secs)
Usage reported in TRES Minutes
--------------------------------------------------------------------------------
  Cluster         Account     Login     Proper Name       TRES Name      Used
--------- --------------- --------- --------------- -------------- ---------
      hpc       project-x                                     cpu   2863935
      hpc       project-x                                gres/gpu     88738
      hpc       project-x     user1           user1           cpu        17
      hpc       project-x     user1           user1       gres/gpu        14
      hpc       project-x     user2           user2           cpu   2863917
      hpc       project-x     user2           user2       gres/gpu     88724
```

In this example, the user is checking the resources consumed in the context of project `project-x` since January 1. Usage values are presented in minutes.

### 1.4.4 Common Slurm commands

In this section you can find some common useful commands to submit, manage and monitor jobs. You can find more detailed information on https://slurm.schedmd.com/quickstart.html.

#### Submit a job

To submit a job, you should use the `sbatch` command:

```
$: sbatch my-job-script.sh
Submitted batch job 439
```

In this example, the job was submited with the id 439.

#### Cancel a job

To cancel a job, the user should use the `scancel` command:

```
$: sbatch my-job-script.sh
Submitted batch job 439
```

In this example, the job was submited with the id 439.

### List job queue

To list the job queue, the user use should use the command `squeue`. This command lists all submited jobs to the cluster, including the job status and the node(s) where they are running:

```
$: squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             444     compute theJobNa     user  R      11:01      1 vision2
```

### List job information

To list detailed information about a job, the user should use the `scontrol` command. This list the relevant information about the job, including the requested resources, the job script and the output files. The user needs to know the job id:

```
$: scontrol show jobid <jobId>
JobId=444 JobName=The_Job_Name
   UserId=user(1000) GroupId=emedeiros(1000) MCS_label=N/A
   Priority=4294901696 Nice=0 Account=asr-pt QOS=normal
   JobState=RUNNING Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=00:18:08 TimeLimit=5-00:00:00 TimeMin=N/A
   SubmitTime=2022-05-11T10:32:29 EligibleTime=2022-05-11T10:32:29
   AccrueTime=2022-05-11T10:32:29
   StartTime=2022-05-11T10:32:30 EndTime=2022-05-16T10:32:30 Deadline=N/A
   SuspendTime=None SecsPreSuspend=0 LastSchedEval=2022-05-11T10:32:30
   Partition=compute AllocNode:Sid=vision1:3110592
   ReqNodeList=vision2 ExcNodeList=(null)
   NodeList=vision2
   BatchHost=vision2
   NumNodes=1 NumCPUs=255 NumTasks=1 CPUs/Task=255 ReqB:S:C:T=0:0:*:*
   TRES=cpu=255,mem=980288M,node=1,billing=255,gres/gpu=8
   Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
   MinCPUsNode=255 MinMemoryNode=980288M MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
   Command=/path/to/my-job-script.sh
   WorkDir=/path/of/my-job
   StdErr=/path/to/my-job-script.out
   StdIn=/dev/null
   StdOut=/path/to/my-job-script.out
   Power=
   TresPerNode=gpu:8
   NtasksPerTRES:0
```

#### Check node status

To check the status of eacch node of the cluster, users should use the `sinfo` command:

```
$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
compute*     up   infinite      1    mix vision1
compute*     up   infinite      1  alloc vision2
debug        up      15:00      1    mix vision1
debug        up      15:00      1  alloc vision2
```

## 1.5 Job examples

In this page you can find Slurm job script examples to submit jobs in Vision. Typically, AI applications use Conda or Python Environments to manage dependencies. The examples presented in this page are grouped by the dependency management system.

The source code of this examples can be found at: https://github.com/vistalab-uevora/vision-examples.

### 1.5.1 Examples:

#### Conda

In this page you can find Slurm job script examples that use Conda to manage the dependencies.

The source code of the exemples presented in this page can be found at: https://github.com/vistalab-uevora/vision-examples/tree/master/examples/conda.

#### Convolutional Neural Network (CNN)

This is an example on how to submit a Slurm job which uses Conda for dependency management. This example uses TensorFlow and is based on the official examples from TensorFlow: https://www.tensorflow.org/tutorials/images/cnn.

Conda is available in all nodes of Vision (head and compute nodes) in `/opt/conda/`. You can use this Conda version, or if you prefer, can install another version in your home folder: https://docs.conda.io/projects/conda/en/latest/user-guide/install/linux.html.

In this example we use Conda installed in `/opt/conda/`.

To submit a Python application that uses Conda for dependency management and project isolation as a Slurm job, you need to perform the following tasks:

1. Create the Conda environment with the project dependencies
2. Define the Slurm job script
3. Submit the Slurm job

### 1. Creating the Conda environment

To submit this script in Slurm using Conda for dependency management you should start by activating Conda:

```
$ source /opt/conda/etc/profile.d/conda.sh
```

and then create the Conda environment. To create the Conda environment, you can use:

- create it manually
- use an envirnment file

#### Creating the Conda environment manually

To create the Conda environment manually, you should start by creating the Conda environment:

```
(base) $ conda create -n tf-gpu tensorflow-gpu
```

activate the Conda environment:

```
(base) $ conda activate tf-gpu
```

and install the project dependencies:

```
(tf-gpu) $ pip install tensorflow==2.7.0
(tf-gpu) $ pip install matplotlib
```

After installing all dependencies, you should deactivate the virtual environment:

```
(tf-gpu) $ conda deactivate
```

#### Creating the Conda environment using an environment file:

To create the Conda environment from the environment file you should run the following command:

```
(base) $ conda env create -f environment.yml
```

This will create a Conda envirnment with the name and dependencies defined in the file environment.yml

### 2. Configure the Slurm job script

To submit the job, you first have to create a slurm batch script where you have to specify the required resources that will be allocated to your job and specify the tasks that will be run.

The following is a Slurm job script to run this project:

```
#!/bin/bash
#SBATCH --job-name=cnn           # create a short name for your job
#SBATCH --output="slurm-cnn-conda-%j.out"      # %j will be replaced by the slurm jobID
#SBATCH --nodes=1                # node count
#SBATCH --ntasks=1               # total number of tasks across all nodes
#SBATCH --cpus-per-task=4        # cpu-cores per task (>1 if multi-threaded tasks)
```

(continues on next page)

(continued from previous page)

```
#SBATCH --gres=gpu:2              # number of gpus per node

source /opt/conda/bin/activate
conda activate tf-gpu

python3 cnn.py

conda deactivate
```

The script is made of two parts: 1) specification of the resources needed as well to run the job as some general job information; and 2) specification of the taks that will be run.

In the first part of the script, we define the job name, the output file and the requested resources (4 CPUs and 2 GPUs). Then, in the second part, we define the tasks of the job. When using Conda, we should run the following:

1. Activate the Conda environment;
2. Excecute the code;
3. Deactivate Conda environment;

### 3. Submit the job

To submit the job, you should run the following command:

```
$ sbatch script_conda.sh
Submitted batch job 144
```

You can check the job status using the following command:

```
$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             143       batch      cnn     user  R       0:33      1 vision2
```

## Python Environments

In this page you can find Slurm job script examples that uses Python Environment to manage the dependencies.

The source code of the exemples presented in this page can be found at: https://github.com/vistalab-uevora/vision-examples/tree/master/examples/venv.

### Convolutional Neural Network (CNN)

This is an example on how to submit a Slurm job which uses Python Virtual Environment for dependency management. This example uses TensorFlow and is based in one of the official examples from TensorFlow: https://www.tensorflow.org/tutorials/images/cnn.

To submit a Python application that uses Python's Virtual Environments for dependency management and project isolation as a Slurm job, you need to perform the following tasks:

1. Create the Python Virtual Environment with the project dependencies
2. Define the Slurm job script

3. Submit the Slurm job

### 1. Creating the Python Virtual Environment

To submit this project in slurm using python virtualenv, you should start by creating the virtual env:

```
$ python3 -m venv ./venv
```

After creating the virtual env, you should activate it:

```
$ source ./venv/bin/activate
```

upgrade pip:

```
(venv) $ pip install --upgrade pip
```

#### Install dependencies

You can install the project dependencies manually or using a requirements file.

#### Install dependencies manually

To install the dependencies manually, you should run:

```
(venv) $ pip install tensorflow==2.7.0
(venv) $ pip install matplotlib
```

#### Install dependencies using a dependency file

To install the dependencies using a requirements file, you should run:

```
(venv) $ pip install -r requirements.txt
```

After installing all dependencies, you should deactivate the virtual environment:

```
(venv) $ deactivate
```

### 2. Configure the Slurm job script

To submit the job, you first have to create a slurm batch script where you have to specify the required resources that will be allocated to your job and specify the tasks that will be run.

The following is a Slurm job script to run this project:

```
#! /bin/bash
#SBATCH --job-name=cnn                      # create a short name for your job
#SBATCH --output="slurm-cnn-venv-%j.out"    # %j will be replaced by the slurm jobID
#SBATCH --nodes=1                           # node count
#SBATCH --ntasks=1                          # total number of tasks across all nodes
```

(continues on next page)

(continued from previous page)

```
#SBATCH --cpus-per-task=4                # cpu-cores per task (>1 if multi-threaded␣
↪tasks)
#SBATCH --gres=gpu:2                     # number of gpus per node

source venv/bin/activate

python3 cnn.py

deactivate
```

The script is made of two parts: 1) specification of the resources needed as well to run the job as some general job information; and 2) specification of the taks that will be run.

In the first part of the script, we define the job name, the output file and the requested resources (4 CPUs and 2 GPUs). Then, in the second part, we define the tasks of the job. When using Python Virtual Environments, we should run the following steps:

1. Activate the Python environment;
2. Excecute the code;
3. Deactivate the Python environment;

### 3. Submit the job

To submit the job, you should run the following command:

```
$ sbatch script.sh
Submitted batch job 143
```

You can check the job status using the following command:

```
$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
               143     batch      cnn     user  R       0:33      1 vision2
```

## 1.6 Software

Vision has available Python Environments and Anaconda, allowing users to create their own execution envirnment according to the needs of each application. Vision also has available software modules, that users can load to meet the dependencies of their applications.

### 1.6.1 Python Environments

To create a Pytohn Environment, the user can run the following command:

```
$ python3 -m venv ./venv
```

After creating the environment, the user can run the following command to activate the envirnment and install new packages:

```
$ source ./venv/bin/activate
(venv) $ pip install tensorflow==2.7.0
(venv) $ pip install matplotlib
```

### 1.6.2 Anaconda

Anaconda is available at /opt/conda/. To activate Anaconda, the user should run the following command:

```
$ source /opt/conda/etc/profile.d/conda.sh
```

To create a Conda environment, the use can run the following command:

```
(base) $ conda create -n tf-gpu tensorflow-gpu
```

In this example, we are creating environment named `tf-gpu` is created, based on the Conda environment `tensorflow`. After creating the environment, you can activate it and install new packages:

```
(base) $ conda activate tf-gpu
(tf-gpu) $ pip install tensorflow==2.7.0
(tf-gpu) $ pip install matplotlib
```

### 1.6.3 Software modules

To list the available software modules, the user should run the following command:

```
$ module av
```

To load a software module, the user should run, for example, the following command:

```
$ module load foss/2021a
```

You can learn more about software modules in *Software modules*.

## 1.7 Software modules

One of the available options to manage software dependencies in Vision is through the use of software modules, which sets the paths for the necessary binaries, libraries, manuals and include files for the software needed by each application.

The management of these software modules in Vision is achived with the help of EasyBuild together with Lmod to load the modules. You can learn more about EasyBuild and Lmod in https://easybuild.io/ and https://lmod.readthedocs.io/.

You can learn more abot using software modules in *Using modules*. The list of the software modules available in Vision can be found in *Available software modules*.

### 1.7.1 Using modules

To use software modules, the user just need to use the `module load` or `module add` commands to load the required module. When a module is loaded, all required dependencies will also be loaded, making the software ready for the user.

Vision uses a hierarchical module naming scheme (hmns) in which modules availability follows the software hierarchy Core/Compiler/MPI. Core refers to the basic core modules that have to be loaded in order to have access to next levels of software compiled against a specific compiler and a MPI API.

After logging into Vision, the user should execute the command `module av` (av for available) to list the available core modules:

```
$ module av

------------------------ /opt/software/modules/all/Core -------------------------
   Anaconda3/2021.05         binutils/2.36.1            (D)
   Bison/3.8.2               cuDNN/8.2.1.32-CUDA-11.3.1
   CUDA/11.3.1               flex/2.6.4
   GCC/10.3.0                foss/2021a
   GCCcore/10.2.0            gettext/0.21
   GCCcore/10.3.0    (D)     gompi/2021a
   Java/11.0.2       (11)    ncurses/6.2
   M4/1.4.19                 pkg-config/0.29.2
   OpenSSL/1.1               zlib/1.2.11
   binutils/2.35

  Where:
   Aliases:  Aliases exist: foo/1.2.3 (1.2) means that "module load foo/1.2" will load␣
↪foo/1.2.3
   D:        Default Module
```

In the available modules, you can find the toolchains foss/2021a and gompi/2021a, which you can load to access other modules which are built againts each toolchain, e.g.: `module load foss/2021a`. After loading the toolchain, you can view the associated modules using running again the comman `module av`:

```
$ module load foss/2021a
$ module av

----------- /opt/software/modules/all/MPI/GCC/10.3.0/OpenMPI/4.1.1 -----------
   FFTW/3.3.9                   (L)    h5py/3.2.1
   HDF5/1.10.7                         magma/2.6.1-CUDA-11.3.1
   ScaLAPACK/2.1.0-fb           (L)    matplotlib/3.4.2
   SciPy-bundle/2021.05                scikit-learn/0.24.2
   TensorFlow/2.5.3-CUDA-11.3.1        tensorboard/2.8.0
   TensorFlow/2.6.0-CUDA-11.3.1 (D)

--------------- /opt/software/modules/all/Compiler/GCC/10.3.0 ----------------
   FlexiBLAS/3.0.4 (L)    OpenBLAS/0.3.15 (L)    OpenMPI/4.1.1 (L)

------------- /opt/software/modules/all/Compiler/GCCcore/10.3.0 --------------
   Autoconf/2.71                       cppy/1.1.0
   Automake/1.16.3                     double-conversion/3.1.5
   Autotools/20210128                  expat/2.2.9
   Bazel/3.7.2                         expecttest/0.1.3
```

(continues on next page)

(continued from previous page)

```
   Bison/3.7.6                          flatbuffers-python/2.0
   Brotli/1.0.9                         flatbuffers/2.0.0
   CMake/3.20.1                         flex/2.6.4                 (D)
   DB/18.1.40                           fontconfig/2.13.93
   Eigen/3.3.9                          freetype/2.10.4
   FFmpeg/4.3.2                         gettext/0.21               (D)
   FriBidi/1.0.10                       giflib/5.2.1
   GDRCopy/2.2                          git/2.32.0-nodocs
   GMP/6.2.1                            gperf/3.1
   ICU/69.1                             groff/1.22.4
   JsonCpp/1.9.4                        gzip/1.10
   LAME/3.100                           help2man/1.48.3
   LMDB/0.9.28                          hwloc/2.4.1                (L)
   LibTIFF/4.2.0                        hypothesis/6.13.1
   M4/1.4.18                            intltool/0.51.0
   MPFR/4.1.0                           jbigkit/2.1
   Meson/0.58.0                         libarchive/3.5.1
   NASM/2.15.05                         libevent/2.1.12            (L)
   NCCL/2.10.3-CUDA-11.3.1              libfabric/1.12.1           (L)
   Ninja/1.10.2                         libffi/3.3
   PCRE/8.44                            libjpeg-turbo/2.0.6
   PMIx/3.2.3                   (L)     libpciaccess/0.16          (L)
   Perl/5.32.1-minimal                  libpng/1.6.37
   Perl/5.32.1                  (D)     libreadline/8.1
   Pillow/8.2.0                         libtool/2.4.6
   PyYAML/5.4.1                         libxml2/2.9.10             (L)
   Python/2.7.18-bare                   libyaml/0.2.5
   Python/3.9.5-bare                    lz4/1.9.3
   Python/3.9.5                 (D)     makeinfo/6.7-minimal
   Qhull/2020.2                         ncurses/6.2                (D)
   Rust/1.52.1                          nsync/1.24.0
   SQLite/3.35.4                        numactl/2.0.14             (L)
   Szip/2.1.1                           pkg-config/0.29.2          (D)
   Tcl/8.6.11                           pkgconfig/1.5.4-python
   Tk/8.6.11                            protobuf-python/3.17.3
   Tkinter/3.9.5                        protobuf/3.17.3
   UCX-CUDA/1.10.0-CUDA-11.3.1          pybind11/2.6.2
   UCX/1.10.0                   (L)     snappy/1.1.8
   UnZip/6.0                            typing-extensions/3.10.0.0
   X11/20210518                         util-linux/2.36
   XZ/5.2.5                     (L)     x264/20210414
   Yasm/1.3.0                           x265/3.5
   Zip/3.0                              xorg-macros/1.19.3
   binutils/2.36.1              (L,D)   zlib/1.2.11                (L,D)
   bzip2/1.0.8                          zstd/1.4.9
   cURL/7.76.0

------------------------ /opt/software/modules/all/Core ------------------------
   Anaconda3/2021.05          binutils/2.36.1
   Bison/3.8.2       (D)      cuDNN/8.2.1.32-CUDA-11.3.1
   CUDA/11.3.1                flex/2.6.4
   GCC/10.3.0        (L)      foss/2021a                 (L)
```

(continues on next page)

(continued from previous page)

```
   GCCcore/10.2.0             gettext/0.21
   GCCcore/10.3.0    (L,D)    gompi/2021a
   Java/11.0.2       (11)     ncurses/6.2
   M4/1.4.19         (D)      pkg-config/0.29.2
   OpenSSL/1.1       (L)      zlib/1.2.11
   binutils/2.35

  Where:
   L:        Module is loaded
   Aliases:  Aliases exist: foo/1.2.3 (1.2) means that "module load foo/1.2" will load␣
↪foo/1.2.3
   D:        Default Module
```

After loading the desired toolchain, the user can load the desired moduled using the `module load` command. For example, to load TensorFlow, the user can run the following command:

```
$ module load TensorFlow
```

To list the modules that are loaded, the user should use the `module list` command:

```
$ module list
Currently Loaded Modules:
  1) GCCcore/10.3.0                29) Tcl/8.6.11
  2) zlib/1.2.11                   30) SQLite/3.35.4
  3) binutils/2.36.1               31) GMP/6.2.1
  4) GCC/10.3.0                    32) libffi/3.3
  5) numactl/2.0.14                33) Python/3.9.5
  6) XZ/5.2.5                      34) pybind11/2.6.2
  7) libxml2/2.9.10                35) SciPy-bundle/2021.05
  8) libpciaccess/0.16             36) Szip/2.1.1
  9) hwloc/2.4.1                   37) HDF5/1.10.7
 10) OpenSSL/1.1                   38) h5py/3.2.1
 11) libevent/2.1.12               39) cURL/7.76.0
 12) UCX/1.10.0                    40) double-conversion/3.1.5
 13) libfabric/1.12.1              41) flatbuffers/2.0.0
 14) PMIx/3.2.3                    42) giflib/5.2.1
 15) OpenMPI/4.1.1                 43) ICU/69.1
 16) OpenBLAS/0.3.15               44) JsonCpp/1.9.4
 17) FlexiBLAS/3.0.4               45) NASM/2.15.05
 18) FFTW/3.3.9                    46) libjpeg-turbo/2.0.6
 19) ScaLAPACK/2.1.0-fb            47) LMDB/0.9.28
 20) foss/2021a                    48) nsync/1.24.0
 21) CUDA/11.3.1                   49) protobuf/3.17.3
 22) cuDNN/8.2.1.32-CUDA-11.3.1    50) protobuf-python/3.17.3
 23) GDRCopy/2.2                   51) flatbuffers-python/2.0
 24) UCX-CUDA/1.10.0-CUDA-11.3.1   52) typing-extensions/3.10.0.0
 25) NCCL/2.10.3-CUDA-11.3.1       53) libpng/1.6.37
 26) bzip2/1.0.8                   54) snappy/1.1.8
 27) ncurses/6.2                   55) TensorFlow/2.6.0-CUDA-11.3.1
 28) libreadline/8.1
```

If the user wants to "unload" the loaded modules, he can use the command `module purge`:

```
$ module purge
$ module list
No modules loaded
```

### 1.7.2 Available software modules

In this page you can find the list of the available software modules in Vision and instructions on how to load them.

#### Loading software modules

To load any of the modules found in the following list, you should first run the command `module spider` to list the modules that you should before loading the desired module.

For example, to load the Gromacs module, the user should first run:

```
$ module spider GROMACS
....
    You will need to load all module(s) on any one of the lines below before the
↪"GROMACS/2021.3-CUDA-11.3.1" module is available to load.

      GCC/10.3.0  OpenMPI/4.1.1
....
```

Then, the user can load all the needed software modules:

```
$ module load GCC/10.3.0  OpenMPI/4.1.1 GROMACS
```

#### List of software modules

The following is a list of software modules avaliable in Vision. You can check this list in Vision by running the command `module spider`:

```

----------------------------------------------------------------------------------------------------
↪----------------------------------------------------------------------------------------------------
↪----------
The following is a list of the modules and extensions currently available:
----------------------------------------------------------------------------------------------------
↪----------------------------------------------------------------------------------------------------
↪----------
  Anaconda3: Anaconda3/2021.05
    Built to complement the rich, open source Python community, the Anaconda platform␣
↪provides an enterprise-ready data analytics platform that empowers companies to adopt␣
↪a modern open
    data science analytics architecture.

  Autoconf: Autoconf/2.69, Autoconf/2.71
    Autoconf is an extensible package of M4 macros that produce shell scripts to␣
↪automatically configure software source code packages. These scripts can adapt the␣
↪packages to many
```

(continues on next page)

(continued from previous page)

```
    kinds of UNIX-like systems without manual user intervention. Autoconf creates a␣
↪configuration script for a package from a template file that lists the operating␣
↪system features that
    the package can use, in the form of M4 macro calls.

  Automake: Automake/1.16.1, Automake/1.16.2, Automake/1.16.3, Automake/1.16.4
    Automake: GNU Standards-compliant Makefile generator

  Autotools: Autotools/20180311, Autotools/20200321, Autotools/20210128, Autotools/
↪20210726
    This bundle collect the standard GNU build tools: Autoconf, Automake and libtool

  Bazel: Bazel/3.7.2
    Bazel is a build tool that builds code quickly and reliably. It is used to build the␣
↪majority of Google's software.

  Bison: Bison/3.5.3, Bison/3.7.1, Bison/3.7.6, Bison/3.8.2
    Bison is a general-purpose parser generator that converts an annotated context-free␣
↪grammar into a deterministic LR or generalized LR (GLR) parser employing LALR(1)␣
↪parser tables.

  Brotli: Brotli/1.0.9
    Brotli is a generic-purpose lossless compression algorithm that compresses data␣
↪using a combination of a modern variant of the LZ77 algorithm, Huffman coding and 2nd␣
↪order context
    modeling, with a compression ratio comparable to the best currently available␣
↪general-purpose compression methods. It is similar in speed with deflate but offers␣
↪more dense
    compression. The specification of the Brotli Compressed Data Format is defined in␣
↪RFC 7932.

  CMake: CMake/3.16.4, CMake/3.18.4, CMake/3.20.1
    CMake, the cross-platform, open-source build system. CMake is a family of tools␣
↪designed to build, test and package software.

  CUDA: CUDA/11.1.1, CUDA/11.3.1
    CUDA (formerly Compute Unified Device Architecture) is a parallel computing platform␣
↪and programming model created by NVIDIA and implemented by the graphics processing␣
↪units (GPUs)
    that they produce. CUDA gives developers access to the virtual instruction set and␣
↪memory of the parallel computational elements in CUDA GPUs.

  CUDAcore: CUDAcore/11.1.1
    CUDA (formerly Compute Unified Device Architecture) is a parallel computing platform␣
↪and programming model created by NVIDIA and implemented by the graphics processing␣
↪units (GPUs)
    that they produce. CUDA gives developers access to the virtual instruction set and␣
↪memory of the parallel computational elements in CUDA GPUs.

  Check: Check/0.15.2
    Check is a unit testing framework for C. It features a simple interface for defining␣
↪unit tests, putting little in the way of the developer. Tests are run in a separate␣
```

(continues on next page)

(continued from previous page)

```
↪address
    space, so both assertion failures and code errors that cause segmentation faults or␣
↪other signals can be caught. Test results are reportable in the following: Subunit,␣
↪TAP, XML, and
    a generic logging format.

  DB: DB/18.1.32, DB/18.1.40
    Berkeley DB enables the development of custom data management solutions, without the␣
↪overhead traditionally associated with such custom projects.

  Doxygen: Doxygen/1.8.17, Doxygen/1.9.1
    Doxygen is a documentation system for C++, C, Java, Objective-C, Python, IDL (Corba␣
↪and Microsoft flavors), Fortran, VHDL, PHP, C#, and to some extent D.

  Eigen: Eigen/3.3.8, Eigen/3.3.9
    Eigen is a C++ template library for linear algebra: matrices, vectors, numerical␣
↪solvers, and related algorithms.

  FFTW: FFTW/3.3.8, FFTW/3.3.9
    FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in␣
↪one or more dimensions, of arbitrary input size, and of both real and complex data.

  FFmpeg: FFmpeg/4.3.1, FFmpeg/4.3.2
    A complete, cross-platform solution to record, convert and stream audio and video.

  FlexiBLAS: FlexiBLAS/3.0.4
    FlexiBLAS is a wrapper library that enables the exchange of the BLAS and LAPACK␣
↪implementation used by a program without recompiling or relinking it.

  FriBidi: FriBidi/1.0.10
    The Free Implementation of the Unicode Bidirectional Algorithm.

  GCC: GCC/9.3.0, GCC/10.2.0, GCC/10.3.0
    The GNU Compiler Collection includes front ends for C, C++, Objective-C, Fortran,␣
↪Java, and Ada, as well as libraries for these languages (libstdc++, libgcj,...).

  GCCcore: GCCcore/9.3.0, GCCcore/10.2.0, GCCcore/10.3.0, GCCcore/11.2.0
    The GNU Compiler Collection includes front ends for C, C++, Objective-C, Fortran,␣
↪Java, and Ada, as well as libraries for these languages (libstdc++, libgcj,...).

  GDRCopy: GDRCopy/2.1-CUDA-11.1.1, GDRCopy/2.2
    A low-latency GPU memory copy library based on NVIDIA GPUDirect RDMA technology.

  GMP: GMP/6.2.0, GMP/6.2.1
    GMP is a free library for arbitrary precision arithmetic, operating on signed␣
↪integers, rational numbers, and floating point numbers.

  GROMACS: GROMACS/2021.3-CUDA-11.3.1
    GROMACS is a versatile package to perform molecular dynamics, i.e. simulate the␣
↪Newtonian equations of motion for systems with hundreds to millions of particles. This␣
↪is a GPU
    enabled build, containing both MPI and threadMPI builds. It also contains the gmxapi␣
```

(continues on next page)

(continued from previous page)

```
↪extension for the single precision MPI build.

  HDF: HDF/4.2.15
    HDF (also known as HDF4) is a library and multi-object file format for storing and␣
↪managing data between machines.

  HDF5: HDF5/1.10.6, HDF5/1.10.7
    HDF5 is a data model, library, and file format for storing and managing data. It␣
↪supports an unlimited variety of datatypes, and is designed for flexible and efficient␣
↪I/O and for
    high volume and complex data.

  ICU: ICU/69.1
    ICU is a mature, widely used set of C/C++ and Java libraries providing Unicode and␣
↪Globalization support for software applications.

  Java: Java/11.0.2
    Java Platform, Standard Edition (Java SE) lets you develop and deploy Java␣
↪applications on desktops and servers.

  JsonCpp: JsonCpp/1.9.4
    JsonCpp is a C++ library that allows manipulating JSON values, including␣
↪serialization and deserialization to and from strings. It can also preserve existing␣
↪comment in
    unserialization/serialization steps, making it a convenient format to store user␣
↪input files.

  Julia: Julia/1.7.2-linux-x86_64
    Julia is a high-level, high-performance dynamic programming language for numerical␣
↪computing

  LAME: LAME/3.100
    LAME is a high quality MPEG Audio Layer III (MP3) encoder licensed under the LGPL.

  LMDB: LMDB/0.9.28
    LMDB is a fast, memory-efficient database. With memory-mapped files, it has the read␣
↪performance of a pure in-memory database while retaining the persistence of standard␣
↪disk-based
    databases.

  LibTIFF: LibTIFF/4.1.0, LibTIFF/4.2.0
    tiff: Library and tools for reading and writing TIFF data files

  Lua: Lua/5.4.3
    Lua is a powerful, fast, lightweight, embeddable scripting language. Lua combines␣
↪simple procedural syntax with powerful data description constructs based on␣
↪associative arrays and
    extensible semantics. Lua is dynamically typed, runs by interpreting bytecode for a␣
↪register-based virtual machine, and has automatic memory management with incremental␣
↪garbage
    collection, making it ideal for configuration, scripting, and rapid prototyping.
```

(continues on next page)

(continued from previous page)

```
  M4: M4/1.4.18, M4/1.4.19
    GNU M4 is an implementation of the traditional Unix macro processor. It is mostly␣
↪SVR4 compatible although it has some extensions (for example, handling more than 9␣
↪positional
    parameters to macros). GNU M4 also has built-in functions for including files,␣
↪running shell commands, doing arithmetic, etc.

  MPFR: MPFR/4.1.0
    The MPFR library is a C library for multiple-precision floating-point computations␣
↪with correct rounding.

  Meson: Meson/0.55.1-Python-3.8.2, Meson/0.55.3, Meson/0.58.0
    Meson is a cross-platform build system designed to be both as fast and as user␣
↪friendly as possible.

  NASM: NASM/2.15.05
    NASM: General-purpose x86 assembler

  NCCL: NCCL/2.8.3-CUDA-11.1.1, NCCL/2.10.3-CUDA-11.3.1
    The NVIDIA Collective Communications Library (NCCL) implements multi-GPU and multi-
↪node collective communication primitives that are performance optimized for NVIDIA␣
↪GPUs.

  Ninja: Ninja/1.10.0, Ninja/1.10.1, Ninja/1.10.2
    Ninja is a small build system with a focus on speed.

  OpenBLAS: OpenBLAS/0.3.12, OpenBLAS/0.3.15
    OpenBLAS is an optimized BLAS library based on GotoBLAS2 1.13 BSD version.

  OpenMPI: OpenMPI/4.0.3, OpenMPI/4.0.5, OpenMPI/4.1.1
    The Open MPI Project is an open source MPI-3 implementation.

  OpenSSL: OpenSSL/1.1
    The OpenSSL Project is a collaborative effort to develop a robust, commercial-grade,␣
↪full-featured, and Open Source toolchain implementing the Secure Sockets Layer (SSL v2/
↪v3) and
    Transport Layer Security (TLS v1) protocols as well as a full-strength general␣
↪purpose cryptography library.

  PCRE: PCRE/8.44
    The PCRE library is a set of functions that implement regular expression pattern␣
↪matching using the same syntax and semantics as Perl 5.

  PMIx: PMIx/3.1.5, PMIx/3.2.3
    Process Management for Exascale Environments PMI Exascale (PMIx) represents an␣
↪attempt to provide an extended version of the PMI standard specifically designed to␣
↪support clusters
    up to and including exascale sizes. The overall objective of the project is not to␣
↪branch the existing pseudo-standard definitions - in fact, PMIx fully supports both of␣
↪the
    existing PMI-1 and PMI-2 APIs - but rather to (a) augment and extend those APIs to␣
↪eliminate some current restrictions that impact scalability, and (b) provide a␣
```

(continues on next page)

(continued from previous page)

```
↪reference
   implementation of the PMI-server that demonstrates the desired level of scalability.

  Perl: Perl/5.30.2-minimal, Perl/5.30.2, Perl/5.32.0-minimal, Perl/5.32.0, Perl/5.32.1-
↪minimal, Perl/5.32.1, Perl/5.34.0
   Larry Wall's Practical Extraction and Report Language This is a minimal build␣
↪without any modules. Should only be used for build dependencies.

  Pillow: Pillow/8.0.1, Pillow/8.2.0
   Pillow is the 'friendly PIL fork' by Alex Clark and Contributors. PIL is the Python␣
↪Imaging Library by Fredrik Lundh and Contributors.

  PyTorch: PyTorch/1.10.0-CUDA-11.3.1
   Tensors and Dynamic neural networks in Python with strong GPU acceleration. PyTorch␣
↪is a deep learning framework that puts Python first.

  PyYAML: PyYAML/5.3.1, PyYAML/5.4.1
   PyYAML is a YAML parser and emitter for the Python programming language.

  Python: Python/2.7.18-bare, Python/2.7.18, Python/3.8.2, Python/3.8.6, Python/3.9.5-
↪bare, Python/3.9.5
   Python is a programming language that lets you work more quickly and integrate your␣
↪systems more effectively.

  Qhull: Qhull/2020.2
   Qhull computes the convex hull, Delaunay triangulation, Voronoi diagram, halfspace␣
↪intersection about a point, furthest-site Delaunay triangulation, and furthest-site␣
↪Voronoi
   diagram. The source code runs in 2-d, 3-d, 4-d, and higher dimensions. Qhull␣
↪implements the Quickhull algorithm for computing the convex hull.

  Rust: Rust/1.52.1
   Rust is a systems programming language that runs blazingly fast, prevents segfaults,␣
↪and guarantees thread safety.

  SQLite: SQLite/3.31.1, SQLite/3.33.0, SQLite/3.35.4
   SQLite: SQL Database Engine in a C Library

  ScaLAPACK: ScaLAPACK/2.1.0-fb, ScaLAPACK/2.1.0
   The ScaLAPACK (or Scalable LAPACK) library includes a subset of LAPACK routines␣
↪redesigned for distributed memory MIMD parallel computers.

  SciPy-bundle: SciPy-bundle/2020.11, SciPy-bundle/2021.05
   Bundle of Python packages for scientific software

  Szip: Szip/2.1.1
   Szip compression software, providing lossless compression of scientific data

  Tcl: Tcl/8.6.10, Tcl/8.6.11
   Tcl (Tool Command Language) is a very powerful but easy to learn dynamic programming␣
↪language, suitable for a very wide range of uses, including web and desktop␣
↪applications,
```

(continues on next page)

(continued from previous page)

```
    networking, administration, testing and many more.

  TensorFlow: TensorFlow/2.5.3-CUDA-11.3.1, TensorFlow/2.6.0-CUDA-11.3.1
    An open-source software library for Machine Intelligence

  Tk: Tk/8.6.11
    Tk is an open source, cross-platform widget toolchain that provides a library of␣
↪basic elements for building a graphical user interface (GUI) in many different␣
↪programming
    languages.

  Tkinter: Tkinter/3.9.5
    Tkinter module, built with the Python buildsystem

  UCX: UCX/1.8.0, UCX/1.9.0-CUDA-11.1.1, UCX/1.10.0, UCX/1.11.2
    Unified Communication X An open-source production grade communication framework for␣
↪data centric and high-performance applications

  UCX-CUDA: UCX-CUDA/1.10.0-CUDA-11.3.1
    Unified Communication X An open-source production grade communication framework for␣
↪data centric and high-performance applications This module adds the UCX CUDA support.

  UDUNITS: UDUNITS/2.2.26, UDUNITS/2.2.28
    UDUNITS supports conversion of unit specifications between formatted and binary␣
↪forms, arithmetic manipulation of units, and conversion of values between compatible␣
↪scales of
    measurement.

  UnZip: UnZip/6.0
    UnZip is an extraction utility for archives compressed in .zip format (also called
↪"zipfiles"). Although highly compatible both with PKWARE's PKZIP and PKUNZIP utilities␣
↪for MS-DOS
    and with Info-ZIP's own Zip program, our primary objectives have been portability␣
↪and non-MSDOS functionality.

  X11: X11/20200222, X11/20201008, X11/20210518
    The X Window System (X11) is a windowing system for bitmap displays

  XZ: XZ/5.2.5
    xz: XZ utilities

  Yasm: Yasm/1.3.0
    Yasm: Complete rewrite of the NASM assembler with BSD license

  Zip: Zip/3.0
    Zip is a compression and file packaging/archive utility. Although highly compatible␣
↪both with PKWARE's PKZIP and PKUNZIP utilities for MS-DOS and with Info-ZIP's own␣
↪UnZip, our
    primary objectives have been portability and other-than-MSDOS functionality

  binutils: binutils/2.34, binutils/2.35, binutils/2.36.1, binutils/2.37
    binutils: GNU binary utilities
```

(continues on next page)

(continued from previous page)

```
  bzip2: bzip2/1.0.8
    bzip2 is a freely available, patent free, high-quality data compressor. It typically␣
↪compresses files to within 10% to 15% of the best available techniques (the PPM family␣
↪of
    statistical compressors), whilst being around twice as fast at compression and six␣
↪times faster at decompression.

  cURL: cURL/7.69.1, cURL/7.72.0, cURL/7.76.0
    libcurl is a free and easy-to-use client-side URL transfer library, supporting DICT,␣
↪FILE, FTP, FTPS, Gopher, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP,␣
↪RTSP, SCP,
    SFTP, SMTP, SMTPS, Telnet and TFTP. libcurl supports SSL certificates, HTTP POST,␣
↪HTTP PUT, FTP uploading, HTTP form based upload, proxies, cookies, user+password␣
↪authentication
    (Basic, Digest, NTLM, Negotiate, Kerberos), file transfer resume, http proxy␣
↪tunneling and more.

  cppy: cppy/1.1.0
    A small C++ header library which makes it easier to write Python extension modules.␣
↪The primary feature is a PyObject smart pointer which automatically handles reference␣
↪counting
    and provides convenience methods for performing common object operations.

  cuDNN: cuDNN/8.0.4.30-CUDA-11.1.1, cuDNN/8.2.1.32-CUDA-11.3.1
    The NVIDIA CUDA Deep Neural Network library (cuDNN) is a GPU-accelerated library of␣
↪primitives for deep neural networks.

  double-conversion: double-conversion/3.1.5
    Efficient binary-decimal and decimal-binary conversion routines for IEEE doubles.

  expat: expat/2.2.9, expat/2.4.1
    Expat is an XML parser library written in C. It is a stream-oriented parser in which␣
↪an application registers handlers for things the parser might find in the XML document␣
↪(like
    start tags)

  expecttest: expecttest/0.1.3
    This library implements expect tests (also known as "golden" tests). Expect tests␣
↪are a method of writing tests where instead of hard-coding the expected output of a␣
↪test, you run
    the test to get the output, and the test framework automatically populates the␣
↪expected output. If the output of the test changes, you can rerun the test with the␣
↪environment
    variable EXPECTTEST_ACCEPT=1 to automatically update the expected output.

  flatbuffers: flatbuffers/2.0.0
    FlatBuffers: Memory Efficient Serialization Library

  flatbuffers-python: flatbuffers-python/2.0
    Python Flatbuffers runtime library.
```

(continues on next page)

(continued from previous page)

```
  flex: flex/2.6.4
    Flex (Fast Lexical Analyzer) is a tool for generating scanners. A scanner, sometimes␣
↪called a tokenizer, is a program which recognizes lexical patterns in text.

  fontconfig: fontconfig/2.13.92, fontconfig/2.13.93
    Fontconfig is a library designed to provide system-wide font configuration,␣
↪customization and application access.

  foss: foss/2021a
    GNU Compiler Collection (GCC) based compiler toolchain, including OpenMPI for MPI␣
↪support, OpenBLAS (BLAS and LAPACK support), FFTW and ScaLAPACK.

  fosscuda: fosscuda/2020b
    GCC based compiler toolchain __with CUDA support__, and including OpenMPI for MPI␣
↪support, OpenBLAS (BLAS and LAPACK support), FFTW and ScaLAPACK.

  freetype: freetype/2.10.1, freetype/2.10.3, freetype/2.10.4
    FreeType 2 is a software font engine that is designed to be small, efficient, highly␣
↪customizable, and portable while capable of producing high-quality output (glyph␣
↪images). It can
    be used in graphics libraries, display servers, font conversion tools, text image␣
↪generation tools, and many other products as well.

  gcccuda: gcccuda/2020b
    GNU Compiler Collection (GCC) based compiler toolchain, along with CUDA toolkit.

  gettext: gettext/0.20.1, gettext/0.21
    GNU 'gettext' is an important step for the GNU Translation Project, as it is an␣
↪asset on which we may build many other steps. This package offers to programmers,␣
↪translators, and
    even users, a well integrated set of tools and documentation

  giflib: giflib/5.2.1
    giflib is a library for reading and writing gif images. It is API and ABI compatible␣
↪with libungif which was in wide use while the LZW compression algorithm was patented.

  git: git/2.23.0-nodocs, git/2.28.0-nodocs, git/2.32.0-nodocs
    Git is a free and open source distributed version control system designed to handle␣
↪everything from small to very large projects with speed and efficiency.

  gompi: gompi/2020a, gompi/2021a
    GNU Compiler Collection (GCC) based compiler toolchain, including OpenMPI for MPI␣
↪support.

  gompic: gompic/2020b
    GNU Compiler Collection (GCC) based compiler toolchain along with CUDA toolkit,␣
↪including OpenMPI for MPI support with CUDA features enabled.

  gperf: gperf/3.1
    GNU gperf is a perfect hash function generator. For a given list of strings, it␣
↪produces a hash function and hash table, in form of C or C++ code, for looking up a␣
↪value depending
```

(continues on next page)

(continued from previous page)

```
    on the input string. The hash function is perfect, which means that the hash table␣
↪has no collisions, and the hash table lookup needs a single string comparison only.

  groff: groff/1.22.4
    Groff (GNU troff) is a typesetting system that reads plain text mixed with␣
↪formatting commands and produces formatted output.

  gzip: gzip/1.10
    gzip (GNU zip) is a popular data compression program as a replacement for compress

  h5py: h5py/3.2.1
    HDF5 for Python (h5py) is a general-purpose Python interface to the Hierarchical␣
↪Data Format library, version 5. HDF5 is a versatile, mature scientific software␣
↪library designed for
    the fast, flexible storage of enormous amounts of data.

  help2man: help2man/1.47.12, help2man/1.47.16, help2man/1.48.3
    help2man produces simple manual pages from the '--help' and '--version' output of␣
↪other commands.

  hwloc: hwloc/2.2.0, hwloc/2.4.1
    The Portable Hardware Locality (hwloc) software package provides a portable␣
↪abstraction (across OS, versions, architectures, ...) of the hierarchical topology of␣
↪modern
    architectures, including NUMA memory nodes, sockets, shared caches, cores and␣
↪simultaneous multithreading. It also gathers various system attributes such as cache␣
↪and memory
    information as well as the locality of I/O devices such as network interfaces,␣
↪InfiniBand HCAs or GPUs. It primarily aims at helping applications with gathering␣
↪information about
    modern computing hardware so as to exploit it accordingly and efficiently.

  hypothesis: hypothesis/5.41.2, hypothesis/5.41.5, hypothesis/6.13.1
    Hypothesis is an advanced testing library for Python. It lets you write tests which␣
↪are parametrized by a source of examples, and then generates simple and comprehensible␣
↪examples
    that make your tests fail. This lets you find more bugs in your code with less work.

  iimpi: iimpi/2021a, iimpi/2021b
    Intel C/C++ and Fortran compilers, alongside Intel MPI.

  imkl: imkl/2021.2.0, imkl/2021.4.0
    Intel oneAPI Math Kernel Library

  imkl-FFTW: imkl-FFTW/2021.4.0
    FFTW interfaces using Intel oneAPI Math Kernel Library

  impi: impi/2021.2.0, impi/2021.4.0
    Intel MPI Library, compatible with MPICH ABI

  intel: intel/2021a, intel/2021b
    Compiler toolchain including Intel compilers, Intel MPI and Intel Math Kernel␣
```

(continues on next page)

(continued from previous page)

```
↪Library (MKL).

  intel-compilers: intel-compilers/2021.2.0, intel-compilers/2021.4.0
    Intel C, C++ & Fortran compilers (classic and oneAPI)

  intltool: intltool/0.51.0
    intltool is a set of tools to centralize translation of many different file formats␣
↪using GNU gettext-compatible PO files.

  iompi: iompi/2021a
    Intel C/C++ and Fortran compilers, alongside Open MPI.

  jbigkit: jbigkit/2.1
    JBIG-KIT is a software implementation of the JBIG1 data compression standard (ITU-T␣
↪T.82), which was designed for bi-level image data, such as scanned documents.

  libarchive: libarchive/3.4.3, libarchive/3.5.1
    Multi-format archive and compression library

  libevent: libevent/2.1.11, libevent/2.1.12
    The libevent API provides a mechanism to execute a callback function when a specific␣
↪event occurs on a file descriptor or after a timeout has been reached. Furthermore,␣
↪libevent
    also support callbacks due to signals or regular timeouts.

  libfabric: libfabric/1.11.0, libfabric/1.12.1
    Libfabric is a core component of OFI. It is the library that defines and exports the␣
↪user-space API of OFI, and is typically the only software that applications deal with␣
↪directly.
    It works in conjunction with provider libraries, which are often integrated directly␣
↪into libfabric.

  libffi: libffi/3.3
    The libffi library provides a portable, high level programming interface to various␣
↪calling conventions. This allows a programmer to call any function specified by a call␣
↪interface
    description at run-time.

  libiconv: libiconv/1.16
    Libiconv converts from one character encoding to another through Unicode conversion

  libjpeg-turbo: libjpeg-turbo/2.0.5, libjpeg-turbo/2.0.6
    libjpeg-turbo is a fork of the original IJG libjpeg which uses SIMD to accelerate␣
↪baseline JPEG compression and decompression. libjpeg is a library that implements JPEG␣
↪image
    encoding, decoding and transcoding.

  libpciaccess: libpciaccess/0.16
    Generic PCI access library.

  libpng: libpng/1.6.37
    libpng is the official PNG reference library
```

(continues on next page)

(continued from previous page)

```
  libreadline: libreadline/8.0, libreadline/8.1
    The GNU Readline library provides a set of functions for use by applications that␣
↪allow users to edit command lines as they are typed in. Both Emacs and vi editing␣
↪modes are
    available. The Readline library includes additional functions to maintain a list of␣
↪previously-entered command lines, to recall and perhaps reedit those lines, and␣
↪perform csh-like
    history expansion on previous commands.

  libtirpc: libtirpc/1.3.2
    Libtirpc is a port of Suns Transport-Independent RPC library to Linux.

  libtool: libtool/2.4.6
    GNU libtool is a generic library support script. Libtool hides the complexity of␣
↪using shared libraries behind a consistent, portable interface.

  libxml2: libxml2/2.9.10
    Libxml2 is the XML C parser and toolchain developed for the Gnome project (but␣
↪usable outside of the Gnome platform).

  libyaml: libyaml/0.2.5
    LibYAML is a YAML parser and emitter written in C.

  lz4: lz4/1.9.2, lz4/1.9.3
    LZ4 is lossless compression algorithm, providing compression speed at 400 MB/s per␣
↪core. It features an extremely fast decoder, with speed in multiple GB/s per core.

  magma: magma/2.5.4, magma/2.6.1-CUDA-11.3.1
    The MAGMA project aims to develop a dense linear algebra library similar to LAPACK␣
↪but for heterogeneous/hybrid architectures, starting with current Multicore+GPU␣
↪systems.

  makeinfo: makeinfo/6.7-minimal
    makeinfo is part of the Texinfo project, the official documentation format of the␣
↪GNU project. This is a minimal build with very basic functionality. Should only be␣
↪used for build
    dependencies.

  matplotlib: matplotlib/3.4.2
    matplotlib is a python 2D plotting library which produces publication quality␣
↪figures in a variety of hardcopy formats and interactive environments across platforms.
↪ matplotlib can
    be used in python scripts, the python and ipython shell, web application servers,␣
↪and six graphical user interface toolkits.

  ncurses: ncurses/6.1, ncurses/6.2
    The Ncurses (new curses) library is a free software emulation of curses in System V␣
↪Release 4.0, and more. It uses Terminfo format, supports pads and color and multiple␣
↪highlights
    and forms characters and function-key mapping, and has all the other SYSV-curses␣
↪enhancements over BSD Curses.
```

(continues on next page)

(continued from previous page)

```
  ncview: ncview/2.1.8
    Ncview is a visual browser for netCDF format files. Typically you would use ncview␣
↪to get a quick and easy, push-button look at your netCDF files. You can view simple␣
↪movies of the
    data, view along various dimensions, take a look at the actual data values, change␣
↪color maps, invert the data, etc.

  netCDF: netCDF/4.7.4, netCDF/4.8.0
    NetCDF (network Common Data Form) is a set of software libraries and machine-
↪independent data formats that support the creation, access, and sharing of array-
↪oriented scientific
    data.

  netCDF-C++4: netCDF-C++4/4.3.1
    NetCDF (network Common Data Form) is a set of software libraries and machine-
↪independent data formats that support the creation, access, and sharing of array-
↪oriented scientific
    data.

  netCDF-Fortran: netCDF-Fortran/4.5.2, netCDF-Fortran/4.5.3
    NetCDF (network Common Data Form) is a set of software libraries and machine-
↪independent data formats that support the creation, access, and sharing of array-
↪oriented scientific
    data.

  networkx: networkx/2.5.1
    NetworkX is a Python package for the creation, manipulation, and study of the␣
↪structure, dynamics, and functions of complex networks.

  nsync: nsync/1.24.0
    nsync is a C library that exports various synchronization primitives, such as mutexes

  numactl: numactl/2.0.13, numactl/2.0.14
    The numactl program allows you to run your application program on specific cpu's and␣
↪memory nodes. It does this by supplying a NUMA memory policy to the operating system␣
↪before
    running your program. The libnuma library provides convenient ways for you to add␣
↪NUMA memory policies into your own program.

  pkg-config: pkg-config/0.29.2
    pkg-config is a helper tool used when compiling applications and libraries. It helps␣
↪you insert the correct compiler options on the command line so an application can use␣
↪gcc -o
    test test.c `pkg-config --libs --cflags glib-2.0` for instance, rather than hard-
↪coding values on where to find glib (or other libraries).

  pkgconfig: pkgconfig/1.5.4-python
    pkgconfig is a Python module to interface with the pkg-config command line tool

  protobuf: protobuf/3.14.0, protobuf/3.17.3
    Google Protocol Buffers
```

(continues on next page)

(continued from previous page)

```
  protobuf-python: protobuf-python/3.14.0, protobuf-python/3.17.3
    Python Protocol Buffers runtime library.

  pybind11: pybind11/2.6.0, pybind11/2.6.2
    pybind11 is a lightweight header-only library that exposes C++ types in Python and␣
↪vice versa, mainly to create Python bindings of existing C++ code.

  rootless-docker: rootless-docker

  scikit-build: scikit-build/0.11.1
    Scikit-Build, or skbuild, is an improved build system generator for CPython C/C++/
↪Fortran/Cython extensions.

  scikit-learn: scikit-learn/0.24.2
    Scikit-learn integrates machine learning algorithms in the tightly-knit scientific␣
↪Python world, building upon numpy, scipy, and matplotlib. As a machine-learning module,
↪ it
    provides versatile tools for data mining and analysis in any field of science and␣
↪engineering. It strives to be simple and efficient, accessible to everybody, and␣
↪reusable in
    various contexts.

  snappy: snappy/1.1.8
    Snappy is a compression/decompression library. It does not aim for maximum␣
↪compression, or compatibility with any other compression library; instead, it aims for␣
↪very high speeds
    and reasonable compression.

  tensorboard: tensorboard/2.8.0
    TensorBoard is a suite of web applications for inspecting and understanding your␣
↪TensorFlow runs and graphs.

  typing-extensions: typing-extensions/3.7.4.3, typing-extensions/3.10.0.0
    Typing Extensions - Backported and Experimental Type Hints for Python

  util-linux: util-linux/2.35, util-linux/2.36
    Set of Linux utilities

  x264: x264/20201026, x264/20210414
    x264 is a free software library and application for encoding video streams into the␣
↪H.264/MPEG-4 AVC compression format, and is released under the terms of the GNU GPL.

  x265: x265/3.3, x265/3.5
    x265 is a free software library and application for encoding video streams into the␣
↪H.265 AVC compression format, and is released under the terms of the GNU GPL.

  xorg-macros: xorg-macros/1.19.2, xorg-macros/1.19.3
    X.org macros utilities.

  zlib: zlib/1.2.11
    zlib is designed to be a free, general-purpose, legally unencumbered -- that is, not␣
```

(continues on next page)

(continued from previous page)

```
↪covered by any patents -- lossless data-compression library for use on virtually any␣
↪computer
    hardware and operating system.

  zstd: zstd/1.4.5, zstd/1.4.9
    Zstandard is a real-time compression algorithm, providing high compression ratios.␣
↪It offers a very wide range of compression/speed trade-off, while being backed by a␣
↪very fast
    decoder. It also offers a special mode for small data, called dictionary compression,
↪ and can create dictionaries from any sample set.

-------------------------------------------------------------------------------------------
↪-------------------------------------------------------------------------------------------
↪----------

To learn more about a package execute:

   $ module spider Foo

where "Foo" is the name of a module.

To find detailed information about a particular package you
must specify the version if there is more than one version:

   $ module spider Foo/11.1

-------------------------------------------------------------------------------------------
↪-------------------------------------------------------------------------------------------
↪----------
```