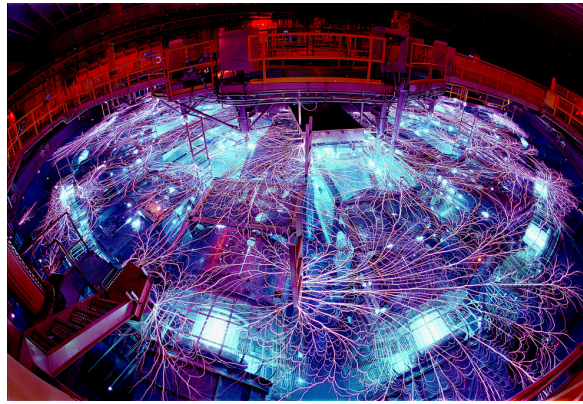
Exceptional service in the national interest

Sandia National Laboratories



The Evolution of Alegra's Devops Ecosystem

Tim Fuller, Steve Bova, and Michael Powell

U.S. DEPARTMENT OF ENERGY

National Nuclear Security Administration

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

boratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, L
Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE

Overview

In 2019, the Alegra team transitioned from a legacy set of tools that handled building, testing, and running `alegranevada` to a new toolset we call `toolset2`. These slides are an overview of `toolset2`.

Alegra

Alegra tooling modernizations

Example usage

Continuous Integration

Conclusion

Sandia National Laboratories

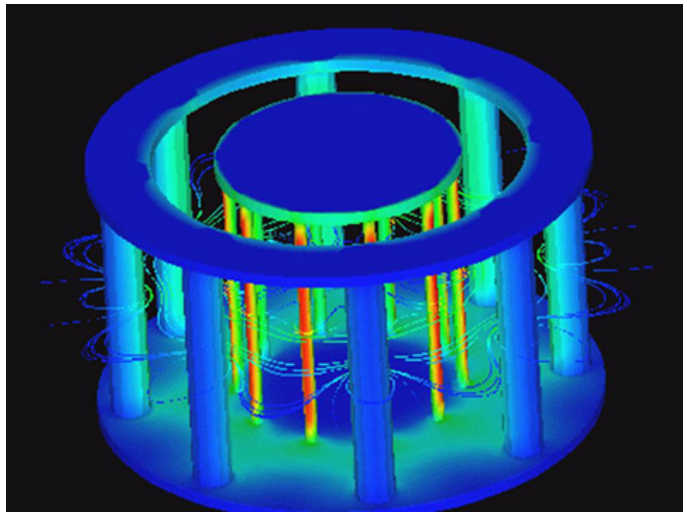
Alegra

Sandia National Laboratories

Summary

Alegra is a roughly 25-year old code that provides approximate solutions to multiphysics problems involving

- large-deformation Lagrangian, Eulerian, or ALE solid dynamics/hydrodynamics;
- electrical conductivity, magnetic induction/diffusion, nonlinear ohmic heating, Lorentz forces;
- finite element discretizations;
- material data and equations of state;
- radiation transport, thermonuclear burn; and
- piezo and ferro electric effects.



Sandia National Laboratories

Challenges

Code base

- 25 year old “legacy code”
- large code base with C++, Fortran, C, and Python components
- extremely complex physics

Dependencies

- complex dependencies: roughly 30 TPLs including Dakota, Trilinos, Xyce
- each having its own build system
- some TPLs have proprietary licenses

Data

- relies on material data from a variety of sources
- ITAR, UCNI, LANL proprietary, and LLNL proprietary data
- not all customers are authorized to receive data

Testing

- most testing done on gifted, and aging, hardware
- thousands of tests with tens of Gb of data
- some tests take longer then 24 hours to execute

Building

- maintaining builds on all SNL CEE-LAN and HPC machines
- maintaining builds on select SNL test beds
- providing builds on customer machines for which there are no SNL counterparts

Running

- complex user interface
- interactions with many other tools: MPI, exodus, etc.

$\Rightarrow$ No dedicated customer support personnel

$\Rightarrow$ No dedicated devops personnel

The legacy toolset

The Alegra "toolset" grew out of the necessity to manage these challenges. The legacy toolset, among other things

- managed and built all TPLs;
- managed and built the `alegranevada` source code;
- managed source code testing;
- managed source code releases;
- defined compiler interfaces and compiler flags;
- provided scripts and tools for interacting with `alegranevada`; and
- provided user interfaces to `alegranevada` executables.

The legacy toolset began as a collection of csh scripts and has evolved in to a mixed-language set of tools mostly written in Python and csh.

The legacy toolset: addressing challenges

Code base: single SVN repository for code and data

- ⇒ code/data kept in consistent states
- ⇒ difficult to manage access controls
- ⇒ no (easy to use) pull/merge request mechanism

Dependencies: "vendor" TPLs in to code repository, patch if necessary, write custom build scripts for each

- ⇒ consistent builds
- ⇒ must do double duty as a package manager and build system
- ⇒ TPLs have drifted from upstream versions and are difficult to update

Data: store all data in centralized location

- ⇒ easy to find/navigate
- ⇒ difficult access controls
- ⇒ must be filtered for releases to remove sensitive data

Testing: scripts to run nightly and weekly tests

- ⇒ code is kept safe from regressions
- ⇒ scripts duplicate existing tools (cron, CDash, etc)
- ⇒ test invocation does not match user invocation
- ⇒ no automated commit testing

Building: use custom build system

- ⇒ consistent builds that we control from end to end
- ⇒ duplicates specialized tools Spack, CMake, etc.
- ⇒ requires considerable expertise to maintain compiler files, MPI files, etc.

Running: provide scripts for interacting with the code

- ⇒ Alegra (usually) invoked in a consistent way
- ⇒ scripts are not terribly consistent/integrated
- ⇒ Many of the Python scripts wrap older csh scripts
- ⇒ Python scripts written in Python2

The legacy toolset grew out of necessity, becoming very robust over the years. Still, it requires a lot of tribal knowledge to understand all the parts and some design decisions have been made that are not aging well.

Sandia National Laboratories

Alegra tooling modernizations

Alegra tooling modernizations

Requirements

Reduce technical debt: use outside tools to do what they do well

- ⇒ Spack for dependency management
- ⇒ CMake for `alegranevada` build system
- ⇒ vvtest with “scripting” interface for testing
- ⇒ git for version control

Alegra tooling modernizations

Requirements

Reduce technical debt: use outside tools to do what they do well

- ⇒ Spack for dependency management
- ⇒ CMake for `alegranevada` build system
- ⇒ vvtest with “scripting” interface for testing
- ⇒ git for version control

Protect UCNI, ITAR, and other proprietary data

- ⇒ separate git repositories with access controls for sensitive components
- ⇒ separate components brought in through git submodules
- ⇒ “opt in” instead of “opt out”

Alegra tooling modernizations

Requirements

Reduce technical debt: use outside tools to do what they do well

- ⇒ Spack for dependency management
- ⇒ CMake for `alegranevada` build system
- ⇒ vvtest with “scripting” interface for testing
- ⇒ git for version control

Protect UCNI, ITAR, and other proprietary data

- ⇒ separate git repositories with access controls for sensitive components
- ⇒ separate components brought in through git submodules
- ⇒ “opt in” instead of “opt out”

Be as future “proof” as possible

- ⇒ Python3.6+
- ⇒ developer and user documentation
- ⇒ unit testing, code formatters, static analyzers used before code checkins

Alegra tooling modernizations

Requirements

Reduce technical debt: use outside tools to do what they do well

- ⇒ Spack for dependency management
- ⇒ CMake for `alegranevada` build system
- ⇒ vvtest with “scripting” interface for testing
- ⇒ git for version control

Protect UCNI, ITAR, and other proprietary data

- ⇒ separate git repositories with access controls for sensitive components
- ⇒ separate components brought in through git submodules
- ⇒ “opt in” instead of “opt out”

Be as future “proof” as possible

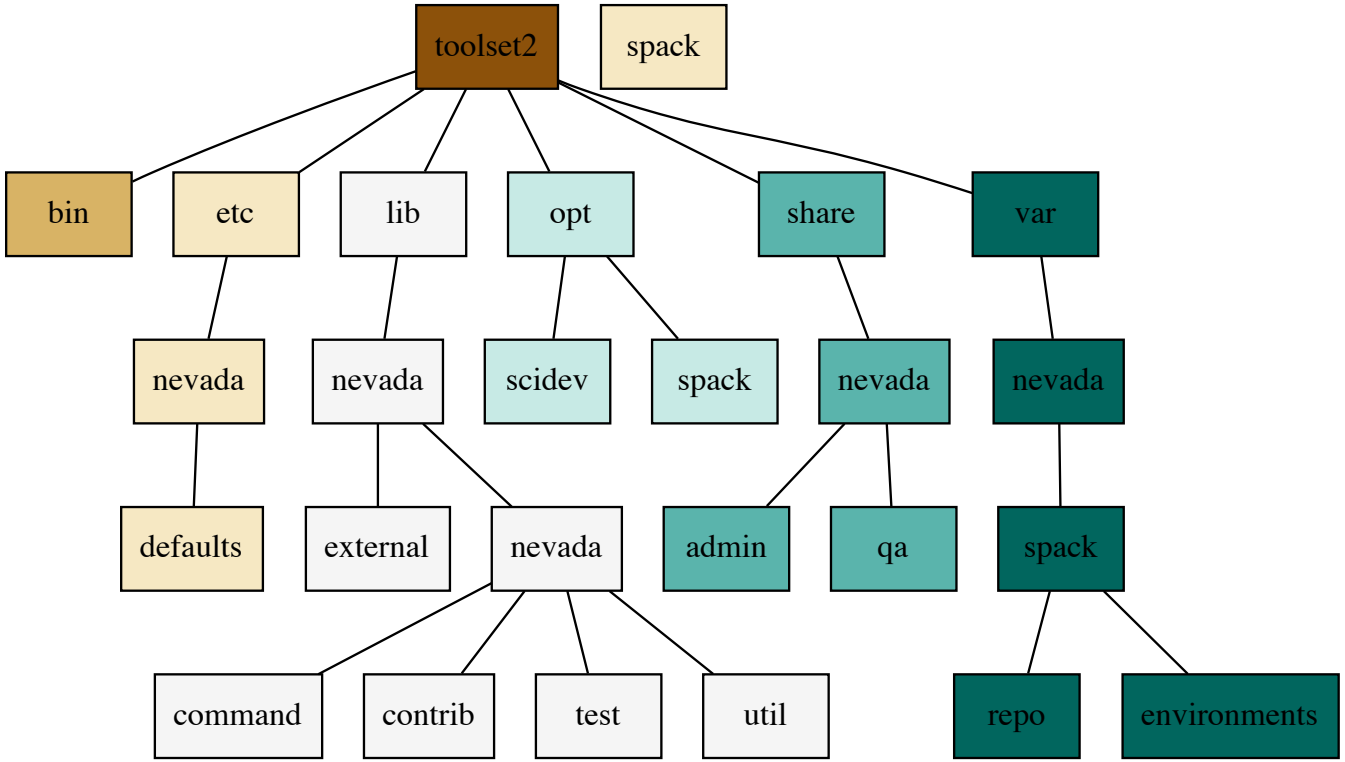
- ⇒ Python3.6+
- ⇒ developer and user documentation
- ⇒ unit testing, code formatters, static analyzers used before code checkins

Consistent and integrated design

- ⇒ developers, users, tests use same interfaces
- ⇒ implement capabilities as library functions with command line, user, and test interfaces, etc.
- ⇒ consistent code formatting (`black`), static analyzers (`flake8`)

Alegra tooling modernizations

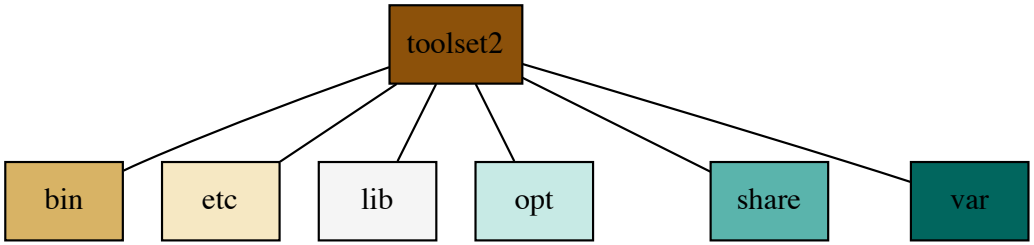
toolset2



- layout borrows from Spack and the Linux filesystem hierarchy standard (FHS)
- toolset2 code written entirely in Python 3.6+
- extensive (and growing) documentation

Alegra tooling modernizations

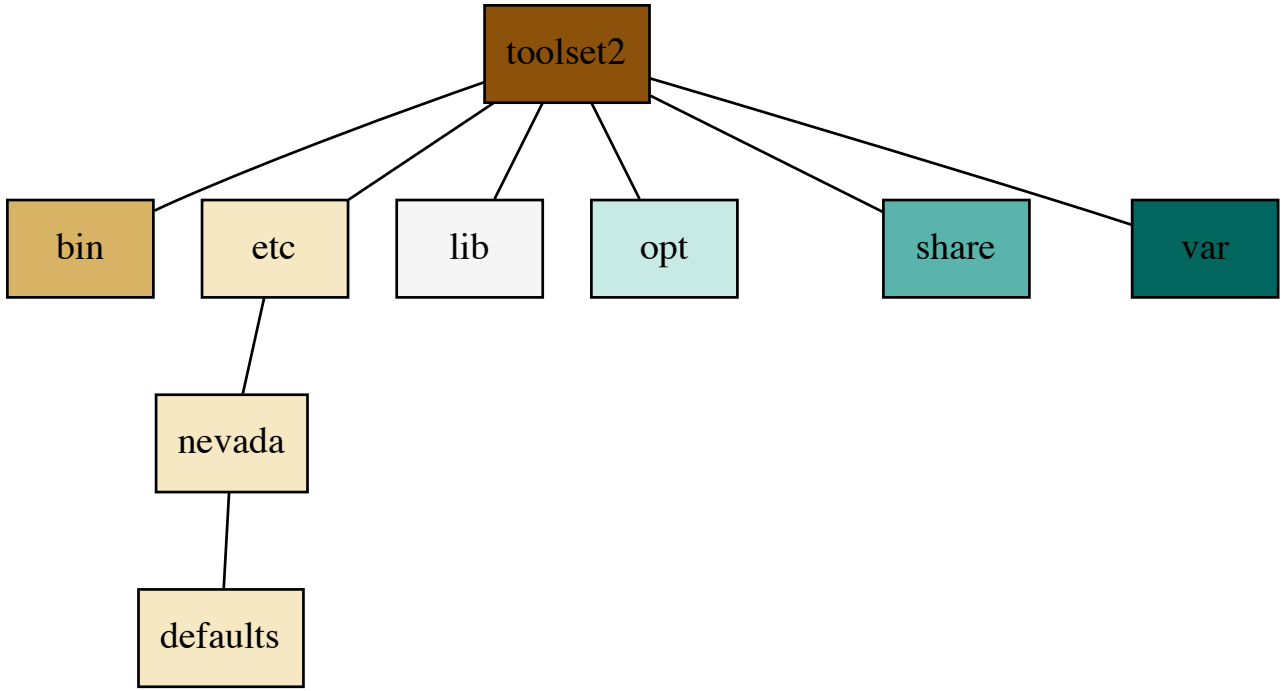
toolset2



- `bin` directory contains executable scripts
- the `bin/nevada` script is the main entry point to toolset2
- `nevada` has many subcommands that run, build, and interact with Alegra (and friends)

Alegra tooling modernizations

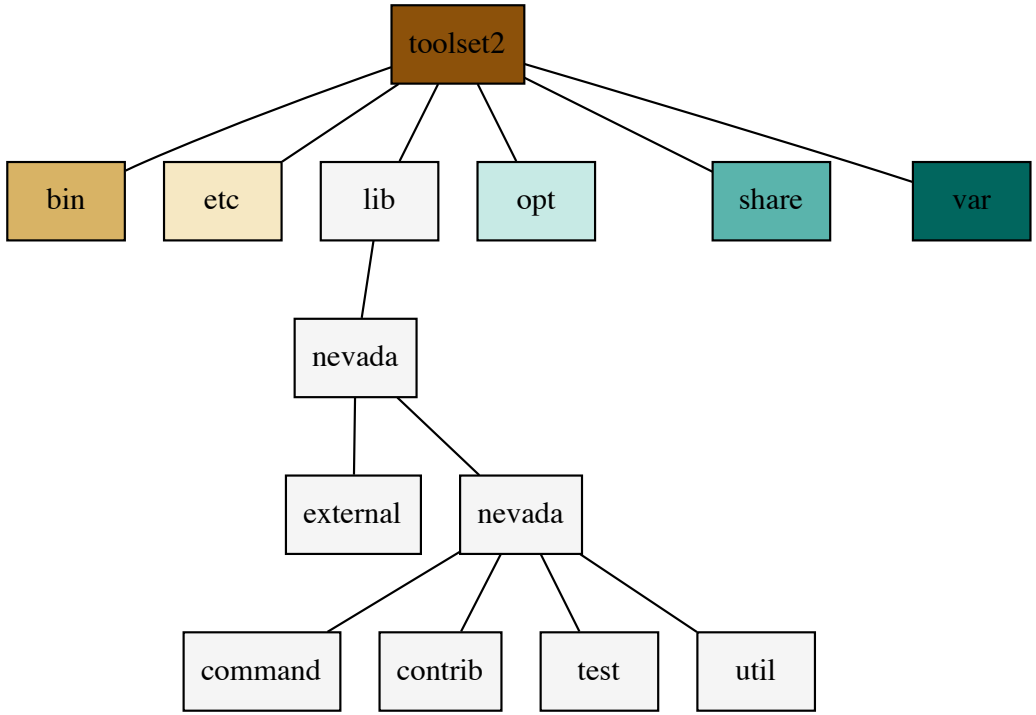
toolset2



- the `etc/nevada/defaults` contains default configurations for toolset2

Alegra tooling modernizations

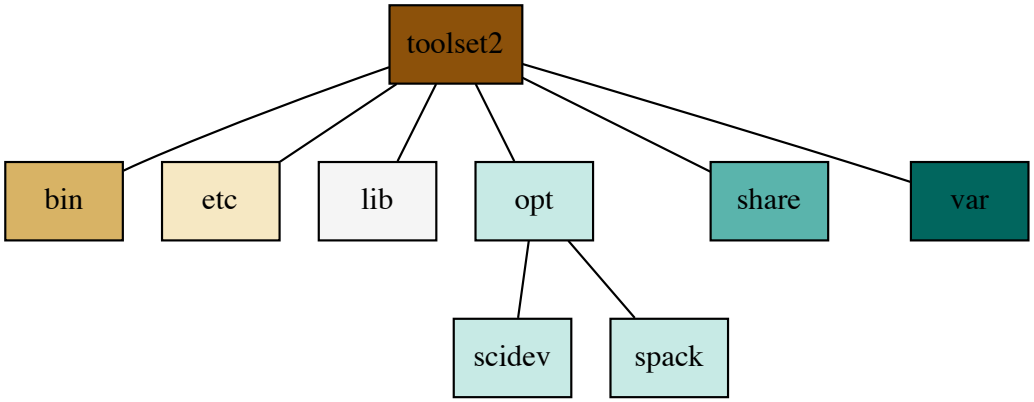
toolset2



- the `lib` directory contains most python library code
- `lib/nevada/external` are vendored libraries
- `lib/nevada/nevada/command` has the implementations of the subcommands called by `nevada`
- `lib/nevada/nevada/contrib` are library functions that run and control Alegra and friends
- `lib/nevada/nevada/test` are internal toolset2 tests
- `lib/nevada/nevada/util` are general utilities

Alegra tooling modernizations

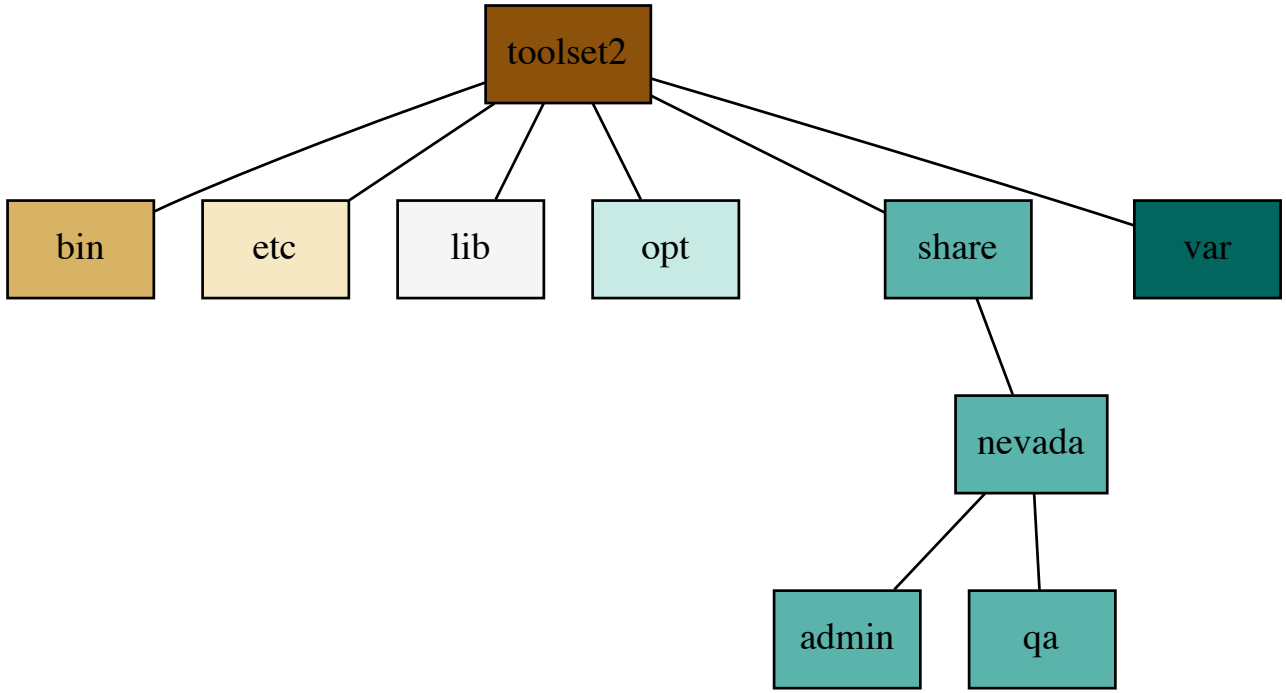
toolset2



- the `opt` directory contains necessary external software that is actively developed
- Spack for build management
- Scidev for Alegra integration testing
- Spack and Scidev are included in toolset2 as submodules

Alegra tooling modernizations

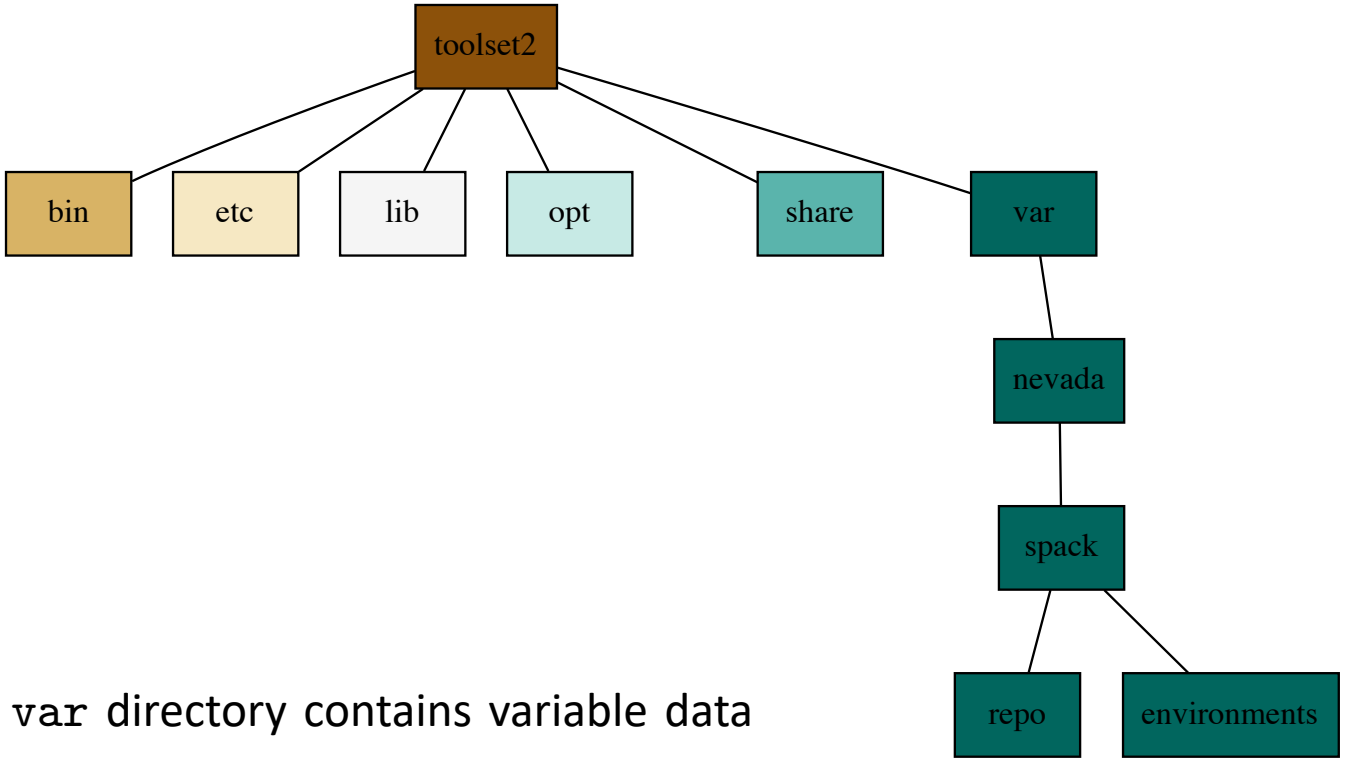
toolset2



- the `share` directory contains other executables and data
- `share/nevada/qa` contains toolset2 QA scripts
- `share/nevada/admin` contains code used for administrative purposes

Alegra tooling modernizations

toolset2



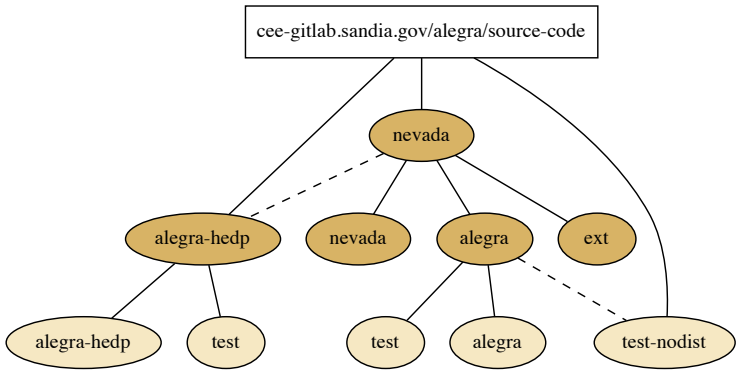
- the `var` directory contains variable data
- `var/nevada/spack/environments` contains pre-configured Spack environments for supported machines
- `var/nevada/spack/repo` contains the “spackages” for Alegra and its TPLs
- Even though we don’t want to build and manage the TPLs ourselves, we want to control build options (through the spackage)

Alegra tooling modernizations

git version control

Transitioned from svn version control to git
Latest version of alegranevada copied from `teamforge.sandia.gov` and distributed as shown in `cee-gitlab.sandia.gov/alegra/source-code`

- Alegra-HEDP is a git submodule of alegranevada
- sensitive HEDP data maintained with strict access controls
- tests distributed with source



Sandia National Laboratories

Spack integration

Spack chosen for dependency and build management

What is spack?

- a package manager under active development at LLNL
- think `rpm, brew, port` etc
- designed from the ground up for building software in a scientific computing environment

Why spack?

- actively developed and funded
- Spack team has expertise in compilers/mpi/architectures etc.
- Spack works closely with DOE to target software for upcoming machines
- enables easier transition from our copies of TPLs to their externally developed and supported versions

Alegra tooling modernizations

Sandia National Laboratories

Spack integration

Spack is included as a git submodule and wrapped with the `nevada` script

- including as a submodule guarantees our users/developers are using the right version of Spack
- wrapping Spack allows us to
  - isolate toolset2's version of Spack from the user's
  - provide non-default Spack settings in a transparent way

Example: install `alegra` (and its dependencies) on a ceelan machine

```
$ nevada config set config:spack_env:cee-rhel7-gcc4.9.2-openmpi1.8.3
$ nevada spack install alegranevada@master ~alegra-hedp
```

And on macOS

```
$ nevada config set config:spack_env:darwin-gcc8.3.0-openmpi3.1.3
$ nevada spack install alegranevada@master ~alegra-hedp
```

Alegra tooling modernizations

TPL management

- Alegra group still maintains many TPLs
- each TPL was moved from the original svn repository to its own git repository
- TPLs having sensitive data were further separated to control access, for example, the Lambda TPL
  - $\Rightarrow$ was separated into source, SNL proprietary data, LANL proprietary data, each with its own repository
- "spackage" for each TPL written that allows Spack to build the TPL to our specs
- when TPL is upgraded, its spackage will be changed to point to the native upstream source, so that we transition away from maintaining our own copies
  - $\Rightarrow$ Trilinos, DiomSpy, Boost, netCDF, hdf5, SEACAS, and Dakota are all now fetched from their host repositories

Example: install TPL Dakota on the ceelan

```
$ nevada config set config:spack_env:cee-rhel7-gcc4.9.2-openmpi1.8.3
$ nevada spack install dakota
```

Alegra tooling modernizations

Sandia National Laboratories

Build system

Build system transitioned from homegrown xml+Makefile system to CMake

- CMake is an industry standard
- leverage CMake's expertise in building to specific targets/platforms
- requires CMake > 3.13

Sandia National Laboratories

Example usage

Example usage

Developer workflow

```
$ git clone --recursive git@cee-gitlab.sandia.gov:alegra/source-code/alegra
$ nevada spack develop -p `pwd`/alegranevada alegranevada@master
$ nevada spack install alegranevada@master
```

- `nevada spack develop` marks the package as in development
- Invocations of `nevada spack install` for development packages build the local source

Example usage

Developer workflow: integration testing

```
$ nevada config set config:build_config:BUILD_CONFIG_PATH
$ nevada vvtest [options] +builtin
$ # check for broken tests
```

A simple test file

```
#VVT: keywords : fast 2D
#VVT: parameterize (autotype) : np = 1 4
import vvtest_util as vvt
from seacas import exo_diff
from nevada.contrib import alegra


def test():
    alegra(vvt.NAME, dimension=3, nproc=vvt.np, preprocess="aprepro")
    exo_diff(f"{vvt.NAME}.base_exo", f"{vvt.NAME}.exo", f"{vvt.NAME}.exodiff")


if __name__ == "__main__":
    import sys

    sys.exit(test())
```

Example usage

Developer workflow: committing code

```
$ nevada vvtest +builtin
$ # check for broken tests
$ git add ...
$ git commit -m ...
$ git push origin <branch name>
```

- Developers run the “builtin” integration tests
- Developers are responsible for making sure to run the tests and that they pass
- Developers push to feature branch and open merge request
- Merge requests must pass pre-defined GitLab pipelines before merging

Example usage

Analyst workflow

```
$ ls
runid.inp runid.py
```

`runid.inp` is the user's normal Alegra input file and can be run in the normal way:

```
$ nevada run-alegra --preprocess=aprepro --nproc=4 runid
```

Under the hood, `nevada run-alegra` processes the command line arguments and calls `nevada.contrib.alegra`.

Example usage

Analyst workflow

Alternatively, the user can run a Python script, such as the following

```
from nevada.contrib import alegra
def main():
    alegra("runid", nproc=4, preprocess="aprepro")

if __name__ == "__main__":
    main()
```

Use the `nevada python` command to execute the script:

```
$ nevada python runid.py
```

An advantage of this method over running the input file directly is that the python script can contain any other pre and post processing steps, in a single location.

Example usage

Analyst workflow

Alternatively, the user can run a Python script, such as the following

```
from nevada.contrib import alegra
def main():
    alegra("runid", nproc=4, preprocess="aprepro")

if __name__ == "__main__":
    main()
```

Use the `nevada python` command to execute the script:

```
$ nevada python runid.py
```

An advantage of this method over running the input file directly is that the python script can contain any other pre and post processing steps, in a single location.

There is only one way of calling Alegra - and it is exercised in the test, cli, and script interface identically!

Sandia National Laboratories

Continuous Integration

Sandia National Laboratories

Developer Workflow

SVN Commit Policy

- Developers responsible for running tests
- Developers solicited code review
- Developers have unlimited commit privileges to SVN repository

⇒ Enforced by verbal honor system

GIT Merge Request Policy

- Developer opens merge request
- Merge request cannot be merged until
  - Merge request is approved by another developer
  - GitLab pipelines pass
- Developers do not have push access to master branch