



Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

# Spack

presented by

Teffy Sam

Fakultät für Mathematik, Informatik und Naturwissenschaften  
Fachbereich Informatik  
Arbeitsbereich Wissenschaftliches Rechnen

Course of Study: Ocean and Climate Physics  
Matrikelnummer: 7091617  
Supervisor: Dr. Michael Kuhn

Hamburg, 2020-08-29

# Abstract

Automated software installation tools are common on many Operating Systems. On Linux, most users would be familiar with the Apt package manager, other examples include RPM[6], Pacman[5]. Some package managers allow the user to build software from source, like Homebrew[4] on MacOS, BSD ports on FreeBSD[3] and so on. What is Spack? Spack is a simple package manager for High Performance Computing environments. It has been built to cover a wide array of software. Typical HPC software has many libraries and dependencies required to make a complete software package. Installing software from source on a Supercomputer is time consuming and a massive effort. There are many pieces in a supercomputer software ecosystem which include, but are not limited to, compilers, platforms, programming models etc.

Spack has very flexible syntax enabling a user to specify which parts of a software stack needs to be installed and how it needs to be done too. Why do we use Spack? Software build time is drastically reduced since almost everything is efficiently automated. How is Spack similar to other package managers? Package managers on different operating systems tend to install software as pre-compiled binaries. This may not be the best possible way to ensure, say, how a library is installed. If, for example, this library needs to be linked with another binary, one must ensure that both of them are built using the same compiler. Such incompatibilities are overlooked when installing a pre-compiled binary. Spack solves this problem by having a neat and organised dependency tree which is installed separately for an individual software. Each variation of a software install, even within the same version, are located separately too. Not to mention the fact that the whole process is automated at the same time giving the user control over which exact dependencies are installed.

# Contents

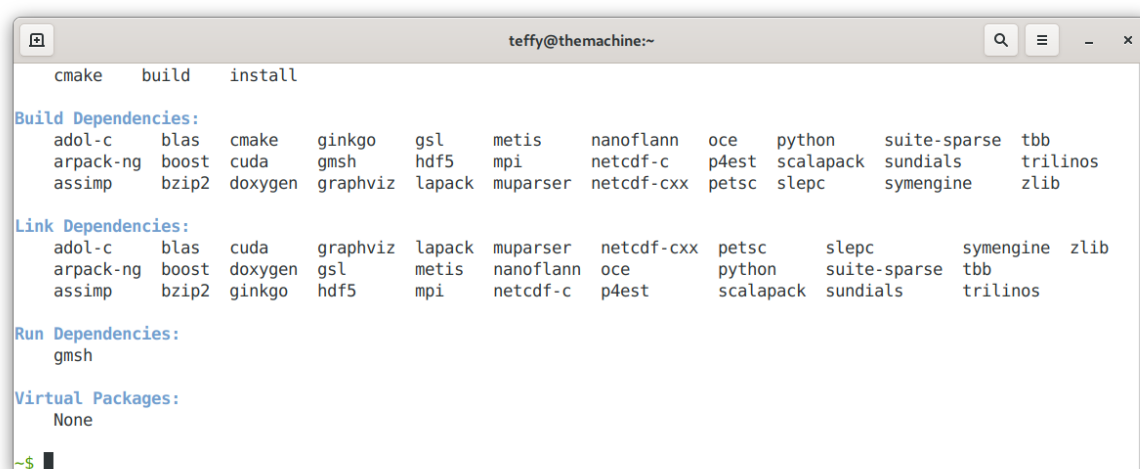
<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Current Complexity of Package Managers . . . . .	4
1.2	Containers and Environments . . . . .	5
1.3	Spack Usage . . . . .	6
1.4	Combinatorial Versioning . . . . .	9
<b>2</b>	<b>Conclusion</b>	<b>13</b>
	<b>Bibliography</b>	<b>14</b>
	<b>List of Figures</b>	<b>15</b>

# 1 Introduction

## 1.1 Current Complexity of Package Managers

Installing scientific software manually takes a lot of effort and time. The process begins with researching and figuring out what additional components are required to build the main software package. This then leads to installing and configuring the additional components which normally include libraries, compilers and other bits and pieces.

To give some perspective of the scale of how spread out a dependency tree can be, a finite element library software, dealii, is depicted in Fig 1.1. There are thirty three dependencies needed to install this software and trying to do so manually needs careful execution.



```
teffy@themachine:~  
cmake build install  
Build Dependencies:  
adol-c blas cmake ginkgo gsl metis nanoflann oce python suite-sparse tbb  
arpack-ng boost cuda gmesh hdf5 mpi netcdf-c p4est scalapack sundials trilinos  
assimp bzip2 doxygen graphviz lapack muparser netcdf-cxx petsc slepc symengine zlib  
Link Dependencies:  
adol-c blas cuda graphviz lapack muparser netcdf-cxx petsc slepc symengine zlib  
arpack-ng boost doxygen gsl metis nanoflann oce python suite-sparse tbb  
assimp bzip2 ginkgo hdf5 mpi netcdf-c p4est scalapack sundials trilinos  
Run Dependencies:  
gmsh  
Virtual Packages:  
None  
~$
```

Figure 1.1: Spack Package Dependencies

In general HPC systems use environment modules. Environment modules allow the user to list available software along with their different versions. The user has the freedom to quickly load and unload modules and use them for their projects without having to install anything. Ease of use (for the end user) is the primary feature here. But the problem with environment modules is that the software still needs to be compiled and built from source which is still a great effort for the system administrator. A user is also limited to whatever available software. The burden then falls on the system

administrator, rather than the user, to comply to the needs of individual software versions and configurations. This either results in the sysadmin having to maintain multiple versions of said software or it simply results in the specific software being unavailable for use.

The time to build and compile is drastically reduced as everything is streamlined. Manual labour is a lot lesser. Spack is accessible to users of HPC systems and even locally on personal computers too. Thanks to its refined flexibility Spack is gaining a lot of traction in HPC communities.

## 1.2 Containers and Environments

Containers provide a lot of portability but within the container a binary package manager is still required or packages have to be compiled from source. Spack also has container support, in that the user can deploy a Docker container with a constrained package base or environment in Spack that is easy to install with a few commands.

Containers like Docker[2] provide sand boxed environments and are good in terms of portability. Docker containers allow a higher level of accessibility to deploy, and run applications. Containers group together libraries and other dependencies for a software package. This ensures all settings and configurations of the software work in a pristine way in this closed environment. The operating system is usually stripped down to a bare bones Linux distribution with minimal system processes running in the background, in addition to the pieces required for the software to work. One small caveat is that the Docker daemon requires root access and such access is not possible for the common user base of a supercomputer. This isn't a major drawback per se but some of extra features of Docker containers remain out of reach for the user, for example mounting file systems works only with root access.

Spack[8] enables users to install software in home directories thereby circumventing the problem of root access that Docker containers have. Spack also brings together the possibility of combining a Docker container with Spack environment which in turn installs all necessary softwares. This can be especially powerful to deploy as this further improves the automating process of installing a software stack. At the same time Spack also has its own environment setup. This environment feature of Spack mirrors the portability of Docker in that it can be installed on different computers while maintaining compatibility with different Spack versions.

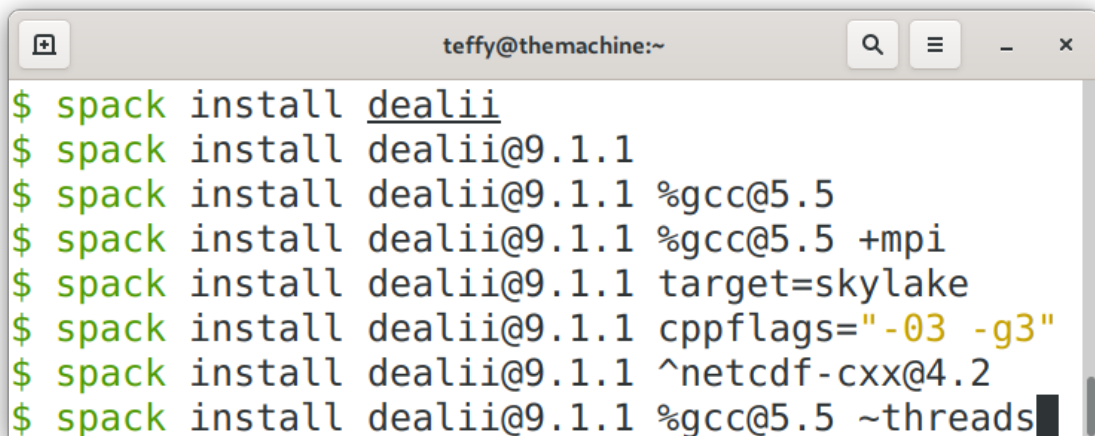
An environment consolidates a group of `spack specs` to enable simple deployment of a group of software.

## 1.3 Spack Usage

The Spack `spec` command lists out what will be installed given a certain specification or constraint of a software install.

Unconstrained package installs simply install the software with default settings, meaning it installs the latest version and builds using the default compiler on the system. But options to install any software (Fig 1.2) are easy with clauses:

- `@` lets the user specify a version of the software
- `%` lets you specify the compiler, compiler flags can be predefined before installation
- the operating system can be specified with `os=` and system architecture with `target=`
- with `^` dependencies for a certain software can be fine tuned recursively too. The `^` also allows a user to install two variations of a software package but say with different dependency versions
- `+` and `~/-` are used to choose build options



```
teffy@themachine:~  
$ spack install dealii  
$ spack install dealii@9.1.1  
$ spack install dealii@9.1.1 %gcc@5.5  
$ spack install dealii@9.1.1 %gcc@5.5 +mpi  
$ spack install dealii@9.1.1 target=skylake  
$ spack install dealii@9.1.1 cppflags="-O3 -g3"  
$ spack install dealii@9.1.1 ^netcdf-cxx@4.2  
$ spack install dealii@9.1.1 %gcc@5.5 ~threads
```

Figure 1.2: Spack Build Options

This in itself is a very useful tool when configuring a software stack for a project. With the addition of clauses to install a specific variation of software, the install can be said to be constrained. Fig 1.3 shows what the input specs are and if Spack concretize is run for an environment, Spack will save the state of the installed package list into that environment. This also lets the user add more constrained packages into an environment.

```

1: teffy@themachine:~
~$ spack spec netcdf-c@4.7%intel^openmpi@1.10.7
Input spec
-----
netcdf-c@4.7%intel
  ^openmpi@1.10.7

Concretized
-----
netcdf-c@4.7%intel@19.1.1.217~dap~hdf4~jna+mpi~parallel-netcdf+pic+shared arch=linux-arch-skylake
  ^hdf5@1.10.6%intel@19.1.1.217~cxx~debug~fortran+hl+mpi+pic+shared~szip~threadsafe api=none arch=linux-arch-skylake
    ^openmpi@1.10.7%intel@19.1.1.217~atomics~cuda~cxx~cxx_exceptions+gpfs~java~legacylaunchers~memchecker~pmi~singularity~sqlite3+static~thread_multiple+vt+wrapper-rpath fabrics=none patches=51e39ef20662d45a772f61db21cfac60f5339125b8cd8b7cd83dcdf4e8371366 schedulers=none arch=linux-arch-skylake
      ^hwloc@1.11.11%intel@19.1.1.217~cairo~cuda~gl~libudev+libxml2~netloc~nvml+pci+shared arch=linux-arch-skylake

2: teffy@themachine:~
~$ spack spec netcdf-c@4.7%gcc
Input spec
-----
netcdf-c@4.7%gcc

Concretized
-----
netcdf-c@4.7%gcc@10.1.0~dap~hdf4~jna+mpi~parallel-netcdf+pic+shared arch=linux-arch-skylake
  ^hdf5@1.10.6%gcc@10.1.0~cxx~debug~fortran+hl+mpi+pic+shared~szip~threadsafe api=none arch=linux-arch-skylake
    ^openmpi@3.1.6%gcc@10.1.0~atomics~cuda~cxx~cxx_exceptions+gpfs~java~legacylaunchers~memchecker~pmi~singularity~sqlite3+static~thread_multiple+vt+wrapper-rpath fabrics=none schedulers=none arch=linux-arch-skylake
      ^hwloc@1.11.11%gcc@10.1.0~cairo~cuda~gl~libudev+libxml2~netloc~nvml+pci+shared arch=linux-arch-skylake
        ^libpciaccess@0.13.5%gcc@10.1.0 arch=linux-arch-skylake
          ^libtool@2.4.6%gcc@10.1.0 arch=linux-arch-skylake

```

Figure 1.3: Spack Input Specs

Also shown in in Fig 1.3 is how the constraints for different openmpi versions are set by default in order to make the install successful. Installing netcdf-c with an older version of openmpi includes additional parts which Spack atomically takes care of. Spack then ensures all options and dependencies for the older openmpi line up to make the final build work flawlessly. This flexible modularity is where Spack shines when compared against other HPC package managers.

When an environment is initialised the user is given the option of choosing which software to install, whether to concretize that software or leave it as it is and then to finally proceed with the actual installation itself. This ensures stability is preserved within the environment over the course of upgrades too.

Environments install all its contained software over a single Spack call when installing software individually. Multiple specs of the same package can be contained in an

environment.



Figure 1.4: Package Managers

There are other package managers available too but they are limited in their use case. Conda is a popular package manager but it distributes binaries instead of building from source thereby leaving users with less freedom to tweak a software build. Conda and Python environments also have some of the feature set of Spack environments.

EasyBuild[9](Fig 1.4) is another HPC package manager but works on the module basis.

Spack currently holds a repository of more than 4300 packages. The `find` command lists all installed packages. Each install of a software has a unique hash which helps in distinguishing very similar installs of a package.



[illegible]

## 1.4 Combinatorial Versioning

Spack handles combinatorial versioning[7] in a much more efficient way than say Environment Modules[10]. From the installation layout, Spack extracts the directory tree structure or Directed Acyclic Graph and stores it into a hash for each software package built with a certain set of clauses, varying compiler version and/or dependency changes. This makes finding constrained packages more easily. There is also a higher level of control when changing dependencies easily.

9

- By setting `build_jobs` to use desired (in this case lesser as Spack by default uses all available cores to build) number of cores.
- `source_cache` allows the user to point Spack towards already downloaded tarballs and stored in other locations.
- `install_tree` specifies the location of where packages are installed.
- The package directory locations sometimes turn out to be very lengthy leading to segmentation faults in scripts used to build. To get around this problem Spack includes the option of specifying `install_path_scheme: '{name}/{version}/{hash:7}'` in the `config.yaml` file. Here, the path is limited to a short and simple seven character hash to distinguish between versions. The directory format can then be shortened thereby maintaining human readability when searching for packages manually.

Application Binary Interfaces provide a cohesive link between various binaries. A good example of this would be a full scale Climate Model which is built of many sub models each built in a different way. Spack ensures each version of a custom build is stored in their own individual DAG hash.

Concretization involves specifying an abstract spec. Such features are lucrative for HPC use cases where an environment with various configurations of a software can be put together.

Concretization of an environment can use packages already built outside the environment but is not limited to just that. If an extra spec is specified inside the environment spack pulls together the needed dependencies and sets the respective build options too. Running `spack spec` reveals the full concretization of a package, basically telling the user how each dependency is built.

In Fig 1.6 `gcc 4.8.5` did not optimise for 6th generation Intel Skylake architecture simply due to the version being out of date. So Spack decided to fallback one generation below to Haswell to ensure the package installed and that the compiler worked.

Spack installs are sand-boxed processes and each dependency install forks into a separate process. Once the dependency builds are successful and all checks completed, the Spack installer goes forward in building the main package.

The user can list out all the details of a package which range from all available versions to configuration options. Before installation the user can use the `info` command to figure out what variation of the install is required.

```

teffy@themachine:~
~$ spack find
=> 87 installed packages
-- linux-arch-haswell / gcc@4.8.5 -----
autoconf@2.69      gdbm@1.18.1      libpciaccess@0.13.5  libxml2@2.9.10  openblas@0.3.9  readline@8.0      zlib@1.2.11
autoconf@2.69      hwloc@1.11.11  libsigsegv@2.12      m4@1.4.18      perl@5.30.1    util-macros@1.19.1
automake@1.16.2    hwloc@1.11.11  libtool@2.4.6        ncurses@6.2     perl@5.30.2    xz@5.2.4
automake@1.16.2    libiconv@1.16  libxml2@2.9.9        numactl@2.0.12  pkgconf@1.6.3  xz@5.2.5

-- linux-arch-skylake / gcc@9.3.0 -----
autoconf@2.69      gdbm@1.18.1      libpciaccess@0.13.5  mpc@1.1.0        numactl@2.0.12  readline@8.0
autoconf@2.69      gmp@6.1.2        libsigsegv@2.12      mpfr@3.1.6        openblas@0.3.9  util-macros@1.19.1
autoconf-archive@2019.01.06 hdf5@1.12.0      libtool@2.4.6        mpfr@3.1.6        openmpi@3.1.5   xz@5.2.4
automake@1.16.2    hwloc@1.11.11  libxml2@2.9.9        ncurses@6.2       perl@5.30.1     zlib@1.2.8
automake@1.16.2    isl@0.20         m4@1.4.18            netcdf-c@4.7.3    perl@5.30.2     zlib@1.2.11
gcc@9.3.0          libiconv@1.16   mpc@1.1.0            netcdf-fortran@4.4.3 pkgconf@1.6.3   zstd@1.4.4

-- linux-arch-skylake / gcc@10.1.0 -----
intel@20.0.1

-- linux-arch-skylake / intel@19.1.1.217 -----
autoconf@2.69      hwloc@1.11.11  libtool@2.4.6      netcdf-c@4.7.3    openmpi@3.1.6  pkgconf@1.7.3      zlib@1.2.11
automake@1.16.2    libiconv@1.16  libxml2@2.9.10     netcdf-fortran@4.5.2 perl@5.30.2     readline@8.0
gdbm@1.18.1        libpciaccess@0.13.5 m4@1.4.18          numactl@2.0.12   perl@5.30.3    util-macros@1.19.1
hdf5@1.10.6        libsigsegv@2.12 ncurses@6.2        openblas@0.3.10  pkgconf@1.6.3  xz@5.2.5

```

Figure 1.6: Example of architecture fallback

With `spack find --paths` Spack lists out the complete paths of all installed software. This makes searching for binaries or libraries very simple. If the user only requires a certain version of a package, `spack find --format "{name}-{version}-{hash}"` lists out name of package with their respective hashes.

```

teffy@themachine:~
~$ spack find --format "{name}-{version}-{hash}"
autoconf-2.69-ar2pptsnt4nvqxq2gqcxrej43eqkoavp
autoconf-2.69-vyb3jxb3ugu7llpkgnxx24rcktiupia
autoconf-2.69-eqkaied2fdk3w6gtq7bpv3ncoc2cepwd
autoconf-2.69-r6bhkhgg423mfxrdqgm55qgrzlyhy
autoconf-2.69-r6ahqocwanr4sdpj5atrth3cvvwbhga
autoconf-archive-2019.01.06-obppocpokcdf65vupzqm6npx2rj3wrs4
automake-1.16.2-zga6dhrckjezffpjrbv2rwsqpdnkojqn
automake-1.16.2-a67efs27i2xmyueysthl2vyftzhi5qq2
automake-1.16.2-kan6f2me2uxmxurf26koo3dtdlahz64k
automake-1.16.2-r3xb5yyvptkyuxh2nvajwlsxytpvnqlz
automake-1.16.2-gqgmdbdt2skunawt37dkil6zc4fotpb7
gcc-9.3.0-clkeenj3hs2a6jhpwldfuqssj3yt3goh
gdbm-1.18.1-2lnrli4og3dvsaudkdh7yul6tekpra5b
gdbm-1.18.1-sf5oqunsye6zmahesmdwnhblised7nu
gdbm-1.18.1-6oy7ofkusbo4unhgtptuwqc5nfmfzrfr
gmp-6.1.2-v4ezeemlv3evorb6sh3f3vvvicbior2
hdf5-1.10.6-wuzqgnqp4xdndxp6reffsxl2lun2norz
hdf5-1.12.0-2vhrsxn3koudrzqbvik4r7hukztofag
hwloc-1.11.11-yxahj6pojpcr5tj4swjalnhwbh7oud
hwloc-1.11.11-34qyrukunb2pcmeo3pbkxhiuruayab5m
hwloc-1.11.11-6o7bq2tecqnne7zxtmokhgsvmwma7aa
hwloc-1.11.11-s6t6t4gwnkaxwm6r7wq7vfmnbzadxdjg
intel-20.0.1-j2zwuebiwdtvoiirhxqb7yjt77ptzww
isl-0.20-bsdnowu2y7n27xvgv7cwcqyzwoglyviu
libiconv-1.16-brxmtknkeznvvd1z3xsfnnt2gb6l4irm
libiconv-1.16-i4g3hnbqgcefgfor6pnqt5in4yfc4xkb
libiconv-1.16-hc4vpzv26fmkfgykdisf55d7tp4dhyzk
libpciaccess-0.13.5-pciqeu4uo2nu25dkicbdut63zo5fxeo
libpciaccess-0.13.5-qzjk3qd5xunnkgc5yjoewf2ehrfmbjiy
libpciaccess-0.13.5-hnhobpxnkapxow5nmez4s7sxyxs6sngxw
libsigsegv-2.12-zpajhpdqtlgxtz7ivhkhwdhfyechikl6
libsigsegv-2.12-1fux3sgv7g7dptehevuxijulyn6xb3xa
libsigsegv-2.12-pwycgdbpuesz5cpeioaxp5kofs3hffvn
libtool-2.4.6-t3b66h3abpw465y5zwhzjkjecmyhir1mc
libtool-2.4.6-3ligehrok2ffqfvnognb3llkp7defhwy
libtool-2.4.6-c4yzfslldltdk2dqfnyrlwqtd6cyhduep
libxml2-2.9.9-ytohu2eno2uxaskm5vg6wazdtdcd2dh6o
libxml2-2.9.9-lyqk7ixq6cbpo5oegfpiqyrltqzdui5z
libxml2-2.9.10-lrff6kz6uldbxvh57gebfbhxewrc4y
libxml2-2.9.10-2bsr2ciqtbagwny66hosawcbizyupung
m4-1.4.18-klskbnh2mp73bjgkwy26ml74hxadp
m4-1.4.18-fxvj6naaya2qkz2zqxqfynvrhrlldgje5
m4-1.4.18-le7xsw6cl6k5n3elgu6h3akdbt3ihkwf
mpc-1.1.0-o7672mik6f5s46kjwhqs6vhiv2lr7hfy
mpc-1.1.0-a3pudtp6pyr476duqlvwmnoq7lyymzf
mpfr-3.1.6-jqm5xripfigx6bbpmdm5syrzr5ehwxo
mpfr-3.1.6-kegwzfonqj35dadzjbayfu5347slvcwq
ncurses-6.2-oj5dbdpnhv62dzcxx55mm5op32qn4wj6
ncurses-6.2-pfbkyzhuujg2hvn4jft6bp6gnficjk7e
ncurses-6.2-etlntsicavllsxbhesaqzqifouc2sskc
netcdf-c-4.7.3-4vgthnpqtc5oi4e2dbou3amcd5hcz3u
netcdf-c-4.7.3-7mg65ej46f2xf7cjfykfbaobafmullk7
netcdf-fortran-4.4.3-5ritdkv3wvpkgekp3qxmvrulau44tzg
netcdf-fortran-4.5.2-3mzjij622xh7vjfixz7qo4xmy4vnz3la3
numactl-2.0.12-yakltyczjiyyaaqj7jvdyseohlkt57le
numactl-2.0.12-krflvmmisoppcq62ivxd2kjzzbjvu5nj
numactl-2.0.12-6mqaxaw5q4u3tygx6ibqo2bwbourdsw3
openblas-0.3.9-zmuwrsdpikwj6qndndmuzccjakklvzg5
openblas-0.3.9-vmqwohu2wknmufvpgoxwm4ermewb2jm7
openblas-0.3.10-hfycupkeohg5d5rbocnv35hoh6li25yh
openmpi-3.1.5-ihrooc5ffkz7audzw6oux4pkqj6hx7m
openmpi-3.1.6-ufsqi3uitoguxj4fhwxwatwhcv5ho3ay
perl-5.30.1-h7s2paqqn7ebumotswztaobrpqh3v5il
perl-5.30.1-ydp5ljw4vx3q4hejzq7fe4dlserhupj
perl-5.30.2-5wdnklzdnf5ckliqvgxa7x7lvmgsxdw3
perl-5.30.2-hfqfzji4wfk3u7xkhwcehdzjg6zkszu
perl-5.30.2-rcdae35f5hvq3ufxkgupxtsaswo3y2oc
perl-5.30.3-oibs6qaiijlrdrrm43cigzmxcgch4wc
pkgconf-1.6.3-gn2duioek2bi3gwyuvs6tjralkg12y74
pkgconf-1.6.3-uzwbl5hmdqkvxm6m6zeyfsgcloiulukz
pkgconf-1.6.3-5kxll6wphr4i2zannkg6tjjukxisnp2b
pkgconf-1.7.3-xm3ts5m75c4hy2amxwaxpvagjkdobv46
readline-8.0-tkxzbq23sb3wp46i2lincdq7bmn6r5oi
readline-8.0-ez6qqjapebqfwnkq3onbadjmnal6e4v
readline-8.0-md2267by533qbc7hogy42kicp4fhqpb
util-macros-1.19.1-zhcej43aonts3yobxqiamaxpkbbp272u
util-macros-1.19.1-lgd55754tdjxsqldqrpoybd7juym7gf4n
util-macros-1.19.1-n3lkm3evnrfcgdcwuk35u6fvcz2t5r5g
xz-5.2.4-cnptk2lq6g6rqtvarq6nrgxg24yjoinz
xz-5.2.4-2hhfh7vzhfons12yirlyqyd4uy3iptir
xz-5.2.5-a7eb37cc3vabqhm5fqxbuyjzkt7bt7ed
xz-5.2.5-25wqyokf6mt54uzfot4dyux6gikt4rl
zlib-1.2.8-7ojyexaztugyiu44cfmotvxxvedb32j6
zlib-1.2.11-6mv2bpskvx4fn23qyfoewlvjyy3czh4
zlib-1.2.11-rjkuc6l5gqihmlvwehds3c6dckxnk3n
zlib-1.2.11-ljoqaswp5fjrm2ot4x52pd7ty2yap3h6ns
zstd-1.4.4-u2b2o5i7lejyrmqubyalji6nazcmbv4
~$

```

Figure 1.7: Package hashes

## 2 Conclusion

Spack is a very flexible package manager that integrates many useful features from many other packaging solutions like Conda, Environment Modules etc. It is constantly gaining more traction and adoption in the scientific computing community. High Performance Computing divisions have a lot to gain when software needs to be reconfigured or built from scratch, say when new hardware upgrades are due. Such a porting process takes a lot of effort when the task is assigned to manual labour. But with Spack's intelligent automation build time is drastically reduced.

The general everyday user is also encouraged to use Spack as its versatile nature is beneficial for compiling and maintaining custom software builds. Porting software environments between colleagues within a project can be done without any hassle too.

# Bibliography

- [1] Apt for debian. [https://www.debian.org/doc/manuals/debian-reference/ch02.en.html#\\_literal\\_apt\\_literal\\_vs\\_literal\\_apt\\_get\\_literal\\_literal\\_apt\\_cache\\_literal\\_vs\\_literal\\_aptitude\\_literal](https://www.debian.org/doc/manuals/debian-reference/ch02.en.html#_literal_apt_literal_vs_literal_apt_get_literal_literal_apt_cache_literal_vs_literal_aptitude_literal).
- [2] Docker container. <https://www.docker.com/resources/what-container>.
- [3] Freebsd ports. <https://www.freebsd.org/ports/master-index.html>.
- [4] Package manager for MacOS. <https://brew.sh/>.
- [5] Pacman for archlinux. <https://www.archlinux.org/>.
- [6] Redhat package manager. <https://rpm.org/about.html>.
- [7] G. Becker, P. Scheibel, M. LeGendre, and a. T. Gamblin. Managing combinatorial software installations with spack. In *2016 Third International Workshop on HPC User Support Tools (HUST)*, pages 14–23, 2016.
- [8] T. Gamblin, M. P. LeGendre, M. R. Collette, G. L. Lee, A. Moody, B. R. de Supinski, and W. S. Futral. The Spack Package Manager: Bringing order to HPC software chaos. In *Supercomputing 2015 (SC'15)*, Austin, Texas, November 15-20 2015.
- [9] M. Geimer, K. Hoste, and R. McLay. Modern scientific software management using easybuild and lmod. In C. Bording and A. Georges, editors, *Proceedings of the First International Workshop on HPC User Support Tools, HUST '14, New Orleans, Louisiana, USA, November 16-21, 2014*, pages 41–51. IEEE Computer Society, 2014.
- [10] R. McLay, K. W. Schulz, W. L. Barth, and T. Minyard. Best practices for the deployment and management of production hpc clusters. In *State of the Practice Reports, SC '11*, New York, NY, USA, 2011. Association for Computing Machinery.

# List of Figures

1.1	Spack Package Dependencies . . . . .	4
1.2	Spack Build Options . . . . .	6
1.3	Spack Input Specs . . . . .	7
1.4	Package Managers . . . . .	8
1.5	Directory tree of installed packages . . . . .	9
1.6	Example of architecture fallback . . . . .	11
1.7	Package hashes . . . . .	12