Software programs on NAS systems are managed as modules or packages. Among the NAS systems, Cabeus' aarch64-based GH200 GPU nodes running the TOSS5 operating system have a relatively small collection of sofware packages; the x86_64-based CPU-only Athena system running with TOSS5 mainly provides software packages from the Cray Programming Environment. The other x86_64-based CPU-only systms running with TOSS4 have the largest collection of software packages listed in tables below.

Note: The name of a software module or package may contain additional information, such as the vendor name, version number, or what compiler/library is used to build the software. For example:

- comp-intel/2020.4.304 - Intel Compiler version 2020.4.304
- mpi-hpe/mpt.2.30 - HPE MPI library version 2.30
- netcdf/4.4.1.1_mpt - NetCDF version 4.4.1.1, built with HPE MPT

Modules

Use the `module avail` command to see all available software modules. Run `module whatis` to view a short description of every module. For more information about a specific module, run `module help` *modulename*.

See Using Software Modules for more information.

Available Modules (as of September 22, 2026)

Module	Function
apptainer	open-source HPC container platform
boost	C++ library
cdo	Climate and NWP data analysis tool
comp-aocc	AMD Optimizing C/C++ Compiler
comp-intel	Intel Compiler
cuda	NVIDIA parallel computing platform and programming model
dakota	Software toolkit from Sandia
fieldview	Data visualization & analysis tool
firefox	Web browser
gcc	C/C++ compiler
grads	open-source tool for manipulation and visualization of Earth science data
hdf4	I/O library & tools
hdf5	I/O library & tools
idl	Data visualization & analysis tool
mathematica	System & language for mathematical application
matlab	Numerical computing environment & programming language
metis	Graph, mesh, & hypergraph partitioning software
mpi-hpe	HPE MPI library
mpi-intel	Intel MPI library
ncl	NCAR Command Language
nco	NetCDF Operators
ncview	Visual browser for NetCDF files
netcdf	I/O library
nvhpc	NVIDIA HPC Software Development Kit
octave	Numerical computations language
openfoam	Open source CFD software
paraview	Data analysis and scientific visualization application
parmetis	Math/numerical library
petsc	Portable, Extensible Toolkit for Scientific Computation
pkgsrc	NetBSD Package Source collection of software packages
pointwise	high-fidelity mesh generation software
python3	Programming language (version 3)
scicon	Tools created by NAS Scicon team
szip	Compression library
tecplot	Data visualization & analysis tool
totalview	Debugger
uprof	AMD's performance analysis tool suite
visit	open-source parallel interactive analysis tool for large-scale 2D/3D scientific simulation
vtune	Performance analysis tool
xxdiff	Graphical file & directories comparison & merge tool

Packages in the NetBSD Package Source Collection (pkgsrc)

Many software programs are now managed as packages on NAS systems, via the NetBSD Package Source Collection. To use a package, you must first access a package collection by loading one of the pkgsrc modules. A partial list of availalble packages in pkgsrc/2023Q3 available on NAS x84_64 systems are listd in the Modules table.

See Using Software in the NetBSD Packages Collection for more information.

Partial list of NetBSD Packages in pgksrc/2023Q3

Module	Function
bioperl	Perl tools for computational molecular biology
bison	GNU yacc(1) replacement
blas	Basic Linear Algebra System
bmake	Portable (autoconf) bersion of NetBSD "make" utility
clang	C language family front-end for LLVM
cscope	Interactive C program browser
fetch	Client to fetch URLs
ffmpeg4	Decoding, encoding, & streaming software (v4.x)
fftw	Fast C routines to compute DFTs
gdal	Translator library for raster geospacial data formats
gettext	Tools for providing messages in different languages
git	GIT version control suite meta-package
GMT	Generic Mapping Tools
gnuplot	Portable, interactive function-plotting utility
grace	GRaphing, Advanced Computation & Exploration of data
GraphicsMagick	X application for displaying & manipulating images
h5utils	Utilities for conversion to/from HDF5
hdf5-c++	New generation HDF C++ wrappers
htop	Enhanced version of "top" utility
ImageMagick	Package for display & interactive image manipulation
lrzip	Long range ZIP or Lzma RZIP
lz4	Extremely fast compression algorithm
lzop	Fast file compressor similar to gzip, using the LZO library
mercurial	Fast, lightweight source control management system
metis	Unstructured graph partitioning & sparse matrix ordering system
mg	Small, fast, public domain Emacs-style editor
nano	Small, friendly text editor (a replacement for Pico)
netcdf-fortran	Fortran support for NetCDF
nvi	Berkeley nvi with additional features
pbzip2	Parallel implementation of bzip2 block-sorting file compressor
pkgfind	Find packages by package name in pkgsrc
qgis	Geographic information system (GIS)
R	Statistical language for data analysis & graphics
rhash	Calculate/check CRC32, MD5, SHA1, GOST, or other hash sums
ruby	Ruby programming language
tkcvs	Tcl/Tk front-ends to CVS & diff
tmux	BSD-licensed terminal multiplexer (GNU screen alternative)
valgrind	Debugging & profiling tools
vtk	Visualization toolkit
wget	Retrieve files from the internet via HTTP & FTP

The NetBSD Packages Collection (pkgsrc) is a third-party software package management system for Unix-like operating systems. Among the over 17,000 packages available in pkgsrc, a few hundred of them have been built on NAS x86_64 systems and are available in the public directory /nasa/pkgsrc.

A new collection of software packages is released every quarter by the pkgsrc developers. In addition to new packages or updated versions, each release is largely comprised of the same packages (with identical version numbers) as previous releases.

Software Packages Available on NAS Systems

For a list of software programs that are currently available as packages, see Available Software Modules and Packages.

You can also see which pkgsrc modules are currently available by running module avail pkgsrc. For example:

```
pkgsrc/2022Q1-rome pkgsrc/2023Q3
```

are currently available on NAS systems with x86_64 architecture running TOSS 4.

All of the source distributions for building the pkgsrc modules are stored in the/nasa/pkgsrc/distfiles directory. If you plan to build software from source, check to see whether the source distribution is already there before downloading the files from the internet.

To view a complete list of all packages and a one-line description of each package on a NAS x86_64 system, load a pkgsrc module and type the pkg_info command without any additional options. If you are looking for a particular package, you can check whether it is available by using the grep command plus a keyword from the output of the pkg_info command.

TIP: Because some package names contain capital letters, add -i to the grep command to ignore case distinctions.

For example:

```
% module load pkgsrc/2023Q3
% pkg_info | grep -i cairo
cairo-1.16.0nb9         Vector graphics library with cross-device output support
cairo-gobject-1.16.0nb7 Vector graphics library with cross-device output support
py311-cairo-1.24.0      Python bindings for cairo
```

You can also use the -e option to test the existence of a given package. For example:

```
% pkg_info -e cairo
cairo-1.16.0nb4
```

Once you know the package name, you can use additional pkg_info options to find more information about the package. You can specify the full package name, or you can leave out the version number. For example:

```
% pkg_info -B cairo-1.16.0nb9
```

or

```
% pkg_info -B cairo
```

Some useful options for pkg_info include:

-B
: shows build information of the package

-d
: shows a long description of the package

-L
: shows the files within the package

-N
: shows what other packages were used to build the package

-R
: shows what other packages require the package

Using Packages

The root directory of a pkgsrc release (for example, /nasa/pkgsrc/toss4/2023Q3) contains familiar directories such as bin, sbin, lib, include, man, and info. For most packages, you can find their executables, libraries, or include files in these directories.

Using an Executable

When you load a pkgsrc module, the paths to its bin and sbin directories are prepended to the PATH environment variable, as shown in the following example. Therefore, you do not need to provide full paths to these directories in order to run executables in them.

```
% module show pkgsrc/2023Q3
-------------------------------------------------------------------
/nasa/modulefiles/pkgsrc/toss4/pkgsrc/2023Q3:
```

```
system          /bin/logger -p local2.info -t envmodule username display pkgsrc/2023Q3
module-whatis   A collection of software built via NetBSD's pkgsrc
prepend-path    INFOPATH /nasa/pkgsrc/toss4/2023Q3/info
prepend-path    MANPATH /nasa/pkgsrc/toss4/2023Q3/man
prepend-path    PATH /nasa/pkgsrc/toss4/2023Q3/bin:/nasa/pkgsrc/toss4/2023Q3/sbin
setenv          PKGSRC_BASE /nasa/pkgsrc/toss4/2023Q3
-------------------------------------------------------------------
```

Linking a Library

Unlike the PATH environment variable, loading a pkgsrc module does not set the LD_LIBRARY_PATH environment variable. The reason for this is to prevent libraries in the pkgsrc directory from overriding those with the same filenames at the system default locations or directories that may have been included in your existing LD_LIBRARY_PATH.

To link a static library (for example, libcairo.a) in the /nasa/pkgsrc/toss4/2023Q3/lib directory:

```
-I/nasa/pkgsrc/toss4/2023Q3/include
-L/nasa/pkgsrc/toss4/2023Q3/lib -lcairo
```

To link to a dynamic library (for example, libfftw.so) in the /nasa/pkgsrc/toss4/2023Q3/lib directory, follow the recommendations in the list below, in order of preference.

Note: Dynamic libraries have the suffix ".so" rather than the suffix ".a", which is used for static libraries.

- Set the LD_RUN_PATH environment variable as shown below prior to the link step. This will get /nasa/pkgsrc/toss4/2023Q3/lib encoded into the executable. Setting LD_RUN_PATH or LD_LIBRARY_PATH is not needed during run time.

  ```
  setenv LD_RUN_PATH /nasa/pkgsrc/toss4/2023Q3/lib
  -I/nasa/pkgsrc/toss4/2023Q3/include
  -L/nasa/pkgsrc/toss4/2023Q3/lib -lfftw
  ```

- Add the -Wl,-R linker option during the link step. This will also get /nasa/pkgsrc/toss4/2023Q3/lib encoded into the executable. Setting LD_RUN_PATH or LD_LIBRARY_PATH is not needed during run time.

  Note: The linker option -Wl,-R takes precedence over the LD_RUN_PATH setting.

  ```
  -I/nasa/pkgsrc/toss4/2023Q32/include
  -L/nasa/pkgsrc/toss4/2023Q3/lib -lfftw
  -Wl,-R/nasa/pkgsrc/toss4/2023Q3/lib
  ```

- Do not set LD_RUN_PATH or use the -Wl,-R linker option during the link step. Since /nasa/pkgsrc/toss4/2023Q3/lib is not encoded into the executable, you must set LD_LIBRARY_PATH during run time.

  Link time:

  ```
  -I/nasa/pkgsrc/toss4/2023Q3/include
  -L/nasa/pkgsrc/toss4/2023Q3/lib -lfftw
  ```

  Run time:

  ```
  setenv LD_LIBRARY_PATH "/nasa/pkgsrc/toss4/2023Q3/lib:$LD_LIBRARY_PATH"
  ```

For more information about using the setenv command, see **man setenv**.

The files of some packages are not installed in the bin, lib, and include directories. For example, the grace package is installed in the /nasa/pkgsrc/toss4/2023Q3/grace directory, which has its own bin, lib, and include subdirectories. The best way to find the exact location of a file you need from a package is to run the pkg_info command as follows:

```
pfe% pkg_info -L package_name
```

Separate modules have been created for some commonly used packages to make them easier to use—you can simply load the modulefile without loading a pkgsrc module.

On the NAS x86_64 nodes running TOSS 4, the following modules from pkgsrc have been made available:

- boost/1.78
- boost/1.83
- gcc/6.5
- gcc/7.5
- gcc/8.4
- gcc/9.3
- gcc/10.3
- gcc/13.2
- python3/3.9.12
- python3/3.11.5
- octave/6.4
- octave/8.3

Separate modules for other packages can be created when there is demand from the NAS user community.

For more information about pkgsrc, see https://www.pkgsrc.org.

See the commands described below to learn how to use modules.

Note: Software programs on NAS systems are managed either as modules or packages. For information about using software programs that are managed as packages, see Using Software in the NetBSD Packages Collection.

Using Modules

To find out what software modules are available on NAS systems, run the module avail command. You can also view a short description of every module by running module whatis, or find information about a specific module by running module help *modulename*.

To use the software provided in a module, you must first load the module by running the module load command. You can load several modules at the same time, as follows:

% module load *module_name_1 module_name_2 ... module_name_N*

Testing Modules

New software modules and new versions of existing modules are sometimes available for testing before they are released for general use. To access the test modules, run:

```
module use -a /nasa/modulefiles/testing
module avail
module load module_name
```

If you use the test modules, please send your feedback to support@nas.nasa.gov. Let us know whether the modules worked well or, if you had any problems, describe any steps you took to reproduce them.

Note: Test modules may be moved into production or deleted without notice.

Additional Module Commands

Some useful module commands include:

module list
: Lists which modules are already loaded in your environment.

module purge
: Removes all the modules that are loaded in your environment.

module switch *old_module_name new_module_name*
: Enables you to switch between two modules.

module show *module_name*
: Shows changes that will occur in your environment if you load *module_name.*

For more information, see **man module**.

TIP: If the module commands don't work, ensure that the following line is included in your .cshrc or .profile file:

- For csh: source /usr/local/lib/global.cshrc
- For bash: source /usr/local/lib/global.profile

Creating Your Own Modulefiles

If you maintain software for your own use or to share with your group, it may be useful to create modulefiles that can be loaded, purged, and so on, just like any other modulefile.

To create your own modulefile, you can simply copy an existing modulefile to your directory (commonly named /u/*your_username*/privatemodules), and edit it as appropriate.

First, find a module that you want to mimic and run module show *module_name*. The modulefile itself will be listed at the top of the module show output. You can find most available modulefiles in the /nasa/modulefiles/toss3 directory.

Here are some common features in well-designed modulefiles:

```
#%Module
##
##  All modulefiles must begin on the first line with those 8 characters
##  Here, in the comments, you can describe how the package was built, or, better yet,
##  put it in the ModulesHelp section below

proc ModulesHelp { } {
   puts stderr "Put the information you want to display when the user
   does module help on this"
   puts stderr "modulefile. Note each continuation line begins with puts
```

```
   stderr and comments in double quotes"
}

# include the following line only if you want to log the loading of the
# modulefile in our system logs
# most users would leave this out
source /nasa/modulefiles/common/log_module_usage.tcl

# if there might be several versions of this software, it would be good to
#set the version number
set version 1.0-something

# setting basepath or SWDIR is for naming the root directory where your
# various bin, include, lib directories reside
# see later in the modulefile on how it is used
set basepath /u/your_userid/dir_name/dir_name

# if this software requires other modules to be loaded first,
# then use the prereq specifier, for example:
prereq comp-intel

# if your software requires other software that you build,
#then you might simply load them, as in:
module load  my_other_module1  my_other_module2

# if this software conflicts with another software that the user
# may inadvertently load, then use:
conflict another_modulefile

# Finally, the actual setting of the PATHs etc.  You have a choice of either
# prepend-path or append-path:
prepend-path   PATH                $basepath/$version/bin
prepend-path   CPATH               $basepath/$version/include
prepend-path   FPATH               $basepath/$version/include
prepend-path   LD_LIBRARY_PATH          $basepath/$version/lib
prepend-path   LIBRARY_PATH        $basepath/$version/lib
prepend-path   MANPATH             $basepath/$version/share/man
```

If you plan to share this software with other users in your group, make sure that they have read permission to every file and directory, and execute permission on all directories and the executables in the software's bin directory.

In order to see your newly created modulefile in the output of the module avail command, run the module use command first, as follows:

```
% module use -a /u/your_username/privatemodules
```

Note: The -a option puts the new modulefile at the end of module avail output. Without it, the modulefile will be listed at the top.