Multi-Repository Development and Integration in CASL using TriBITS

Roscoe A. Bartlett

Oak Ridge National Laboratories

Trilinos User Group Meeting

November 6, 2013

Motivations and Outline

Outline:

- Overview of CASL VERA Development Efforts
- TriBITS Support for Multi-Repository Projects
- Multi-Repository Integration Models and Processes
- Miscellaneous Issues and Topics for TriBITS and VERA
- Future TriBITS Development

Motivations for this presentation:

- Describe how CASL does development using TriBITS
- Inform about current status of TriBITS
- Inspiration for other projects similar to CASL
- Provide ideas for Trilinos development and distribution
- Discuss future of TriBITS development and distribution

Overview of CASL VERA Development Efforts

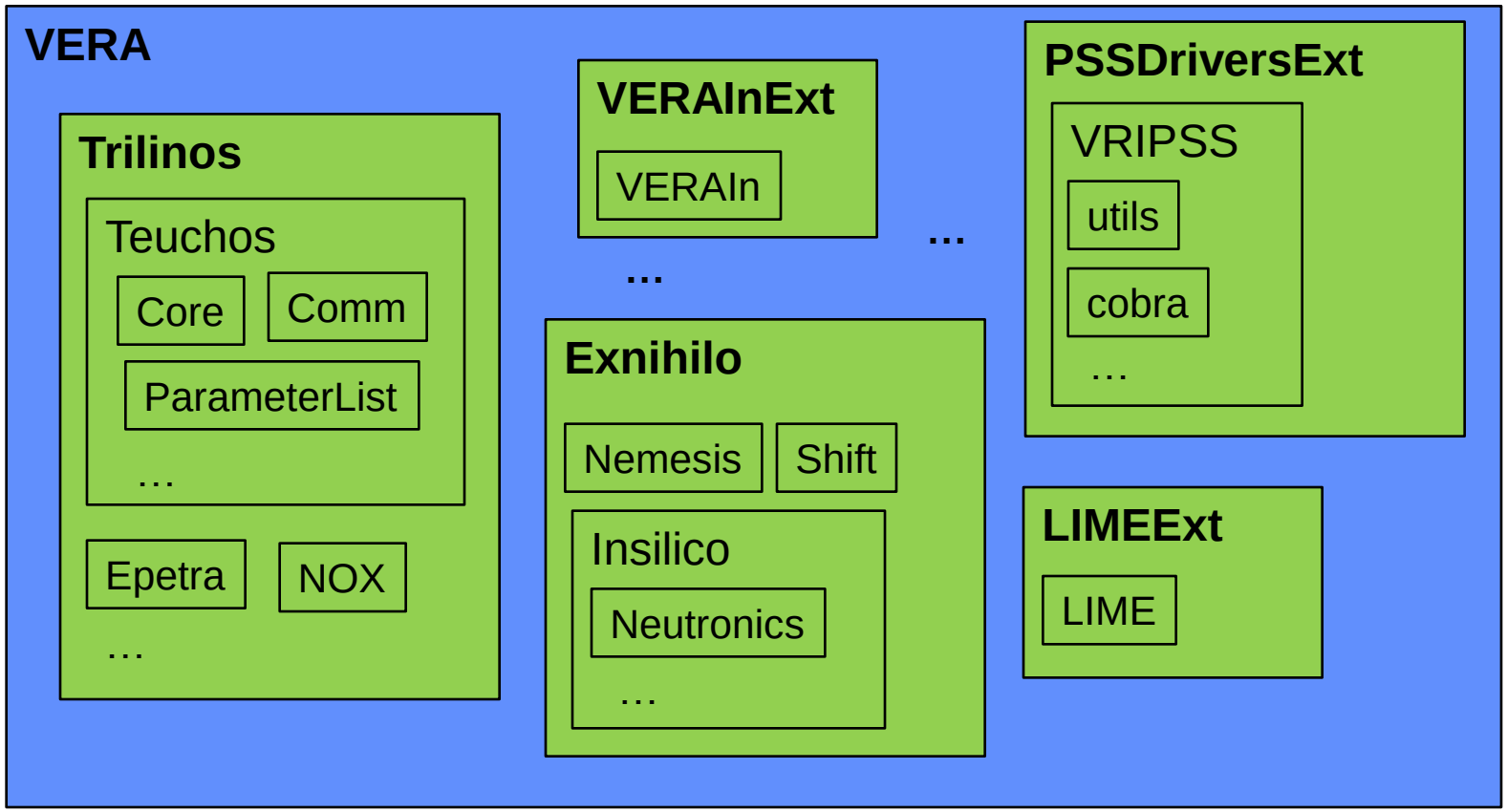
Overview of CASL

- **CASL: C**onsortium for the **A**dvanced **S**imulation of **L**ightwater reactors
- DOE Innovation Hub including DOE labs, universities, and industry partners
- Goals:
  - Advance modeling and simulation of lightwater nuclear reactors
  - Produce a set of simulation tools to model lightwater nuclear reactor cores to provide to the nuclear industry: **VERA: Virtual Environment for Reactor Applications**.
- Phase 1: July 2010 – July 2015
- Phase 2: Under review by DOE
- Organization and management:
  - ORNL is the hub of the Hub
  - Milestone driven (6 month plan-of-records (PoRs))
  - Focus areas: **Physics Integration (PHI)**, Thermal Hydraulic Methods (THM), Radiation Transport Methods (RTM), Advanced Modeling Applications (AMA), Materials Performance and Optimization (MPO), Validation and Uncertainty Quantification (VUQ)

VERA Development Overview

- VERA Development is complicated in almost every way ☹
- VERA Currently Composed of:
  - 17 different git repositories on casl-dev.ornl.gov (clones of other repos) most with a different access list (NDAs, Export Control, etc.)
  - 14 different TriBITS repositories providing packages
  - VERA: 144 SE Packages, 12 TPLs
  - VERA-Baseline: 39 SE Packages, 11 TPLs
- TriBITS (Tribal Build, Test, and Integrate System):
  - Based on CMake/CTest/CDash
  - Scalable package dependency system
- Software Development Process:
  - Official definition of VERA is 'master' branch of git repos under: casl-dev.ornl.gov:/git-root/*
  - Primary development platform: CASL Fissile/Spy Machines
  - VERA integration maintained by continuous and nightly testing:
    - Pre-push CI testing: checkin-test-vera.sh, cloned VERA git repos, on Fissile machine
    - Post-push CI testing: CTest/CDash, all VERA git repos, shared libs
    - Nightly CI testing: Debug and Release builds
    - 100% passing builds and tests!
  - VERA snapshots and releases are taken off of 'master' branches on casl-dev git repos.

VERA Meta-Project, Repositories, Packages & Subpackages



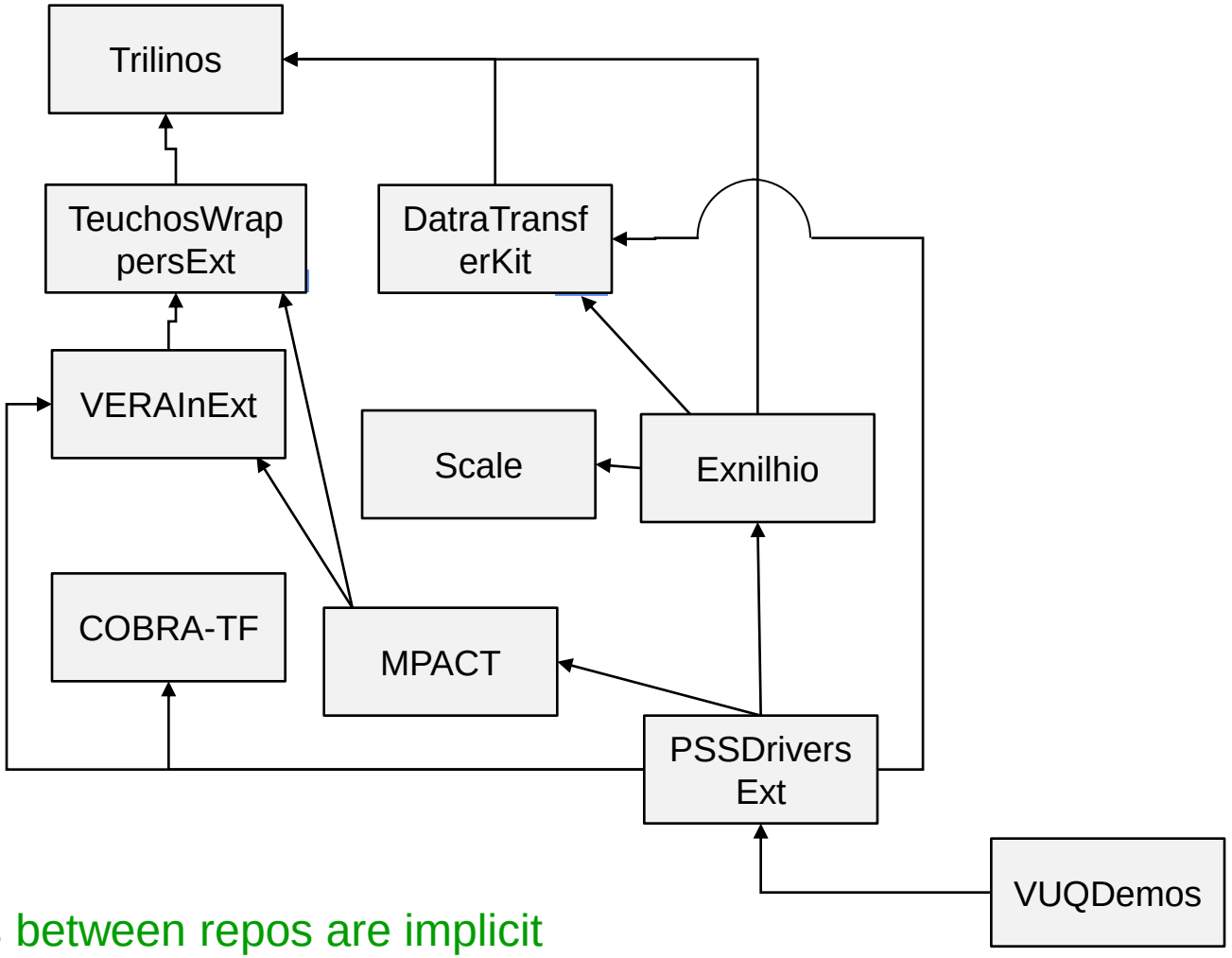
- **VERA**: Git repository and TriBITS meta-project (contains no packages)
- Git repos and TriBITS repos: **Trilinos, VERAInExt, LIMEExt, Exnihilo, …**
- TriBITS packages: **Teuchos, Epetra, VERAIn, Insilico, LIME, VRIPSS, …**
- TriBITS subpackages: **TeuchosCore, InsilicoNeutronics, …**
- TriBITS SE (Software Eng.) packages: **Teuchos, TeuchosCore, VERAIn, Insilico, InsilicNeutronics, …**

VERA/cmake/ExtraRepositoriesList.cmake

```
SET( VERA_EXTRAREPOS_DIR_REPOTYPE_REPOURL_PACKSTAT_CATEGORY
  Trilinos       ""   GIT  casl-dev:/git-root/Trilinos    ""  Continuous
  Dakota  Trilinos/packages/TriKota/Dakota  GIT
    casl-dev:/git-root/Dakota  NOPACKAGES  Continuous
  TeuchosWrappersExt "" GIT casl-dev:/git-root/TeuchosWrappersExt ""  Continuous
  VERAInExt      ""   GIT  casl-dev:/git-root/VERAInExt      ""  Continuous
  DataTransferKit ""   GIT casl-dev:/git-root/DataTransferKit ""  Continuous
  COBRA-TF       ""   GIT  casl-dev:/git-root/COBRA-TF       ""  Continuous
  Scale          ""   GIT  casl-dev:/git-root/Scale          ""  Continuous
  Exnihilo       ""   GIT  casl-dev:/git-root/Exnihilo       ""  Continuous
  MPACT          ""   GIT  casl-dev:/git-root/casl_mpact     ""  Continuous
  MOOSEExt       ""   GIT  casl-dev:/git-root/MOOSEExt      ""  Continuous
  MOOSE          MOOSEExt/MOOSE   GIT
     casl-dev:/git-root/MOOSE   NOPACKAGES  Continuous
  LIMEExt        ""   GIT  casl-dev:/git-root/LIMEExt      ""  Continuous
  Mamba          ""   GIT  casl-dev:/git-root/casl_mamba     ""  Continuous
  hydrath        ""   GIT  casl-dev:/git-root/hydrath       ""  Nightly
  PSSDriversExt  ""   GIT  casl-dev:/git-root/casl_vripss    ""  Continuous
  VUQDemos       ""   GIT  casl-dev:/git-root/VUQDemos     ""  Continuous
  )
```

- Official version of VERA in on master branch used for CI & Nightly testing
- Partial set of repos can be cloned (protected by different groups)
- Non-git repos are converted into git repos: Dakota, Scale, MOOSE

Dependencies Between Selected VERA Repositories



- Dependencies between repos are implicit
- Real dependencies are between packages in repos

checkin-test-vera.sh

```
$VERA_BASE_DIR/VERA/checkin-test.py \
--src-dir=$VERA_BASE_DIR_ABS/VERA \
--extra-repos-file=project \
--extra-repos-type=Continuous \
--ignore-missing-extra-repos \
--default-builds=MPI_DEBUG,SERIAL_RELEASE \
-j16 \
--ctest-timeout=400 \
$EXTRA_ARGS
```

- Very thin bash script wrapper for TriBITS checkin-test.py
- Automatically picks up cloned repos listed in ExtraRepositoriesList.cmake
- Safe pushes requires all affected repos to be cloned and available

Current Adoption of TriBITS in CASL

- VERA Repositories that are also independent projects using TriBITS:
  - Trilinos: SNL
  - SCALE: ORNL
    - Requires GCC 4.6.1+ and Intel 13.1+
    - Mixed Fortran, C, C++
    - Linux builds
    - Windows builds
  - MPACT: Univ. of Mich.
    - Requires GCC 4.6.1+ and Intel 13.1+
    - Mostly Fortran
    - Windows builds
  - Exnihilo: ORNL
    - Mostly C++ with some Fortran 90/77, Python, etc.
    - Contains Denovo, Shift, Insilico
  - COBRA-TF: Penn. State Univ.
    - Mostly Fortran 77 and 90
- Native TriBITS repos just providing packages: TeuchosWrappersExt, VERAInExt, DataTransferKit, LIMEExt
- VERA Repositories/packages not using TriBITS as native build system but have secondary native TriBITS support: DAKOTA, MAMBA, Hydra-TH
- VERA Repositories/packages not providing secondary TriBITS build: MOOSE

CASL VERA Development Challenges

- Large legacy codes under active development being constantly updated => Memory leaks, memory access errors, etc.
- Fortran 2003/2008 features being used causing portability problems (bugs with gfortran 4.6.1)
- Long configure and build times, unnecessary rebuilds after configure, etc.
- Non-native configure/build of MOOSE not using TriBITS/CMake
- Large stack of required TPLs:
  - MPI, BLAS, LAPACK, Boost, Netcdf, Zlib, PETSC, HDF5 QT, MOAB, SILO, DDRIV, MUMPS, SCALAPAK (and others for HydraTH).
  - Different problems with different TPLs on different platforms
  - QT (required by SCALE) almost always hard to build and install
- Testing:
  - Little unit and verification testing for major features in legacy codes
  - Many full system acceptance tests that exist require 1000s of processors and hours to run. (currently not automated).
  - Lack of hardware for doing automated testing
- Some package codes don't support shared library builds => Creates problems with CI and Nightly testing and installations
- Insufficient staff for infrastructure, testing, deployments, support etc.

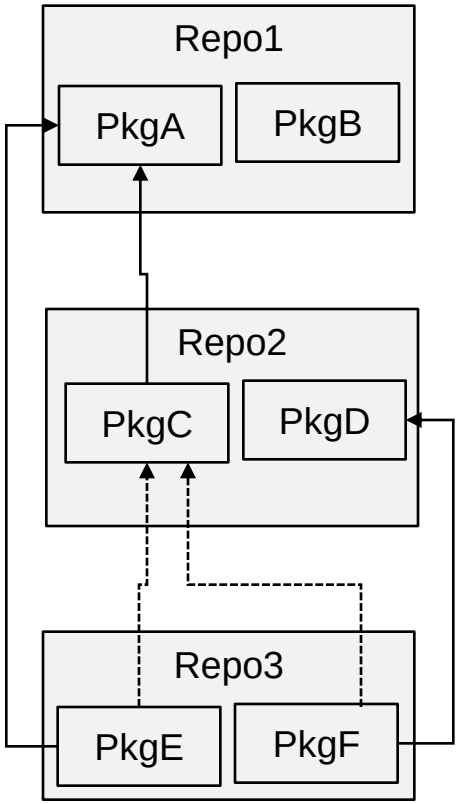
TriBITS Support for Multi-Repository Projects

Managing Compatible Repos and Repo Versions

- Issues that need to be addressed:
  - Flexibility for development inside and outside of particular project
  - Managing changes between different repos versions and projects.
  - Full tracking of changes and updates
  - Reproducibility of prior versions
  - Repos may be missing with optional package dependencies. (ex. Repo2 may be missing.)

- Issues in selecting compatible repo versions:
  - Even if packages in upstream repos maintains backward compatibility new features are always being added.
  - Packages in repos often break backward compatibility.

- How to manage compatible repos versions?



Managing Multiple Compatible Repo Versions with git?

- Snapshot all repos into one big repo (e.g. SIERRA/Trilinos style):
  - Advantages:
    - One set of SHA1s, easy to do git bisect
    - One repo to pull and build final version
  - Disadvantages:
    - Hard to coordinate changes back and forth to native component repos
    - Hard for regular developers to try out updated versions of native repos
    - Does not allow for partitioning based on access control

- Use git modules:
  - Advantages:
    - Built-in git support and documentation
    - Individual repos stay independent (let git do its job).
  - Disadvantages:
    - Extra commands to pull component repos
    - Updating repos versions is complex for non-git savvy developers

- Clone repos under master repo (egdist) and track sets of compatible repos as files and provide tools for accessing specific versions (<Project>RepoVersion.txt files)

=> Primary approach currently directly supported by TriBITS

egdist : eg/git for collection of git repos

- **egdist**: Simple stand-alone Python tool for distributing eg/git commands across multiple git repos. Contained in tribits/common_tools/git/egdist.

  Usage: egdist [egdist options] <raw-git-command> [git options]

- .egdist file in base git repo:
  Trilinos
  Trilinos/packages/TriKota/Dakota
  …

Example:

```
$ egdist status
*** Base Git Repo: VERA
(On branch master)
*** Git Repo: Trilinos
(On branch master)
*** Git Repo: Trilinos/packages/TriKota/Dakota
…
```

- Common bulk commands: pull, push, local-stat, log -1
- Works well for < 10-20 repos, not for 100s of repos!

Pre-Push Testing and Pushing of Multiple Repos

```
$VERA_BASE_DIR/VERA/checkin-test.py \
--src-dir=$VERA_BASE_DIR_ABS/VERA \
--extra-repos-file=project \
--extra-repos-type=Continuous \
--ignore-missing-extra-repos \
--default-builds=MPI_DEBUG,SERIAL_RELEASE \
-j16 \
--ctest-timeout=400 \
$EXTRA_ARGS
```

- Very thin bash script wrapper for TriBITS checkin-test.py
- Automatically picks up cloned repos listed in ExtraRepositoriesList.cmake
- Safe pushes requires all affected repos to be cloned and available

Restricting Post-Push CI and Nightly Testing

- Projects don't want/need to directly process and test all TriBITS packages (i.e. from external repos).

- Restricting the set of packages directly processed by TribitsCTestDriverCore.cmake code:

  SET(<REPO_NAME>_NO_IMPLICIT_PACKAGE_ENABLE ON)
  SET(<REPO_NAME>_NO_IMPLICIT_PACKAGE_ENABLE_EXCEPT <PKG1> <PKG2> …)

  - This results in all of the packages in the TriBITS repo <REPO_NAME> to be skipped (not disabled) in direct processing by the TriBITS CI server driver.

  Example from VERA:

  SET(Trilinos_NO_IMPLICIT_PACKAGE_ENABLE ON)
  SET(LIMEExt_NO_IMPLICIT_PACKAGE_ENABLE ON)
  SET(Scale_NO_IMPLICIT_PACKAGE_ENABLE ON)
  SET(Exnihilo_NO_IMPLICIT_PACKAGE_ENABLE ON)
  SET(Exnihilo_NO_IMPLICIT_PACKAGE_ENABLE_EXCEPT Insilico)

- Also, direct list of packages to testing in Nightly testing can be listed to test specific sets of packages by setting <PROJECT_NAME>_PACKAGES.

<Project>RepoVersion.txt

- How to keep track of compatible sets of repos?
  => TriBITS support for <Project>RepoVersion.txt file:
  *** Base Git Repo: SomeBaseRepo
  e102e27 [Mon Sep 23 11:34:59 2013 -0400] <author1@someurl.com>
  First summary message
  *** Git Repo: ExtraRepo1
  b894b9c [Fri Aug 30 09:55:07 2013 -0400] <author2@someurl.com>
  Second summary message
  *** Git Repo: ExtraRepo2
  97cf1ac [Thu Dec 1 23:34:06 2011 -0500] <author3@someurl.com>
  Third summary message
  ...
- When setting -D<PROJECT>_GENERATE_REPO_VERSION_FILE=ON, the file <Project>RepoVersion.txt gets:
  - generated in the build base directory,
  - echoed in the configure output (therefore archived to CDash),
  - installed in the base install directory,
  - included in the source tarball ('make package_source'),
  - installed in the base install directory from the untarred source.

- egdist --dist-repo-file=<Project>RepoVersionFile.<somedate>.txt [other options]

Using <Project>RepoVersion.txt for Snapshot Distributions

- Known “good” versions of the Project code are recorded as <Project>RepoVersion.txt files (e.g. archived on CDash).
- Example: If a given Nightly build of Project passed on all platforms then we can give the associated <Project>RepoVersion.txt file as the “version” for a client to use.
- Send client file <Project>RepoVersion.<newdate>.txt for “good” version to install.
- Client gets updated version:

```
$ cd <SOME-BASE-DIR>/<PROJECT>
$ egdist fetch
$ egdist --dist-version-file=~/<PROJECT>RepoVersion.<newdate>.txt \
  checkout _VERSION_
```

- Client can see changes since a previous installs of <Project>:

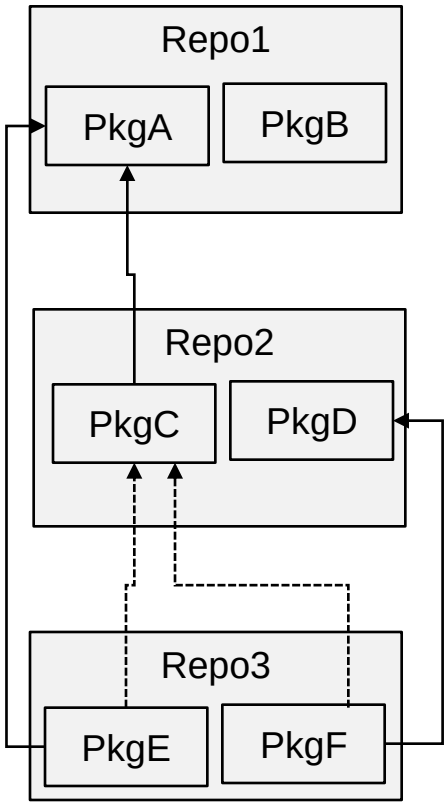
```
$ egdist fetch
$ egdist \
 --dist-version-file=~/<PROJECT>RepoVersion.<newdate>.txt \
 --dist-version-file2=${INSTALL_BASE}/<olddate>/<Project>RepoVersion.txt \
 log-short --name-status _VERSION_ ^_VERSION2_
```

Dealing with Missing Repos and Packages

What if Repo2 is missing? Can we still configure and build the remaining packages in Repo3?

```
# Repo3/PkgE/cmake/Dependencies.cmake
SET(LIB_REQUIRED_DEP_PACKAGES PkgA)
SET(LIB_OPTIONAL_DEP_PACKAGES PkgC)
…
TRIBITS_ALLOW_MISSING_EXTERNAL_PACKAGES(PkgC)
```

```
# Repo3/PkgF/cmake/Dependencies.cmake
SET(LIB_REQUIRED_DEP_PACKAGES PkgD)
SET(LIB_OPTIONAL_DEP_PACKAGES PkgC)
…
TRIBITS_ALLOW_MISSING_EXTERNAL_PACKAGES(PkgC  PkgD)
```

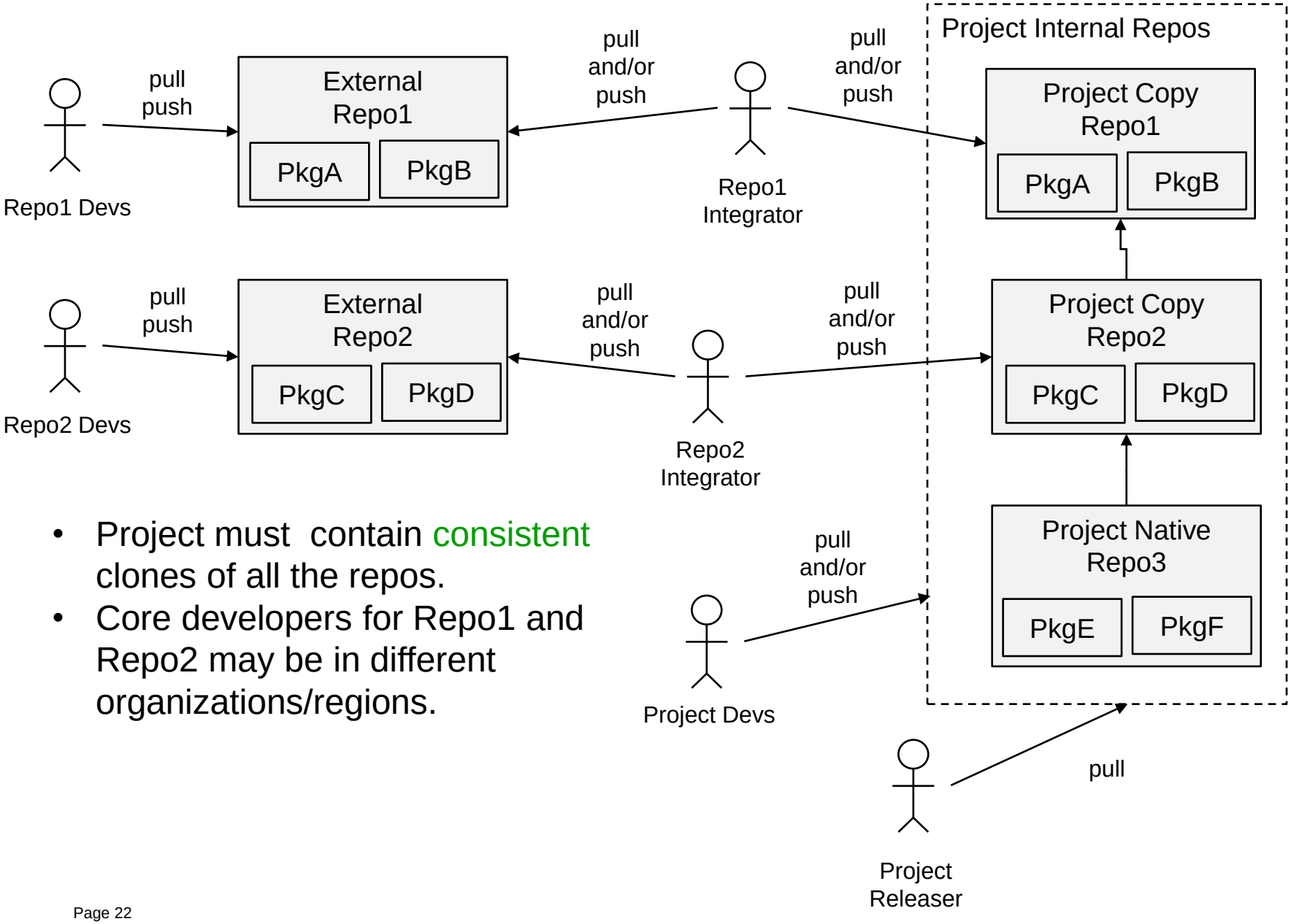


Now when configuring the Project with Repo2 missing TriBITS automatically adjusts:

```
WARNING: PkgC is being ignored since its directory is missing and
PkgC_ALLOW_MISSING_EXTERNAL_PACKAGE = TRUE!
…
WARNING: Setting <PROJECT>_ENABLE_PkgF=OFF because PkgD is a
required missing package!
```

Multi-Repository Integration Models and Processes

Integrating Repos into Project: External and Internal



- Project must contain consistent clones of all the repos.
- Core developers for Repo1 and Repo2 may be in different organizations/regions.

Multi-Repository Integration Models

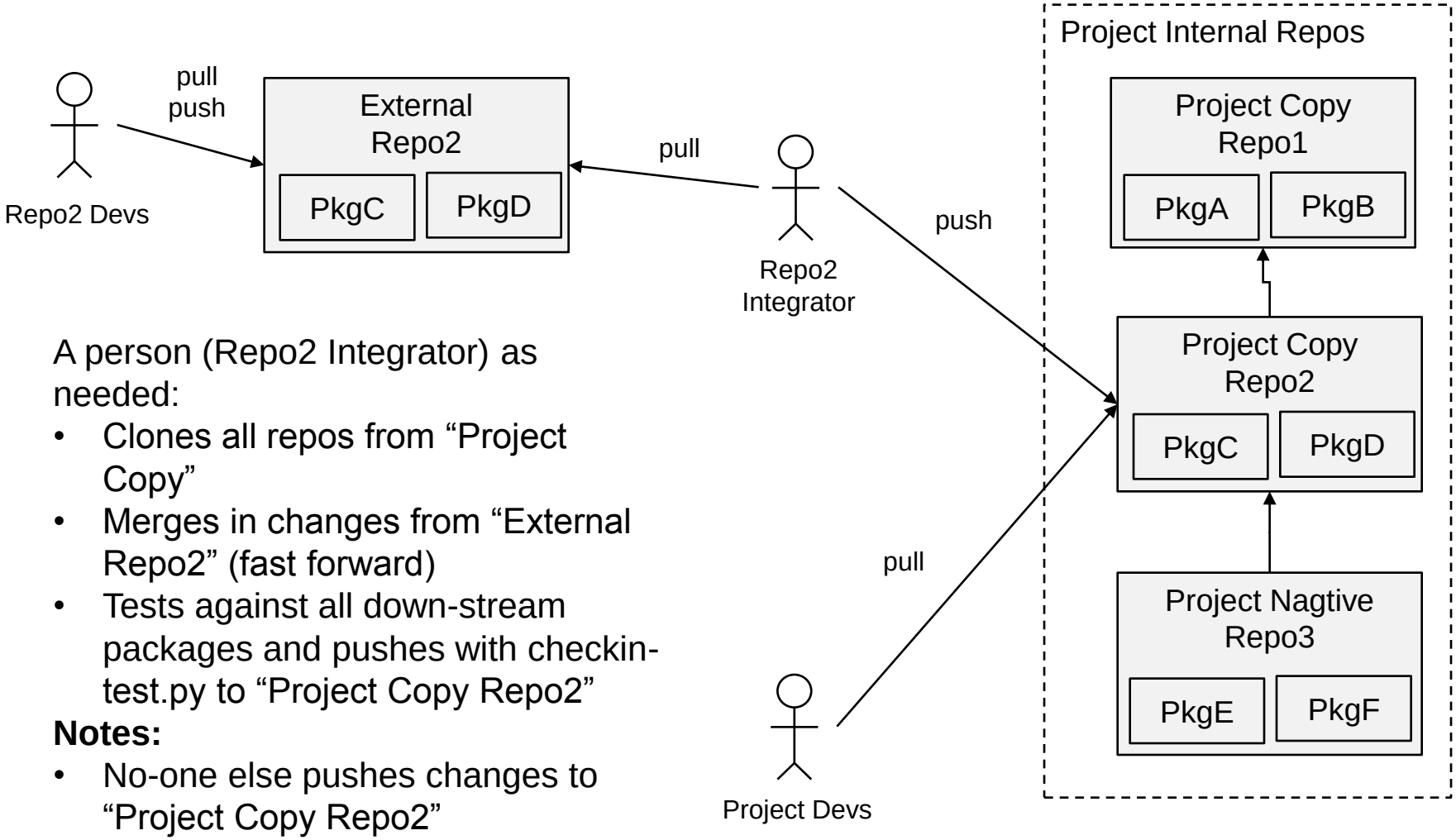
- Range of development and sync models (external dev to internal dev)
  - External repo is manually synced into project/master as needed.
  - External repo is synced automatically using sync server into project/master using the checkin-test.py script.
  - Both external and internal repos pushed to by different development groups with sync servers running one way or both ways.
  - Internally managed repo is synced to an external repo on some schedule to make available to other developers and users and changes from external repo may or may not be synced back into internal repo.
  - Internally managed repo

- A given repo may shift between different integration models at different periods of time (e.g. Trilinos, COBRA-TF)

- Integration of different repos should be done independently if possible (e.g. errors in MPACT should not stop pushes of Exnihilo and visa versa).

- Non-backward compatible changes to upstream repos require coordinated development and combined pushing to project/master

External Repo is manually synced



A person (Repo2 Integrator) as needed:

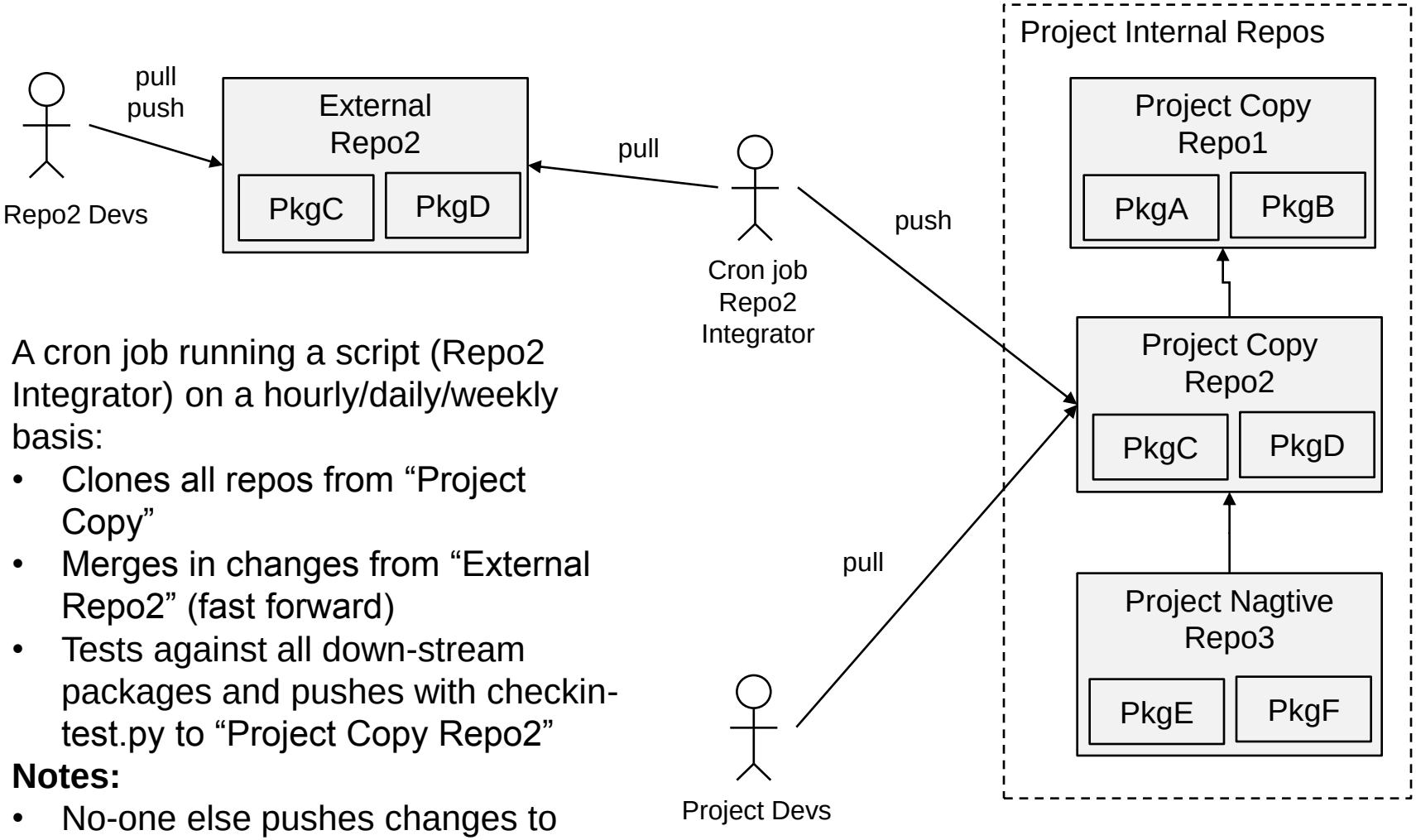
- Clones all repos from “Project Copy”
- Merges in changes from “External Repo2” (fast forward)
- Tests against all down-stream packages and pushes with checkin-test.py to “Project Copy Repo2”

Notes:

- No-one else pushes changes to “Project Copy Repo2”
- Good when changes are **not** urgent for Project or when “External Repo2” is unstable

VERA Examples: DataTransferKit

External repo is synced automatically using sync server



A cron job running a script (Repo2 Integrator) on a hourly/daily/weekly basis:

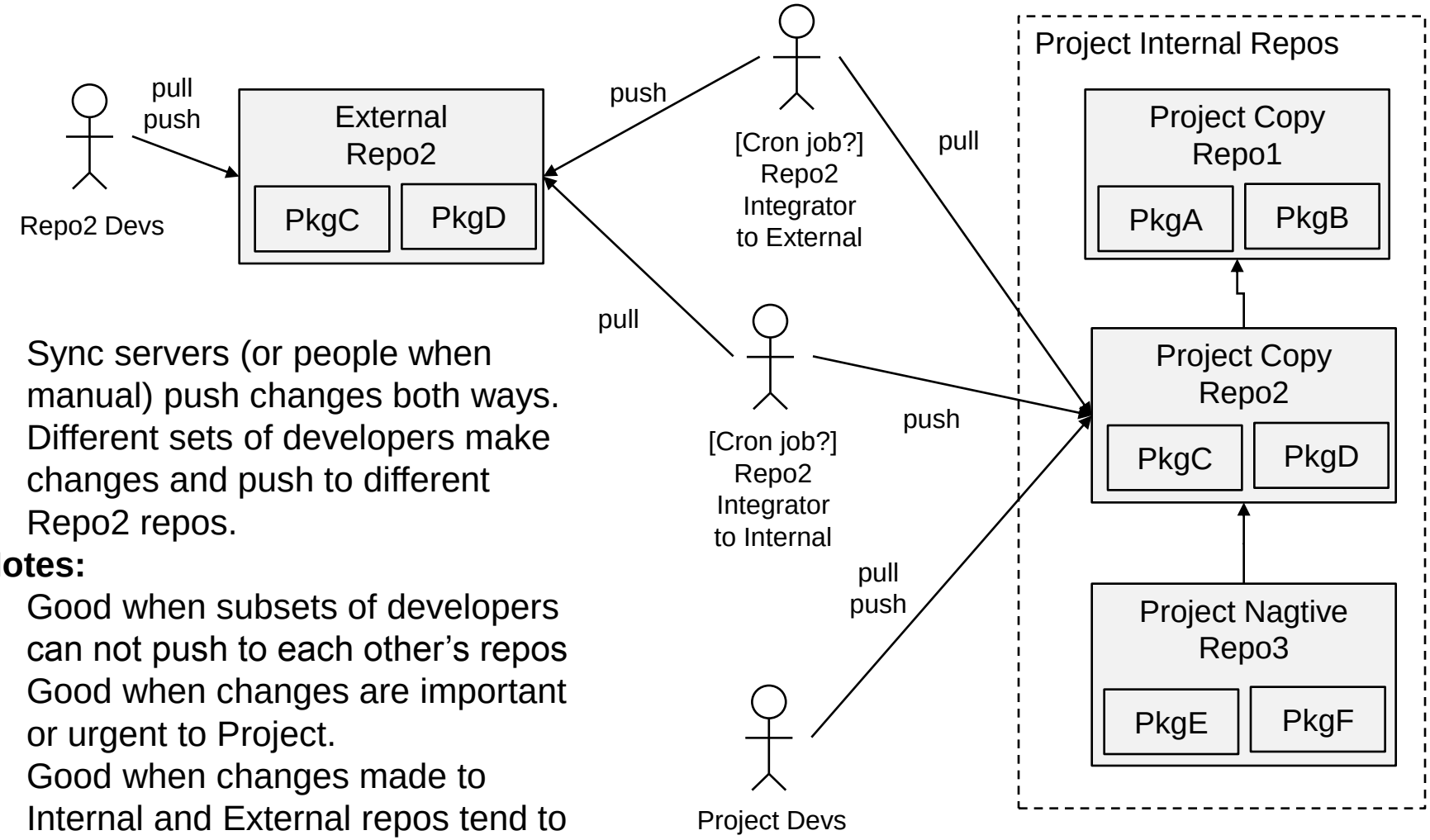
- Clones all repos from “Project Copy”
- Merges in changes from “External Repo2” (fast forward)
- Tests against all down-stream packages and pushes with checkin-test.py to “Project Copy Repo2”

Notes:

- No-one else pushes changes to “Project Copy Repo2”.
- Good when changes are important or urgent to Project and “External Repo2” is fairly stable.

VERA Examples: SCALE, Exnihilo, MPACT

Both external and internal repos pushed to



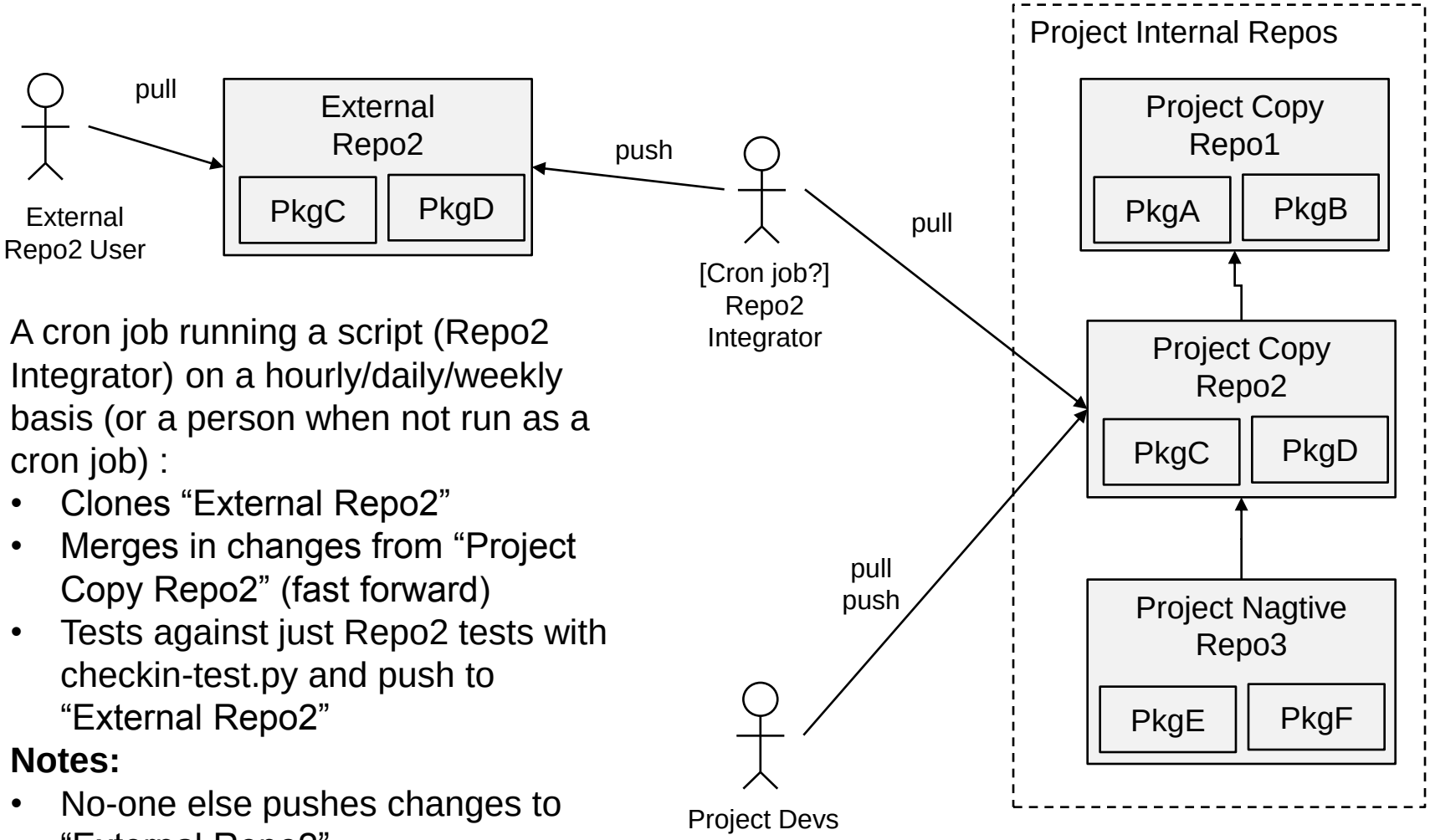
- Sync servers (or people when manual) push changes both ways.
- Different sets of developers make changes and push to different Repo2 repos.

Notes:

- Good when subsets of developers can not push to each other's repos
- Good when changes are important or urgent to Project.
- Good when changes made to Internal and External repos tend to be independent.
- Most complex and danger of merge conflicts that someone has to resolve!

VERA Examples: Trilinos (future), COBRA-TF (future)

Internally managed repo is synced to an external repo



A cron job running a script (Repo2 Integrator) on a hourly/daily/weekly basis (or a person when not run as a cron job) :

- Clones “External Repo2”
- Merges in changes from “Project Copy Repo2” (fast forward)
- Tests against just Repo2 tests with checkin-test.py and push to “External Repo2”

Notes:

- No-one else pushes changes to “External Repo2”
- Good when you just want to make changes available to external users on a continuous basis.

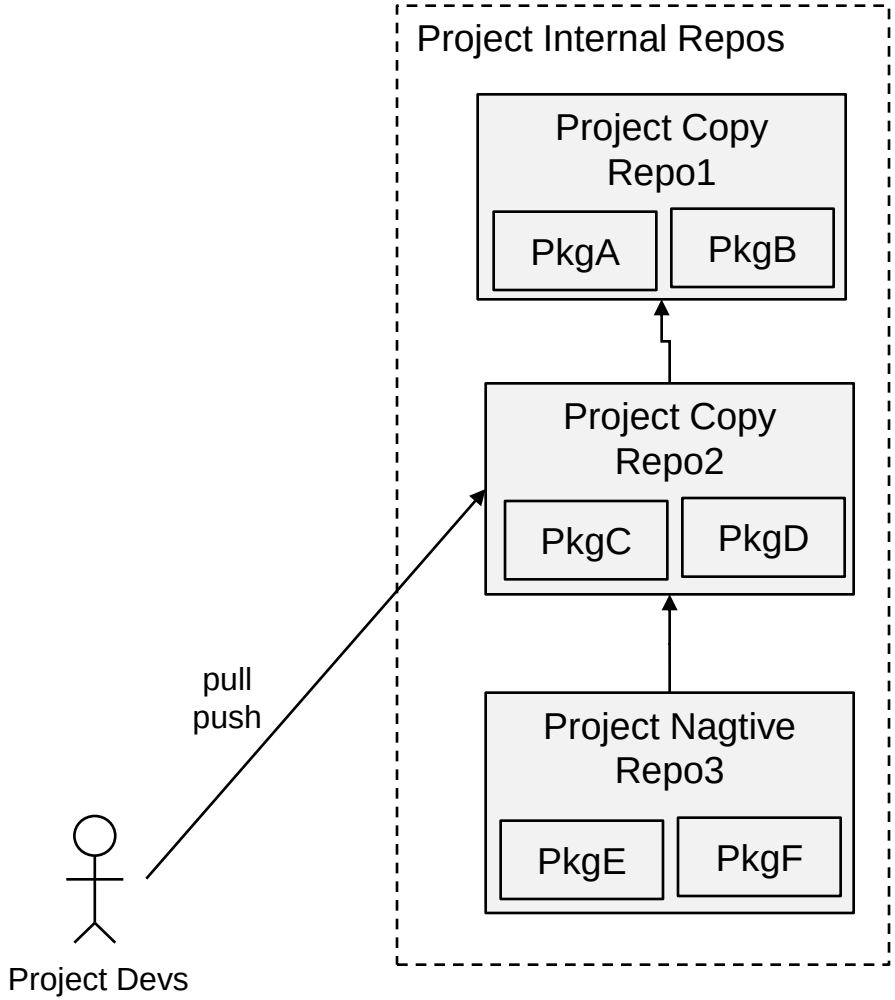
VERA Examples: COBRA-TF (current), TeuchosWrapersExt

Internally managed repo

- Simple single-repo development model

Notes:

- Good when you don't need to coordinate with developers outside of Project



VERA Examples: VERAInExt, PSSDriversExt

Miscellaneous Issues and Topics for TriBITS and VERA

Git Snapshot Repos and Local Project Changes

Basic Concepts:

- The external native repos does not use git
- A git copy of the native repo is created and “snapshot” commits are created.
- Each git snapshot commit must contain info about the version of the repo from the native repo to track versions
- Local changes can be made on a local git branch and merged with snapshot branch.

Example:

```
$ eg log
commit 8423c41af901082a0b05a31cbf2b15363fc09773 (master~1)
Author: Kevin Clarno <clarnokt@ornl.gov>
Date:   Wed Oct 30 10:29:46 2013 -0400

    changeset:   9909:ba36380b3a92
    tag:         tip
    user:        Ugur Mertyurek <u2m@ornl.gov>
    date:        Wed Oct 30 10:13:11 2013 -0400
    files:       src/bonami/CMakeLists.txt src/bonamiM/BonamiData.cpp
    description:
    case:3312 Fixed DBC syntax error on bonamiData commented header information in
bonami cmakelist to preven possible install error
```

Git Snapshot Repos and Local Changes : MOOSE and SVN

Branches in casl-dev/MOOSE.git repo:

- inl_clean_svn: Direct snaphot commits for MOOSE SVN repo
- master: Local changes and merges from inl_clean_svn

Updating snapshot:

```
$ cd MOOSE
$ eg fetch origin
$ eg checkout -b inl_clean_svn origin/inl_clean_svn  # tracking branch, once
$ ./create_snapshot_commit   # update from current SVN repo
$ eg push                    # push to origin/inl_clean_svn
$ eg checkout master
$ eg pull
$ eg merge inl_clean_svn     # hope for no merge conflicts!
$ eg push                    # push to origin/master (TEST FIRST!)
```

Example:

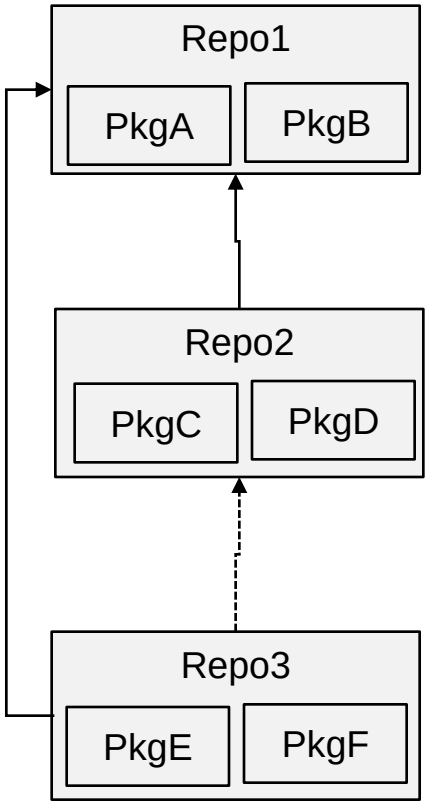
```
commit bacba1ed219d9fba4a50132a3173efcf3f469d18 (master~2^2~1^2)
Author: moosetest <moosetest@dd8cd9ef-2931-0410-98ca-75ad22d19dd1>
Date:   Tue May 28 15:03:57 2013 +0000

r18996 | permcj | 2013-05-28 08:46:46 -0600 (Tue, 28 May 2013) | 1 line
   Holy Warnings Batman! refs-#1777
```

Incorporating Externally Configured/Built Software

- **Motivation**: For some software, it may not be practical or maintainable to create a (secondary) native TriBITS build for a piece of software.
- **Goal:**
  - Use another configure/build tool for external software.
  - Put in CMake/TriBITS hooks to incorporate into TriBITS build with other packages.
- **Easy case:** No upstream TriBITS packages => ***Repo1***
- **Medium case:** No downstream TriBITS packages => ***Repo3***
- **Hard case:** Both upstream and downstream TriBITS packages => ***Repo2***
- **Example:** INL MOOSE developers not willing to support a native TriBITS build of libmesh and MOOSE/Peregrine for CASL VERA. Libmesh depends on DataTransferKit and therefore Trilinos ☹

  Tpetra <= DataTransferKit <= libmesh <= MOOSE <= VRIPSS
- **Technical challenges in TriBITS:**
  - Generate export makefile for upstream Trilinos packages
  - Create CMake rules to produce libs/executables
  - Add dependencies for changes to upstream code
  - Add dependencies for modified external project files



Incorporating Externally Configured/Built Software: Example

```
# TribitsExampleProject/packages/wrap_external/CMakeLists.txt

TRIBITS_PACKAGE(WrapExternal)

…

# A) Write the export makefile that will be used by the external project
TRIBITS_WRITE_FLEXIBLE_PACKAGE_CLIENT_EXPORT_FILES(
  PACKAGE_NAME ${PACKAGE_NAME}
  EXPORT_FILE_VAR_PREFIX TribitsExProj
  WRITE_EXPORT_MAKLEFILE "${EXPORT_MAKKEFILE}"
  )

# B) Run external configure
EXECUTE_PROCESS(
  COMMAND ${PYTHON_EXECUTABLE} ${EXTERNAL_FUNC_SOURCE_DIR}/configure.py
    --with-export-makefile=${EXPORT_MAKKEFILE}
    --src-dir=${EXTERNAL_FUNC_SOURCE_DIR}
    --build-dir=${EXTERNAL_FUNC_BINARY_DIR}
  )

# C) Define a custom build rule and target to create exteranl_func library
ADD_CUSTOM_COMMAND(
  OUTPUT ${EXTERNAL_FUNC_LIB_FILE}
  DEPENDS ${EXTERNAL_FUNC_SOURCE_DIR}/external_func.hpp
    ${EXTERNAL_FUNC_SOURCE_DIR}/external_func.cpp
  COMMAND make
  WORKING_DIRECTORY ${EXTERNAL_FUNC_BINARY_DIR}
  )
ADD_CUSTOM_TARGET( build_external_func
  DEPENDS ${EXTERNAL_FUNC_LIB_FILE} )
```

- Contains dependencies for external project files!

Incorporating Externally Configured/Built Software: Example

```
# TribitsExampleProject/packages/wrap_external/CMakeLists.txt

# continued …

# D) Add the imported library with TRIBITS_ADD_LIBRARY( ... IMPORTED ...)
#
# Below, I just manually do what TRIBITS_ADD_LIBRARY() would do automatically.

# D.1) Create an imported library target and set up the depenancies
ADD_LIBRARY(exteranl_func STATIC IMPORTED GLBOAL) # GLOBAL
SET_PROPERTY(TARGET exteranl_func PROPERTY IMPORTED_LOCATION ${EXTERNAL_FUNC_LIB_FILE})
ADD_DEPENDENCIES(build_external_func pws_c)                # Upstream TriBITS libs pws_s
ADD_DEPENDENCIES(exteranl_func build_external_func)
GLOBAL_SET(exteranl_func_IMPORTLIB_TARGET build_external_func)

# D.2) Update the TriBITS varaibles
APPEND_SET(${PACKAGE_NAME}_LIB_TARGETS exteranl_func)
GLOBAL_SET(${PACKAGE_NAME}_LIBRARIES exteranl_func pws_c)     # Upstream TriBITS libs pws_s
GLOBAL_SET(${PACKAGE_NAME}_INCLUDE_DIRS ${EXTERNAL_FUNC_SOURCE_DIR})
GLOBAL_SET(${PACKAGE_NAME}_HAS_NATIVE_LIBRARIES ON)
INCLUDE_DIRECTORIES(${EXTERNAL_FUNC_SOURCE_DIR})

# E) Add an executable and test to show that it works!
TRIBITS_ADD_EXECUTABLE_AND_TEST(run_external_func
  SOURCES run_external_func.cpp
  DEPLIBS exteranl_func
  PASS_REGULAR_EXPRESSION "external_func C B A"
  )

TRIBITS_PACKAGE_POSTPROCESS()
```

What is missing?

- No rebuild when upstream header files (or Fortran Modules) change
- No rebuild when compiler options change (unless external build handles this)
- Does not work with export makefiles.

TriBITS wrapper for MOOSE is MUCH worse!

Changing Integration Models for Trilinos in CASL VERA

- Integration Models with Trilinos have changed in VERA over 3+ years

- Integration phases for Trilinos in VERA phases:
  1. Push all changes to Trilinos (including TriBITS) directly to ssg/master:
     1. Run VERA CI and Nightly testing directly against Trilinos on ssg/master
     2. Run a sync server to test Trilinos against Denovo and push to casl-dev/master if all tests pass, run on loops of 10 minutes between iter.
     3. " … ", adding more packages, run on loops of 3 hours
  2. Push changes to TriBITS under Trilinos directly to casl-dev/master
     1. Turn off sync server from from ssg/master to casl-dev/master. Manually push changes back and forth as needed. **[CURRENT]**
     2. Run sync server to push TriBITS changes from casl-dev/master to ssg/master on a daily basis.
     3. Run sync server to push general Trilinos changes form ssg/master to ssg/master on a weekly basis (but tested every day flagging regressions).

VERA Development Environment Installation

- VERA Test Stands and RSICC release require a specific build environment:

```
common_tools/
  autoconf-2.69/
  cmake-2.8.5/
  git-1.7.0.4/
  eg
  egidst
gcc-4.6.1/
  toolset/
    gcc-4.6.1/
    openmpi-1.4.3/
  tpls/
    opt/
      common/
        lapack-3.3.1/
        boost-1.49.0/
        zlib-1.2.5-patched/
        moab-4.5.0/
        hypre-2.8.0b/
        petsc-3.3-p4/
      vera_cs/
        hdf5-1.8.7/
        silo-4.8/
        qt-4.8.2/
      ...
```

VERAInstallationGuide.[rst,html,pdf] describe:

- Install scripts in tribits/python download tarballs for common_tools, GCC, and OpenMPI and build/install from source.
- Scripts and source in casl_tpls.svn repo configure/build/install all of the TPLs.
- Standard configurations of VERA set up to automatically work with the Standard VERA Dev Env.

Future TriBITS Development

TriBITS Important Issues and Missing Features

- Configure is very slow for many packages (> 100)
  - Unit and verification testing needed before any optimization work
  - Part of TriBITS cmake code is O(n^2) in worst case w.r.t. n = #packages
  - Make debug-checks optional, etc.
- Need greater flexibility in configuring/building/installing pieces of a large TriBITS meta-project in chunks

  $\Rightarrow$ Meld together TriBITS concepts of Packages and TPLs
- Documentation for TriBITS:
  - High-level overivew of TriBITS (a value statement) (30 pages?)
  - TriBITS project developers documentation (200+ pages?)
  - TribitsExampleProject (and similar example TriBITS projects)
- Official and correct support for externally configured/built software (e.g. MOOSE)
- Lots of small issues (but important when all added up).

CASL Milestones related to TriBITS Development:

- 6/30/2014: “TriBITS Documentation, Performance Optimization and Externalization”
  - Complete initial drafts of TriBITS documentation & examples
  - Address immediate performance concerns
  - Put primary TriBITS dev repo on public Github repo

TriBITS Development and Contributions

```
$ eg shortlog -ns --since=2011-11-05 --before=2013-11-05 -- cmake/tribits/
   309  Roscoe A. Bartlett
    39  Will Dicharry
    25  Brent Perschbacher
    13  Christopher G. Baker
     5  Jeremie Gaidamour
     5  Nico Schlömer
     2  James M. Willenbring
     1  Bill Spotz
     …
$ eg log-oneline --since=2011-11-05 --before=2013-11-05 -- cmake/tribits/ | wc -l
405
```

- Majority of TriBITS development (> 75%) being done to support multi-repo projects like CASL.
- Very little TriBITS development being done for Trilinos

- What development, integration, and deployment model for TriBITS makes sense going forward?

Summary

- CASL VERA development has pushed multi-repo support in TriBITS

- Major remaining issues to be resolved in TriBITS:
  - Configure times/scalability for large numbers of TriBITS packages
  - Combining concepts of packages and TPLs for large meta-projects
  - Documentation

- But once these are done => TriBITS will be a good candidate for a universal meta-build and installation system for a large amount of CSE software.

- What development, integration, and deployment model for TriBITS makes sense going forward?

THE END