ORNL/TM-2021/1979

VERA Installation Guide

Mark Baird
Roscoe A. Bartlett
Ron Lee
Brenden T. Mervin
Benjamin S. Collins
Robert A. Lefebvre

April 2021

OAK RIDGE National Laboratory

ORNL IS MANAGED BY UT-BATTELLE LLC FOR THE US DEPARTMENT OF ENERGY

ORNL/TM-2021/1979

Nuclear Energy and Fuel Cycle Division

VERA INSTALLATION GUIDE

Mark Baird
Roscoe A. Bartlett
Ron Lee
Brenden T. Mervin
Benjamin S. Collins
Robert A. Lefebvre

April 2021

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, TN 37831-6283
managed by
UT-Battelle LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

CONTENTS

ABBREVIATIONS

CASL	Consortium for Advanced Simulation of Light Water Reactors
GCC	GNU Compiler Collection
HPC	high-performance computing
ORNL	Oak Ridge National Laboratory
SSH	Secure Shell Protocol
TPLs	third-party libraries
triBITS	tribal Build, Integrate, and Test System
VUG	VERA Users Group

1 INTRODUCTION

This guide describes the structure and setup of the standard Virtual Environment for Reactor Applications(VERA) development environment (VERA Dev Env) and standard VERA third-party libraries (TPLs) that need to be in place before many of the VERA simulation components are installed. It describes everything from the initial setup on a new machine to the final build, testing, and installation of VERA components. The goal of this document is to describe how to create the directories and contents outlined in Section 2, obtain the remaining VERA source, and build, test, and install any of the necessary VERA components on a given system. This document describes the process for a development version of VERA and for a released tarball of the VERA sources.

Section 2 outlines the directory structure. Section 3 describes the the major steps of the installation process. Section 3 contains all of the information needed to perform the full install of the VERA Dev Env, as well as the install of VERA components themselves. The remaining sections contain more information and details for variations, as well as tips for how to solve problems when things go wrong.

WARNING: This guide describes installation of the VERA Dev Env and TPLs; it does not contain specific information about specific VERA simulation components. That information is found in other sources. Please consult with a VERA Users Group (VUG) representative about what VERA components are available to install from source and what capabilities the components provide. In this manual, information about other VERA repositories is only used for examples and may not be up to date.

2 STANDARD VERA DEV ENV DIRECTORY STRUCTURE

The standard directory structure for the installation of the VERA Dev Env is given below:

```
$VERA_DEV_ENV_BASE/
  gcc-5.4.0/
    load_dev_env.sh
    toolset/
      gcc-5.4.0/
      mpich-3.2.1/
    common_tools/
      autoconf-2.69/
      cmake-3.11.2/
      gitdist
    tpls/
      opt/
        lapack-3.3.1/
        boost-1.55.0/
        zlib-1.2.7/
        hdf5-1.10.2/
        petsc-3.6.4/
        silo-4.10.2/
        qt-4.8.7/
        ...
```

```
opt_static/
 ...
dbg/
  ...
dbg_static/
  ...
...
```

For the example in this guide,

```
VERA_DEV_ENV_BASE=/projects/vera
```

was set, but note that any base directory can be used. This directory is where the prerequisite TPLs (see A.1) and VERA tools that are used for configuring, building, testing, and installing VERA are deployed.

For the standard GNU Compiler Collection (GCC) 5.4.0 VERA Dev Env,

```
VERA_DEV_ENV_COMPILER_BASE=$VERA_DEV_ENV_BASE/gcc-5.4.0
```

was set.

For a given compiler set, TPLs can be installed for different configurations, such as debug (dbg), optimized (opt), or other variations (e.g., dbg-checkedstl). The standard TPL install is used in this guide and is provided below:

```
VERA_TPL_INSTALL_DIR=$VERA_DEV_ENV_COMPILER_BASE/tpls/opt
```

For the install of static TPLs, use:

```
VERA_TPL_INSTALL_DIR=$VERA_DEV_ENV_COMPILER_BASE/tpls/opt_static
```

Shared libraries should be the default (see A.4).

All of the VERA Dev Env, TPL install tools, and other directory environments use the variables and typical values shown in Table 1 to determine a particular installation of the VERA Dev Env to use for building/testing/installing VERA components.

Table 1. VERA directory environment variables

Variable	Common/Example Value
VERA_DEV_ENV_BASE	/projecrts/vera
VERA_DEV_ENV_COMPILER_BASE	$VERA_DEV_ENV_BASE/gcc-5.4.0
VERA_TPL_INSTALL_DIR	$VERA_DEV_ENV_COMPILER_BASE/tpls/opt
VERA_BASE_DIR	$HOME/VERA.base
VERA_SCRATCH_DIR	$VERA_BASE_DIR/scratch
VERA_DIR	$VERA_DEV_ENV_BASE/vera-X.Y.Z-Source
VERA_BUILD_DIR	$HOME/VERA_BUILD
VERA_INSTALL_DIR	/projects/vera/vera_installs/`date +%Y-%m-%d`

WARNING: When using shared libraries, be careful to avoid the error **"RegularExpression::compile(): Expression too big."** when using RPATH and when RPATH gets stripped out on install. To avoid this error, use the shortest build directory possible, such as the following:

```
export VERA_BUILD_DIR=$HOME/VERA_BUILD
```

3 INSTALLATION PROCESS

This section gives the set of commands that must be run to install the VERA Dev Env and perform a test build of VERA. These instructions assume that the entire VERA Dev Env will be installed, starting with GCC, MPI, CMake, and TPLs. Customization can be done as needed, but it is not covered here. All the sources and scripts are provided as downloads from the internet or from networked computers, or they are provided as compressed files (e.g., on compact discs for those who cannot access outside machines from the install machines). Links for additional details are given for each step below, or they are available in the install tools themselves.

The steps are as follows:

1. Make sure the basic prerequisites are satisfied (Section 3.1).
2. Determine source, scratch, and install directories (Section 3.2).
3. Get the base VERA and Tribal Build, Integrate, and Test System (TriBITS) source directories (Section 3.3).
4. Install the base development environment (Section 3.4)
5. Install Anaconda2 (Section 3.5).
6. Install the VERA TPLs (Section 3.6).
7. Build/install VERA Components (Section 3.7).
8. Complete setup of the installed VERA Dev Env, and perform final cleanup (Section 3.8).

3.1 MAKE SURE THE BASIC PREREQUISITES ARE SATISFIED

Before the VERA development/install environment and VERA itself can be installed, the following steps should be performed:

1. Create a `vera-admin` Unix user, a `vera-users` Unix group, and the required base directories for building, testing, installing, and maintaining VERA (see Sections 4.5 and 4.6).
2. Make sure the system can handle an installation of VERA (see Sections 4.2 and 4.3).
3. If the source is from the VERA Git repositories:
    a. Set up Secure Shell Protocol (SSH) access to the Oak Ridge National Laboratory (ORNL) GitLab website `code-int.ornl.gov` (`casl-dev` for short, see Section 4.4 and A.1).
    b. Get approval to access the required protected VERA git repositories (see Sections 4.1 and 6.2).

3.2 DETERMINE SOURCE, SCRATCH, AND INSTALL DIRECTORIES

Set environment variables for the location of the installed/shared VERA development/install environment `VERA_DEV_ENV_BASE` and the base location of the VERA sources `VERA_BASE_DIR` to drive the INSTALL and other specified in Table 1. For example:

```
# Set the base directory  (you can pick any paths you want)
export VERA_DEV_ENV_BASE=/projects/vera
export VERA_BASE_DIR=${VERA_DEV_ENV_BASE}/source
export VERA_SCRATCH_DIR=${VERA_BASE_DIR}/scratch
export VERA_TPL_INSTALL_DIR=${VERA_DEV_ENV_BASE}/gcc-5.4.0/tpls/opt
export VERA_BUILD_DIR=$HOME/VERA_BUILD
export VERA_INSTALL_DIR=${VERA_DEV_ENV_BASE}/installs/`date +%Y-%m-%d`

# Create some base directories (if they do not already exist)
mkdir -p ${VERA_BASE_DIR}
mkdir -p ${VERA_SCRATCH_DIR}
```

All the other directories will be created as described below, or they will be created automatically by running various install tools.

WARNING: When using shared libraries, be careful to avoid the error **"RegularExpression::compile(): Expression too big."** when using RPATH and when RPATH gets stripped out on install. To avoid this error, use the shortest possible build directory, such as:

```
export VERA_BUILD_DIR=$HOME/VERA_BUILD
```

3.3 GET THE BASE VERA AND TRIBITS SOURCE DIRECTORIES

If using a VERA tarball, enter:

```
cd ${VERA_BASE_DIR}/
tar -zxvf vera-X.Y.Z-Source.tar.gz
tar -zxvf VERAData-X.Y.tar.gz
```

then set:

```
export VERA_DIR=${VERA_BASE_DIR}/vera-X.Y.Z-Source
export VERAData_DIR=${VERA_BASE_DIR}/vera-X.Y.Z-Source\VERAData
export CE_DATA_DIR=${VERA_BASE_DIR}/vera-X.Y.Z-Source\VERAData\CEData
export SCALE_DATA=${VERA_BASE_DIR}/vera-X.Y.Z-Source\VERAData\SCALE
export MPACT_DATA=${VERA_BASE_DIR}/vera-X.Y.Z-Source\VERAData\MPACT
```

If cloning the sources from `code-int:VERA` (if they are not already cloned), first set up the SSH tunnel (see A.1) to `code-int` with:

```
ssh -fN tunnelinit
```

Then clone with the following commands:

```
cd ${VERA_BASE_DIR}/
git clone git@code-int:VERA/VERA
cd VERA/
git clone git@code-int:VERA/TriBITS
```

Then set:

```
export VERA_DIR=${VERA_BASE_DIR}/VERA
```

3.4 INSTALL THE BASE DEVELOPMENT ENVIRONMENT

To install all the generic tools (excluding the TPLs), run the following commands:

```
cd ${VERA_SCRATCH_DIR}/
cp ${VERA_DIR}/doc/omnus.sh ./
vi omnus.sh #set the specific base path you would like
./omnus.sh 2>&1 |tee devenv_install.log
```

If internet access to download source libraries is unavailable, then use the `omnus_local.sh` script and the env source files (`vera-4.2_env.tar.gz`), which can be provided by request. To request the `vera-4.2_env.tar.gz` file, email casl-vri-infrastructure@casl.gov. After the file is received, run the following command(s):

```
cd ${VERA_SCRATCH_DIR}/
cp ${VERA-DIR}/doc/omnus_local.sh ./
vi omnus_local.sh #set the specific base path you would like
./omnus_local.sh 2>&1 |tee devenv_install.log
```

TIPS:

- If any error occurs, check the log file (`devenv_install.log`) to see what failed, and then check the log file it indicates to see the actual errors.
- If the tool chain is being built with GCC 5.X and an error is encountered when compiling the VERA version of GCC that contains

  ```
  ...version `GLIBCXX_3.4.20` not found...,
  ```

  then the C++ standard library must be preloaded using the following command:

  ```
  export LD_PRELOAD=/usr/lib/libstdc++.so.6
  ```

  Note that the path to the library may be different for your system, depending on the base install path systems. Also note that this environment variable must be added to the development and runtime environments for VERA (i.e., the `${VERA_DEV_ENV_COMPILER_BASE}/load_dev_env.sh` and the `${VERA_INSTALL_DIR}/load_env.sh` scripts).

Once the `omnus` script completes, it should install GCC 5.4.0, MPICH 3.2.1, CMake 3.14.3, `gitdist`, and `load_dev_env.sh`, as shown in Section 2.

After a successful install, source the installed `load_dev_env.sh` script as:

```
source ${VERA_DEV_ENV_BASE}/gcc-5.4.0/load_dev_env.sh
```

Then, pre-pend PATH to find the installed programs (cmake, gitdist, gcc, mpicc, ...) in the locations shown in Section 2. Make sure to source the new development environment, with, for example

```
which cmake
which gcc
which mpicc
```

This should return

```
.../gcc-5.4.0/common_tools/cmake-3.14.3/bin/cmake
.../gcc-5.4.0/toolset/gcc-5.4.0/bin/gcc
.../gcc-5.4.0/toolset/mpich-3.2.1/bin/mpicc
```

3.5 INSTALL ANACONDA2

The installation of Anaconda2 is required. It provides an updated Python version, as well as numpy and other Python modules used by several VERA packages. Also, if the user plans to utilize the VERArun tool on high-performance computing (HPC) systems, then the additional packages are required for a successful setup.

To install Anaconda2, go to the site

- https://docs.anaconda.com/anaconda/install/

and follow the instructions to download the binary installation script for the version based on Python 2.7.13. For 64-bit Linux systems, use the following script:

- https://repo.anaconda.com/archive/Anaconda2-5.3.1-Linux-x86_64.sh

The URL location may change.

Once this binary install script is downloaded (presumably to ~/Download), then run it as

```
chmod u+x ~/Downloads/Anaconda2-5.3.1-Linux-x86_64.sh
~/Downloads/Anaconda2-5.3.1-Linux-x86_64.sh
```

When the interactive install script asks for an install location, enter the following: $VERA_DEV_ENV_BASE/gcc-5.4.0/common_tools/anaconda2 Note that $VERA_DEV_ENV_BASE/ must be expanded manually. Users can choose to allow the script to edit their ~/.bashrc file or not to add this to their PATH. However, later in this document, it is assumed that that step is not done. Users' load_dev_env file should address this.

To use the installed Anaconda2 Python executable, related modules, and tools, update your dev env path to the following:

```
export PATH=$VERA_DEV_ENV_BASE/gcc-5.4.0/common_tools/anaconda2/bin:$PATH
```

Verify that the correct version of Python is being used by running

```
which python
python --version
```

which returns

```
.../gcc-5.4.0/common_tools/anaconda2/bin/python
Python 2.7.13 :: Anaconda 4.3.1 (64-bit)
```

Before doing any configures, builds, or tests with VERA, this path must be prepended (as described earlier), or the `load_dev_env` file will need to be sourced.

3.6 INSTALL THE VERA TPLs

First, obtain the `vera_tpls` source repository and the matching version of the TPLs:

```
cd ${VERA_BASE_DIR}/
git clone https://code.ornl.gov/VERA/vera_tpls
cd vera_tpls/
git checkout vera-tpls-4.2
```

If users do not have access to GitHub, they can obtain the tarball `vera_tpls-4.2.tar.gz` and untar/unzip it in the current directory using

```
cd ${VERA_BASE_DIR}/
tar -xzf ~/vera_tpls-4.2.tar.gz
```

Once the directory `vera_tpls` for the correct version of the VERA TPLs is obtained (using one of the methods mentioned earlier), the TPLs can be configured and built with

```
cd ${VERA_SCRATCH_DIR}/
${VERA_BASE_DIR}/vera_tpls/TPL_build/install_tpls.sh -DPROCS_INSTALL=8 \
  &> install_tpls.out
```

NOTE: Adjust `-DPROCS_INSTALL=8` to the appropriate number of processes for the current machine, but please be respectful of fellow system users.

By default, this installs a shared library version of the TPLs (see Section A.4). A static library version can be installed by entering `-DENABLE_SHARED=OFF` and passing in

```
-DCMAKE_INSTALL_PREFIX=$VERA_DEV_ENV_COMPILER_BASE/tpls/opt_static
```

For more details, see Section 5.

3.7 BUILD/INSTALL VERA COMPONENTS

Finally, the installed VERA Dev Env must be tested by configuring, building, testing, and installing VERA from source (installation not required).

If the VERA source were obtained from a source tarball file (see earlier in this document), then all of the sources needed to build the VERA components should be in place (pointed to by `${VERA_DIR}`).

If the VERA source were obtained from the Git repositories on `code-int`, then the remaining VERA Git repositories must be cloned. First, if an SSH tunnel is necessary, initialize the tunnel (see Section A.1) with

```
ssh -fN tunnelinit
```

Then the remaining repositories can be cloned using:

```
cd ${VERA_DIR}/
./clone_vera_repos.py
```

Once the source is obtained, users can configure, build, test, and install with

```
# Set up the build dir
mkdir -p ${VERA_BUILD_DIR}
cd ${VERA_BUILD_DIR}/
ln -s $VERA_DIR/cmake/std/gcc-5.4.0/do-configure.MPI_RELEASE_SHARED \
  do-configure

# Configure, build, and test
./do-configure \
   -D CMAKE_INSTALL_PREFIX=$VERA_INSTALL_DIR \
   -DVERA_ENABLE_DEVELOPMENT_MODE=OFF \
   -D VERA_ENABLE_ALL_PACKAGES=ON &> configure.out
make -j8 &> make.out
ctest -j8 &> ctest.out
```

NOTE: The above input shows the usage of `8` processes to build the code and run the tests. Change `8` to the appropriate value for the machine being used.

If all of the tests pass and the output from `ctest` looks like the following

```
100% tests passed, 0 tests failed out of 697

Label Time Summary:
COBRA_TF           =  70.01 sec
CTeuchos           =   0.40 sec
DataTransferKit    =  16.06 sec
ForTeuchos         =   0.73 sec
Insilico           = 281.07 sec
MPACT_Drivers      = 303.89 sec
MPACT_libs         =  86.12 sec
TriBITS            = 469.13 sec
VERAIn             = 199.05 sec

Total Test time (real) = 690.22 sec
```

then all is well. The number of tests may vary based on enables and test updates.

VERA can be installed for use by others with

```
make -j8 install &> make.install.out
```

Permissions on the installed version of VERA should be set using

```
chmod chgrp -R vera-users $VERA_INSTALL_DIR
```

(see Section 4.5.)

After the install, see the instructions using VERA in the readme file:

```
$VERA_INSTALL_DIR/README
```

In particular, users should source the following script to set up their environment (i.e., `PATH` and other variables) to run the installed executables and scripts:

```
$VERA_INSTALL_DIR/load_env.sh
```

See the online copy README.VERA (becomes the installed file `README`).

NOTE: The commands shown above are used to build and install a shared library version of VERA (see Section A.4). A static library version can be installed by replacing `MPI_RELEASE_SHARED` with `MPI_RELEASE_STATIC` above. Users may want to use this approach if they are installing static TPLs using `-DENABLE_SHARED=OFF` in Section 3.6.

3.8 FINAL SETUP OF INSTALLED VERA DEV ENV AND FINAL CLEANUP

After the basic tools and TPLs of the VERA Dev Env (basic tools and TPLs) have been installed, the directories must be opened to provide other users with access to the installed tools and libraries. Since there is no export-controlled information or other sensitive data contained in the installed VERA Dev Env, it can be made public by entering

```
chmod -R a+rX ${VERA_DEV_ENV_BASE}
```

It is important to note that the SOURCE or BUILD directories should not be a sub directory of `${VERA_SCRATCH_DIR}`, or else unlicensed users may have access. Extreme caution should be followed with the SOURCE and BUILD directory access.

Once users have finished installing and testing the VERA Dev Env, the generated texttt*.out and `*.log` files should be saved to archive details about the VERA Dev Env install for later reference. At this point, the scratch directory `${VERA_SCRATCH_DIR}` can be deleted.

More details about VERA and the installation of the VERA Dev Env are given in the following sections.

4 DETAILS ON INITIAL SETUP

Before any VERA-specific prerequisites can be installed, some initial setup is required. This section describes some of the tasks that must be performed to set up a machine so that it can be used to install the VERA Dev Env, clone the VERA repositories (or just unzip the VERA sources from a tarball), and build the various VERA components from source.

4.1 REQUESTING ACCESS TO VERA REPOSITORIES

If VERA is being installed from a release tarball, then users already have the source files. However, access must be granted if VERA is being pulled from the Git repositories on `code-int.ornl.gov`. Access to the

repositories also requires that an ORNL Universal Computer Access Access Management System (UCAMS) account be added to the GitLab site `code-int`, and explicit access is also required to the Git repositories in accordance with the VERA Technology Control Plan. Users must request access from their VUG representative and provide a list of the specific VERA Git repositories (or VERA components) needed.

4.2 SYSTEM CONFIGURATION CONSIDERATIONS

Before work begins, an accounting of the system resources should be made. For example:

- Make sure there are enough processors available for parallel compilation. It is assumed that there are at least eight cores available on which to run parallel builds, tests, or other jobs. If eight cores are not available, then users can reduce the parallel level, as will be obvious with each command.
- Use the fastest file storage system available for holding source code and for compiling and running test cases: try to avoid Network File System–mounted directories for compilation and testing.

Other considerations include:

- If the user is not building from a tarball, can the machine create SSH tunnels to download VERA and TPL repositories?
- Can the machine access the internet (e.g., `github.com`)? If not, ensure that all packages needed are available in the VERA and/or TPL repositories from `code-int` or gzipped tar files using an SSH tunnel/ftp download.
- Can a remote user access the machine if troubleshooting assistance is needed?

4.3 MINIMAL SYSTEM PACKAGE SETUP

Before proceeding to install the various VERA prerequisites, the following software packages must be installed on the system using the system's native package manager:

- **GCC gcc C compiler** (recent version is not necessary): builds the official version of GCC from source. The newer GCC version is used to build everything else.
- **GCC g++ C++ compiler** (recent version is not necessary): builds CMake from source. The newer GCC g++ version is used to build everything else.
- **patch**: supports CMake.
- **texinfo**: The `makeinfo` program is needed to install GCC from source.
- **GNU M4**: needed for the build of `autoconf`.
- **Git** (version 1.6.0 or newer): needed to clone several Git repositories.
- **git-lfs** (version 2.0 or newer): needed to clone several Git repositories.
- **Python** (version 2.7.9 or newer but not version 3.x): needed for the basic install scripts and some helper scripts used in VERA.

- **NumPy** (version 1.8.0 or newer): used in the COBRA-TF test harness.
- **bash**: needed for some of the shell scripts that refer explicitly to bash.
- **Perl** (VERA is tested with version v5.10.1, but newer versions should also work): needed to build libmesh/MOOSE/Peregrine and to run the VERA input parser `react2xml.pl`.
- **X11** (includes development libraries, as well as runtime libraries): needed to build and install the required VERA TPL QT.
- **ZLIB** (includes development libraries, as well as runtime libraries): needed to build and install the required VERA TPL HDF5.
- **libcurl*** (may not be installed by default on some systems): needed for the NetCDF TPL build, which is used by MOOSE/Bison.

The exact packages that users must install on different types of systems are specified in Section A.1.

Users will require many other software packages, but they should already be installed on any proper Linux/Unix system.

4.4 SSH SETUP FOR ACCESSING CODE-INT

To access the VERA repositories on `code-int.ornl.gov`, users must establish public/private SSH keys and register the public SSH with the `code-int.ornl.gov` GitLab website that is used to manage the VERA git repositories. In this guide, <ucams-id> is used to signify the user's 3-char ORNL UCAMS ID.

If the internal ORNL website `code-int` (`code-int.ornl.gov`) is not directly reachable from a user's machine (referred to as <your-machine> in this document), a remote SSH tunnel to `code-int` must be established as described in Section A.1).

First, if no public/private SSH keys have been set up on the machine, then they must be set up with

```
$ cd ~/.ssh && /usr/bin/ssh-keygen -t rsa -b 1024
```

Several prompts will appear. The defaults should be accepted by pressing the <ENTER> key three times.

The public key that was just created, `~/.ssh/id_rsa.pub`, must then be uploaded to GitLab using gocitrix.ornl.gov. To log in, open a browser (`Chrome` recommended), log in to gocitrix, open a browser within the session, and navigate to `code-int.ornl.gov`. Complete the registration, and under settings > SSH keys, copy your RSA public key to the text box and save. Then, send an email to vera-support@ornl.gov and ask for permission to access the `code-int` repositories.

If a user is not on the ORNL network with direct access to the machine `code-int`, then the SSH tunnel must be opened `code-int` using

```
ssh -fN tunnelinit
```

(See Section A.1.)

4.5 CREATE UNIX USER AND GROUP

To properly administer and protect VERA installations, it is recommended to set up the following:

- A `vera-admin` user: a Unix user account specifically for maintaining VERA installations.
- A `vera-users` group: a list of Unix users with access permission and a need to know for running the installed VERA components.

NOTE: The list of users added to `vera-users` must be given explicit permission to access **all** of the installed VERA components. If users are not sure who can be on this list, please contact vera-support@ornl.gov or another responsible representative from Consortium For Advanced Simulation Of Light Water Reactors (CASL) for guidance.

4.6 SET UP BASE DIRECTORIES FOR VERA

Once the `vera-admin` user account has been created, the `${VERA_DEV_ENV_BASE}` and `${VERA_INSTALL_DIR}` directories must be created by the `root` user or a user with `sudo` privileges. Then, ownership of these directories must be transferred to the `vera-admin` user. This transfer can be performed using the following commands:

```
export VERA_DEV_ENV_BASE=<some-dir>
mkdir ${VERA_DEV_ENV_BASE}
chown -R vera-admin ${VERA_DEV_ENV_BASE}

export VERA_INSTALL_DIR=<some-dir>
mkdir ${VERA_INSTALL_DIR}
chown -R vera-admin ${VERA_INSTALL_DIR}
```

To reduce the need for `root` or `sudo` access during installation, a `${VERA_ROOT_DIR}` containing the following directories can be used:

```
${VERA_ROOT_DIR}/
  ${VERA_DEV_ENV_BASE}/
  ${VERA_SCRATCH_DIR}/
  ${VERA_BASE_DIR}/
  ${VERA_BUILD_DIR}/
  ${VERA_INSTALL_DIR}/
```

By using this directory structure, someone with only `root` or `sudo` access must perform the following operations:

1. Ensure that all of the required system packages have been installed.
2. Create the `vera-admin` user.
3. Create the `vera-users` group.
4. Create the ${VERA_ROOT_DIR}.
5. Transfer ownership of ${VERA_ROOT_DIR} to `vera-admin`.

Once these actions have been performed, the `vera-admin` user will have all of the necessary permissions to build, test, install, and maintain the VERA installation.

5 DETAILS ON TPL INSTALLATION

A standard installation of VERA TPLs can be performed using the following command:

```
cd ${VERA_SCRATCH_DIR}/
${VERA_BASE_DIR}/vera_tpls/TPL_build/install_tpls.sh -DPROCS_INSTALL=8 \
  &> install_tpls.out
```

as described in Section 3.6. However, this is just a thin shell script that uses a CMake ExternalProject build of the TPLs.

The TPL build system can be pulled by using

```
cd ${VERA_BASE_DIR}/
git clone https://code.ornl.gov/VERA/vera_tpls
```

when pulling from GitHub.

The shell script `install_tpls.sh` creates a CMake binary directory:

```
${VERA_SCRATCH_DIR}/TPL_build/
```

It then runs CMake, pointing to the CMake ExteranlProject directory:

```
${VERA_BASE_DIR}/vera_tpls/TPL_build/
```

This TPL_build CMake project accepts a number of CMake cache variables which include:

- **CMAKE_BUILD_TYPE**: `Release` or `Debug`
- **ENABLE_SHARED**: `ON` means that only shared `*.so` libraries will be installed, whereas `OFF` means that only static `*.a` libraries will be installed.
- **CMAKE_INSTALL_PREFIX**: base path to TPL install location; TPLs are installed under this directory, with one subdirectory for each TPL.
- **TPL_LIST**: list of TPLs to install; can be a comma-separated list. The full list of TPLs (and the default value for this variable) is

  ```
  ZLIB;HDF5;NETCDF;LAPACK;QT;PETSC;SLEPC;SILO;SUNDIALS;BOOST
  ```

  Any subset of the TPLs can be listed and will be installed instead.
- **PROCS_INSTALL**: number of processors to use to build each TPL: at minimum of four is recommended because of the lengthy BOOST and QT build times. This is the last line path to `vera_tpls/TPL_build` repository subdirectory.

The install tool

```
${VERA_BASE_DIR}/vera_tpls/TPL_build/install_tpls.sh
```

is a bash script that sets all of the defaults needed to perform the basic build, including compiler and other options. However, as shown in Section 3.6, users can pass in any CMake cache variable, and it will override the defaults set in `install_tpls.sh`. For example, to install static libraries in addition to shared libraries, and to use 16 processes, use

```
cd ${VERA_SCRATCH_DIR}/
${VERA_BASE_DIR}/vera_tpls/TPL_build/install_tpls.sh \
   -DPROCS_INSTALL=16 \
   -DENABLE_SHARED=OFF \
   -DCMAKE_INSTALL_PREFIX=$VERA_DEV_ENV_COMPILER_BASE/tpls/opt_static \
   &> install_tpls.out
```

After the install script finishes calling `cmake` to configure the TPL build, it calls

```
make
```

to drive the build and install process.

Users can examine the script `install_tpls.sh` to see what it does, make needed modifications to the install process, and run the commands manually if needed.

6 DETAILS VERA COMPONENT BUILD, TEST, AND INSTALLATION

Once the VERA prerequisites (i.e., compilers, TPLs, other tools) have been installed, users must install the remaining VERA Git repositories for the desired components (if working from the version-controlled source), set up a build configuration, build, test, and finally, install. If users are working from an unzipped (release) tarball, then no extra git clones are needed, since all required source files will already be in place.

6.1 LOAD VERA DEV ENV

Before users can configure, build, test, and install any VERA component software, the installed VERA Dev Env must first be loaded into the user's shell. If it is not already loaded, then run

```
source ${VERA_DEV_ENV_COMPILER_BASE}/load_dev_env.sh
export PATH=$VERA_DEV_ENV_BASE/common_tools/anaconda2/bin:$PATH
```

6.2 CLONE REMAINING VERA COMPONENTS

When working from a tarball distribution of VERA, there is nothing left to clone, so users can skip this section.

However, when working from the version-controlled sources, the remaining VERA Git repositories can be cloned, for example, with

```
cd ${VERA_DIR}/
./clone_vera_repos.py
```

NOTE: The exact list of repositories to be cloned greatly depends on which VERA components with what functionality are desired or needed from VERA. Such information is not provided in this document. Users should ask their CASL VERA contact about what repositories they need to clone for specific purposes.

NOTE: Users must be part of the GitLab group `git@code-int`, which protects a repository; otherwise, when users clone, they will get an error message, like:

```
git clone git@code-int:VERA/MPACT

git clone git@code-int.ornl.gov:VERA/MPACT

Initialized empty Git repository in /home/usr/MPACT/.git/
Permission denied (publickey).
fatal: The remote end hung up unexpectedly
```

6.3 CHECKING OUT A SPECIFIC VERSION OF VERA

Users working from a tarball distribution of VERA can skip this section since the version is fixed.

However, when working from the Git repositories, various versions of VERA can be accessed.

The most recent version of VERA can be pulled using

```
cd ${VERA_DIR}/
gitdist pull
```

This will pull the most recent version of VERA at that moment from the official development `code-int:VERA/<REPO>/master` branches. Although a rigorous, almost continuous integration process ensures that all basic automated tests pass before anything is pushed into the `code-int/master` branches, mistakes do occur, and there may be some more detailed acceptance tests that may not run successfully on any particular version of VERA at any moment for what is in `code-int/master`.

Therefore, to reduce the probability of pulling and using a defective version of VERA components, it is recommended that more specific versions of VERA be pulled since they have undergone more testing. More expensive automated acceptance tests run nightly and weekly, as well as some larger, manually run tests in some cases. A specific version of the VERA repositories can be checked out using the `gitdist` tool and a `VERARepoVersion.txt` file. A `VERARepoVersion.txt` file is created when VERA is configured from local Git repositories. The file is written to the VERA build tree, and it is installed in the base install tree. Every automated VERA build/test posted to the VERA CDash server includes exact Git version information in the generated `VERARepoVersion.txt` file. For a subset of VERA repos, a `VERARepoVersion.txt` file looks like

Before a user updates to a specific version of VERA, a VUG representative may provide the user with a specific VERARepoVersion.txt file which is typically named `VERARepoVersion.<newdate>.txt` (for some specific date `<newdate>=YYYY-MM-DD`). This file can be used to check out the specific version with

```
cd ${VERA_DIR}/
gitdist fetch
gitdist --dist-version-file=~/VERARepoVersion.<newdate>.txt \
```

```
  checkout _VERSION_
```

(see `gitdist --help` for more details).

This creates a "detached head"state for the local VERA Git repositories, and each repository will be at the exact commit listed in the `VERARepoFileVersion.<newdate>.txt` file. Below is a message that users might receive from each of the repositories:

```
Note: checking out '00149f1'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b new_branch_name

HEAD is now at 00149f1... Merge remote branch 'origin/master' into
rsicc_2013_tarball_refactor_3104
```

This is not a troublesome state for the purposes of building the source.

Also, by using the `VERARepoVersion.txt` file for a previous install, users can see what repositories have been changed and what commits have been added. For example, to compare to an older install in

```
${VERA_DEV_ENV_BASE}/vera/<olddate>/
```

users can compare with the new recommended version by running

```
cd ${VERA_DIR}/
gitdist fetch
gitdist \
  --dist-mod-only \
  --dist-version-file=~/VERARepoVersion.<newdate>.txt \
  --dist-version-file2=${VERA_DEV_ENV_BASE}/vera/<olddate>/VERARepoVersion.txt \
  log --name-status _VERSION_ ^_VERSION2_
```

Many other types of Git commands can be used to supply one or two of the repository versions through a `VERARepoVersion.txt` file.

7 DETAILS ON FINALIZING VERA DEV ENV INSTALLATION

Once users have finished installing the VERA prerequisites consisting of the compilers, MPICH, other tools, and a complete set of TPLs shown in Section 2, and once users are finished testing the installs by building and testing needed VERA components, they must to set the permissions on the installed files and directories.

The directory permissions for the Unix tools, compilers, and TPLs can be opened up for all to use. This can be accomplished with the following command:

```
chmod -R a+rX <some-dir>
```

For example, world-readable permissions can be assigned using the following commands:

```
chmod -R a+rX ${VERA_DEV_ENV_BASE}/gcc-5.4.0
```

This should allow anyone to read (but not modify) any of the installed files and directories.

If the owning user is not `vera-admin`, then change the owning user to avoid accidental modifications by the original installer with the following command:

```
chown -R vera-admin <some-dir>
```

For example, if the original installer is not the current owning user, then the following commands can be run to transfer ownership to the `vera-admin` user:

```
chown -R vera-admin ${VERA_DEV_ENV_BASE}/gcc-5.4.0
```

This will avoid problems with accidental modifications to the installed directories by the original installer.

As an optional final step, to clean up disk space, users can delete the scratch space and the TPLs source repository by entering

```
rm -rf ${VERA_SCRATCH_DIR}
rm -rf ${VERA_BASE_DIR}/vera_tpls
```

WARNING: Users should only remove these directories if they are sure that the VERA Dev Env is correctly installed and the necessary dependent VERA components are building and running correctly, at least related to the installed VERA Dev Env.

Only the local VERA source and build tree should remain locally:

```
${VERA_BASE_DIR}/
```

This can be used to clone and build VERA components with the installed VERA Dev Env. However, if VERA will no longer be built under this directory, then it can be removed with:

```
rm -rf ${VERA_BASE_DIR}
```

This would only leave the install of the VERA Dev Env under `${VERA_DEV_ENV_BASE}/`.

At this point, the VERA Dev Env would be considered successfully installed, so users of the dev env must source the script and update path for Anaconda2—for example:

```
source ${VERA_DEV_ENV_BASE}//gcc-5.4.0/load_dev_env.sh
export PATH=$VERA_DEV_ENV_BASE/gcc-5.4.0/common_tools/anaconda2/bin:$PATH
```

This is done in the `.bash_profile` file, for instance.

8 DETAILS ON INSTALLING VERA

Once the VERA Dev Env is installed on a system and loaded into the user's shell environment, anyone with access to the VERA sources (either through a tarball or through the cloned Git repositories) can configure, build, test, and install the VERA components.

Information on the installation directory layout is provided at

```
${VERA_DIR}/doc/install/README.VERA
```

8.1 GET SOURCE FOR VERA COMPONENTS TO INSTALL

The process for obtaining the sources for the VERA components to install is identical to that for obtaining the sources to install and test the VERA Dev Env. This can be done simply with:

```
cd ${VERA_BASE_DIR}/
git clone git@code-int:VERA/VERA
cd VERA/
./clone_vera_repos.py
```

8.2 CONFIGURE, BUILD, AND TEST VERA COMPONENTS TO INSTALL

A configuration setting must be selected so that end users can build, test, and install various VERA components. For most systems, shared libraries are preferred because they typically use less disk space.

When installing VERA, it is important to build in a *very* shallow build directory—for example, using local scratch space to set up a build directory:

```
cd /scratch/<user-id>/
mkdir VERA_BUILD
cd VERA_BUILD/
ln -s $VERA_DIR/cmake/std/gcc-5.4.0/do-configure.MPI_RELEASE_SHARED \
    do-configure
```

If a shallow build directory is not used, then CMake may fail to replace the RPATHs in the executables because the strings could be too long.

Once a do-configure script is set up, users must configure the system to point to the final install location. Assuming the target is

```
export VERA_INSTALL_DIR=/tools/vera_installs/`date +%Y-%m-%d`
```

configure the install with

```
cd /scratch/<user-id>/VERA_BUILD/
./do-configure \
   -D CMAKE_INSTALL_PREFIX=$VERA_INSTALL_DIR \
   -DVERA_ENABLE_DEVELOPMENT_MODE=OFF \
   -D VERA_ENABLE_ALL_PACKAGES=ON &> configure.out
```

If the configure passed, then build and test using

```
make -j8 &> make.out
ctest -j8 &> ctest.out
```

(see Section 3.7).

All of the tests should pass, but if they do not, please email `vera-support@ornl.gov` and provide the tail of `ctest.out` and a copy of your `do-configure` script.

8.3 INSTALL BUILT VERA COMPONENTS

Before running `make install`, users may need to set up the base directory for the install by entering

```
cd /tools/
mkdir vera_installs
chgrp -R vera-users vera_installs
chmod 750 vera_installs
chmod g+s vera_installs
```

to protect VERA appropriately. (See the `vera-users` group described in Section 4.5.)

After a successful configure, build, and test has been performed, an install can be performed by entering

```
cd /scratch/<user-id>/VERA_BUILD/
umask 0007
make -j8 install &> make.install.out
```

Setting `umask 0007` will ensure that only the `vera-users` group (or whatever it is called on the given system) has access to the installed VERA software.

This should set up an installation directory that looks like

```
${VERA_INSTALL_DIR}/
  bin/
  lib/
  share/
  README
  README.react2xml
  ...
```

8.4 DOCUMENTATION FOR INSTALLED VERA COMPONENTS

Once VERA is installed, information on how to access the installed VERA components, along with their documentation and examples, can be found in the readme file:

```
${VERA_INSTALL_DIR}/
  README
```

APPENDIX A. ADDITIONAL INSTALLATION INFORMATION

A ADDITIONAL INSTALLATION INFORMATION

A.1 SET UP REMOTE SSH TUNNEL

To access the Git repositories on `code-int.ornl.gov` when outside the ORNL network, an SSH tunnel must be set up through `login1.ornl.gov`. This requires the user to have an active UCAMS account with the 3-char <ucams-id>. Once established, this SSH tunnel will set up a machine name called `casl-dev` on the local machine that can then be used in Git commands.

In the home directory, create the following file:

```
~/.ssh/config
```

This file contains

```
host tunnelinit
Hostname login1.ornl.gov
User <ucams-id>
LocalForward 28881 code-int.ornl.gov:22

Host code-int
HostKeyAlias code-int.ornl.gov
Hostname localhost
Port 28881
User <ucams-id>
```

Please make sure to change the above port numbers to avoid conflict with other ports being used on the system or other SSH tunnels. If multiple users use the same port numbers, then collisions will occur, or the host machine will disallow the connection altogether.

To set up the SSH tunnel, in any terminal, type the command

```
ssh -fN tunnelinit
```

Each user will be prompted for a PASSCODE; this is the user's pin+6digits from their SecurID token.

The SSH tunnel will remain open for some amount of time and will continue to remain open if it is being actively used. However, it may be important in some cases to ensure that the tunnel is closed before logging off or performing other tasks. If a user creates an SSH tunnel, then the user should be able to close the SSH tunnel. Because the SSH tunnel is in the background, the following command should be used to find the SSH tunnel process:

```
ps aux | egrep ssh -fN tunnelinit
```

This will return the following type of response:

```
<ucams-id> 10535  0.3  0.0  66032  3804 ?   Ss   11:08   0:19 ssh -fN tunnelinit
```

The kill command can then be used to end the tunnel process:

```
kill -9 <process number>
```

In this case `10535`, the process number will always be the second item in the returned fields from the `ps aux` command.

This will close the SSH tunnel.

A.2 MINIMAL SYSTEM PACKAGE SETUP ON VARIOUS SYSTEMS

Package managers on different systems have different names for the needed packages they install, as described in Section 4.3.

The following table lists the names of the packages that must be installed on three different machines where VERA has been installed in the past. In addition, the last column specifies the executable that should be in the system's path if that the package is installed. Testing for the existence of these executables can be accomplished with the `which` command. For example:

```
which gcc
```

may return

```
/usr/bin/gcc
```

If a path to the executable is not returned by the `which` command, then the package is not installed.

Package	**CentOS 6.6/Redhat 6.4**	**Ubuntu 14.04.1**	**openSUSE 11.4**	**Executables**
GCC C Compiler	gcc	gcc	gcc	gcc
GCC C++ Compiler	gcc-c++	g++	g++	gcc-c++
GNU M4	m4	M4	M4	m4
GNU texinfo	texinfo	texinfo		makeinfo
Git	git	git	git	git
Python	python-dev	python-dev	python-dev	python-dev
NumPy	numpy	python-numpy	python-numpy	???
Bash	bash	bash	bash	bash
Perl	perl	perl-base	perl	perl
X11 (dev)	libX11-devel	libx11-dev		
Xorg (dev)	xorg-x11-server-devel	xorg-dev		
Zlib (dev)	zlib-devel	zlib-dev		
libcurl (dev)	libcurl	curl	curl	

There are no executables for the X11, Xorg, or Zlib development libraries, and no executables are needed for the libcurl library. To verify that these packages are installed, users can search for a required library (in the user's `LD_LIBRARY_PATH`) or a required include file or directory in `/usr/include`. Suggested files/folders to check to verify installation of the X11, Xorg, and Zlib development libraries are provided in the following table.

On CentOS and Redhat systems, the command `yum install <package>` installs a package—for example:

```
sudo yum install gcc-c++
```

Package	usr/lib or /usr/lib64	/usr/include
X11 (dev)	libX11.so	X11
Xorg (dev)		xorg
Zlib (dev)	libz.so	zlib.h
Zlib (dev)	libcurl.so	curl/curl.h

On Ubuntu systems, the command `apt-get install <package>` installs a package—for example:

```
sudo apt-get install g++
```

On SUSE and openSUSE systems, the command `zypper install <package>` installs a package. For example:

```
sudo zypper install gcc-c++
```

A user with `sudo` privileges must install these packages because they are installed into the base system directories. However, it is possible to use some package managers to install missing packages to a different location that does not require root or sudo access. However, this is beyond the scope of this document. Consult the documentation for the package manager on your system for more information.

A.3 OFFICIAL VERA TPL VERSIONS

The source code for the official TPL versions that are installed using the `install_tpls.sh` tool (Section 3.6) are stored in the `vera_tpls` Git repository, and the current official versions are shown in the following table.

Third Partly Library (TPL)	Version
LAPACK	3.3.1
Boost	1.55.0
ZLib	1.2.7
HDF5	1.10.2
NETCDF	4.4.1
PETSc	3.6.4
SLEPC	3.6.3
Silo	4.10.2
SUNDIALS	2.9.0
QT	4.8.7

The exact version of TPLs matching this particular version of VERA is always indicated by the Git tag for the `vera_tpls` repository, as given in Section 3.6.

A.4 SHARED VS. STATIC LIBRARIES

The default install of the VERA TPLs in Section 3.6 and VERA itself in Section 3.7 use shared libraries. However, these instructions also describe how to install static libraries. In general, users should prefer shared libraries over static libraries on most platforms: shared libraries use less disk space, link faster, and allow for quick bug fixes and simpler upgrades. However, if RPATH is not written into the executables and shared libraries, then users will have to set `LD_LIBRARY_PATH` pointing to the location of the correct shared libraries.

Moreover, some systems (especially some HPC machines) either require or strongly recommend the usage of static libraries. In addition, static libraries are more self-contained and are less sensitive to the machine environment from which they run and they never have RPATH issues. But executables built using static libraries take up far more disk space and take longer to link.