Managed by Fermi Research Alliance, LLC for the U.S. Department of Energy Office of Science

How to develop in LArSoft

Saba Sehrish

art/LArSoft Course

(08/03/15-08/07/15)

Goals

- Learn about
  - `gitflow`
  - how to contribute to LArSoft repositories
    - New code, modify existing code
  - how to add tests for the new and modified code

What you know so far

- Initial setup for the working environment
  - `ups, mrb, git, gitflow`
- Create a working area
  - `mkdir <workingdir>; cd <workingdir>`
  - `export MRB_PROJECT=larsoft`
  - `mrb newDev -v<> -q<>`
  - `source <localproducts>/setup; setup larsoft …`
- Check out, build and install
  - `cd $MRB_SOURCE`
  - `mrb g <lar repo>`
  - `cd $MRB_BUILDDIR`
  - `mrbsetenv; mrb build -jN; mrb install -jN`

What you would like to do next

- Make changes to the code you checked out
- Add new code to any of LArSoft repositories
- Build and test your code
- Make it available
  - Merge
  - Publish

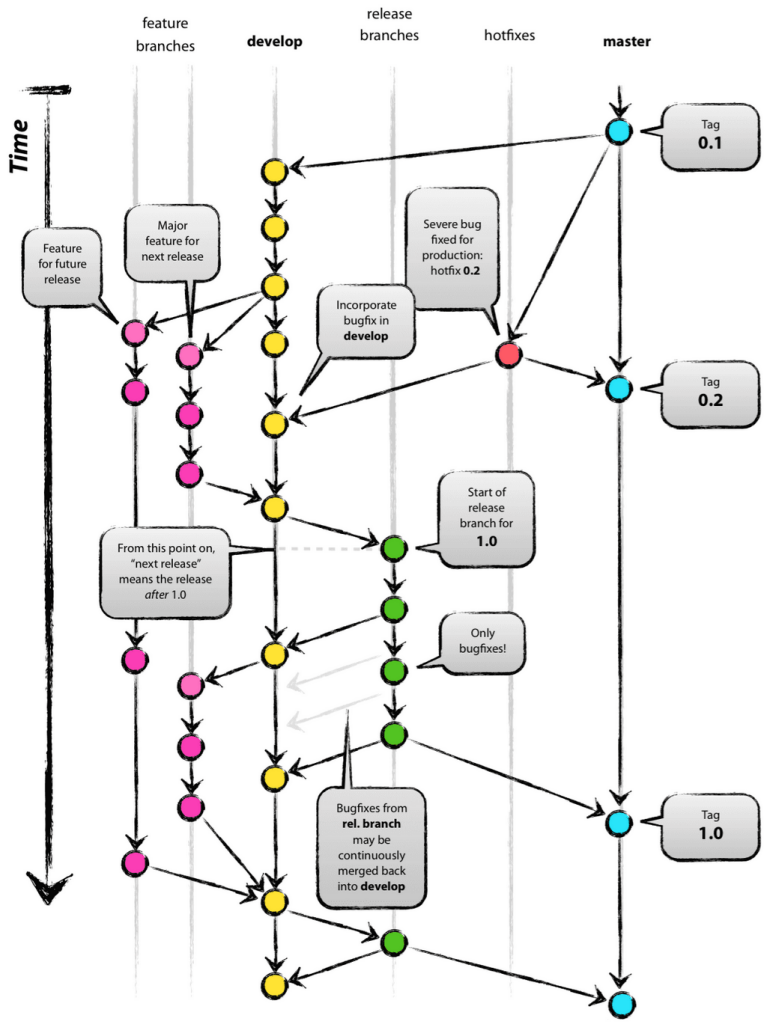
Using gitflow – branch model used by LArSoft

Main branches

- A develop branch that will have the working head of the repository.
  - Used by all developers.
- A master branch that will have only tagged releases.
  - Used only by the software manager.

Supporting branches

- An arbitrary set of feature branches for on-going development.
  - In most cases, these branches will be in local repositories, although "publishing" them to the central repository is allowed whenever needed
- A release branch for the integration of specific tagged releases.
  - Used or authorized only by the software manager.
- A hotfix branch is used to develop patches to tagged releases.
  - By software manager



http://nvie.com/posts/a-successful-git-branching-model/

Modifying existing code

- Assuming `$MRB_SOURCE/<lar repo>` has the checked out code, create a new feature branch e.g. `larexamples`
  - `cd $MRB_SOURCE/larexamples`
  - `git flow feature start ${USER}_testFeature`

```
-bash-4.1$ git flow feature start ss_testFeature
Switched to a new branch 'feature/ss_testFeature'

Summary of actions:
- A new branch 'feature/ss_testFeature' was created, based on 'develop'
- You are now on branch 'feature/ss_testFeature'

Now, start committing on your feature. When done, use:

     git flow feature finish ss_testFeature
```

  - `git branch -a`
    - It will show all branches including the current one you just created.
  - `git branch`
    - Will just show local branches

```
-bash-4.1$ git branch -a
develop
* feature/ss_testFeature
  master
…
remotes/origin/HEAD -> origin/master
  remotes/origin/develop
  remotes/origin/master
…
```

Adding a new package

- From previous lecture; `$MRB_SOURCE/<lar repo>` has the checked out code, create a new feature branch
  - `cd $MRB_SOURCE/<lar repo>`
  - `git flow feature start ${USER}_testFeature`
  - `git branch -a`
- Create a new package inside a repository
  - `mkdir <pkg_dir>`
  - Modify/Edit `CMakeLists.txt`
    - `add_subdirectory()`
  - `cd <pkg_dir>`
    - Create a new `CMakeLists.txt`
    - Create new files/sources
- Can create more sub directories
  - Follow the same procedure

- You can now make all of your changes
- You will need to create a new branch like this for every repository/package in which you are changing code!
  - Do not change code in “develop” branch!

Using `mrb newDev -uv`

What to do when there is a new release

- Start in a new window (make sure UPS is setup)
- To check the list of available larsoft releases:

```
ups list -aK+ larsoft
```

- Make a new local products area for this release:

```
cd <working_dir>
mrb newDev -p -v <new version> -q <qualifiers>
```

  - this creates a new `localProducts_<new version>` directory using the new larsoft release and the existing `srcs` directory

```
source localProducts_<version>/setup
```

- Make sure to use the new `localProducts` directory,
  - delete the old directory

Update code when there is a new release

Now deal with the code:

```
cd $MRB_SOURCE/<lar repo>
git checkout develop
git pull
```

If you are working on a feature branch:

```
cd $MRB_SOURCE/<lar repo>
git checkout develop
git pull
git checkout feature/${USER}_featureName
git merge develop
```

- Repeat for each repository in `$MRB_SOURCE`

```
mrb uv larsoft <new version>
```

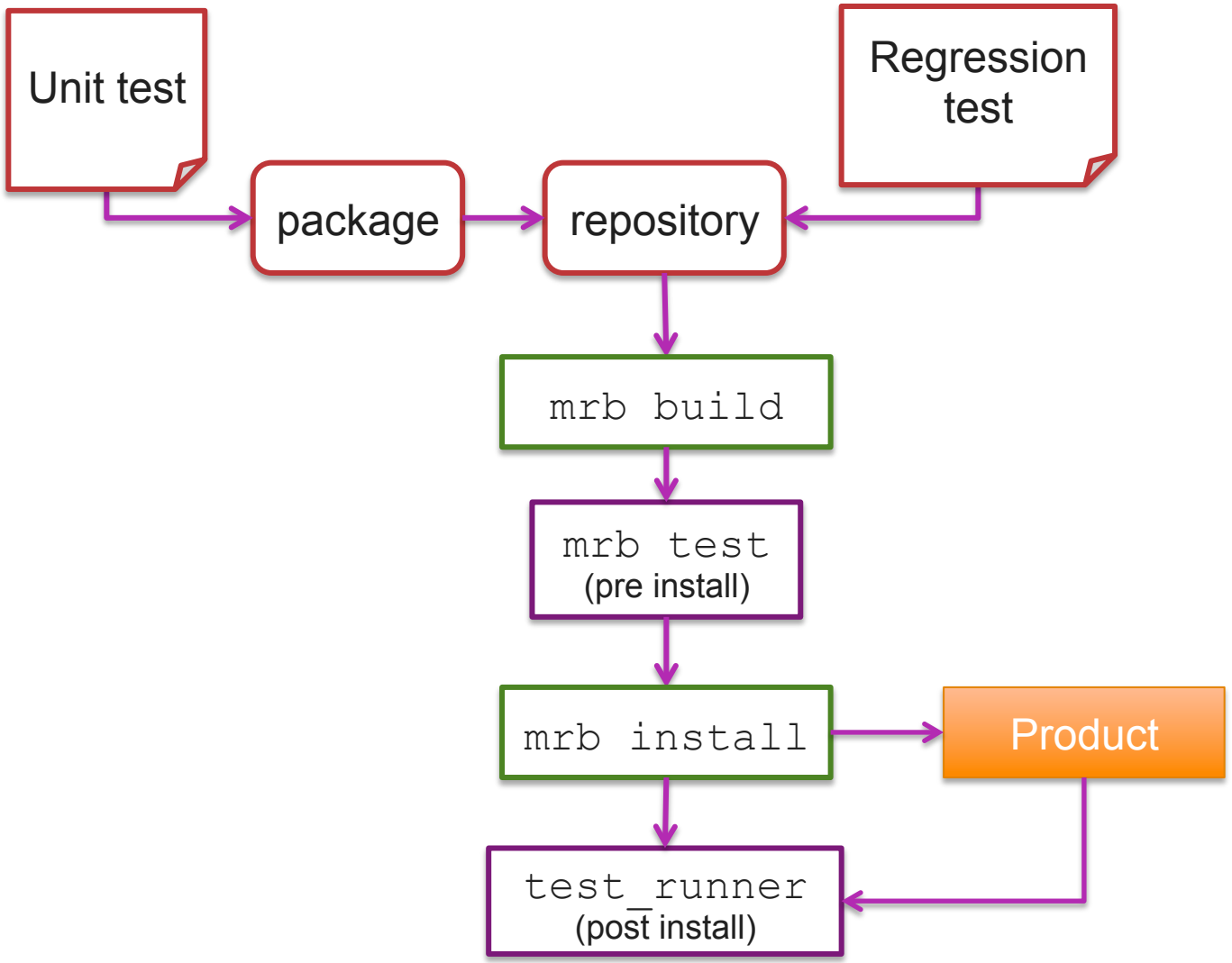
- Resolve any conflicts and make a clean build

Supply tests for your Code

- You have to write/run tests to make sure
  - that your code works! (it does what it was programmed to do and it produces expected results)
  - that your code hasn't broken any other functionality
  - to catch problems caused by later changes to the code (Chris J)
- You are encouraged to
  - Write tests in the test directory for every piece of your code
    - Unit tests/Regression tests
    - Add your test using `cet_test` macro to `CMakeLists.txt` e.g.
      - `include(CetTest)`
      - `cet_test(OpFlashAlg_test… )`

      https://cdcvs.fnal.gov/redmine/projects/cetbuildtools/repository/revisions/master/entry/Modules/CetTest.cmake
  - `cd $MRB_BUILDDIR`
  - `mrb test -jN`
    - build and then run tests specified by `cet_test` inside a `CMakeLists.txt`
  - `setup lar_ci; test_runner <test/test suite name>`

Sequence of test operations



lar_ci test_runner

- The `test_runnner` is part of the `lar_ci` package, and its goals are to make it easy to:
  - run tests
  - add new tests
  - create suites of tests
- The test runner is driven by config files, e.g. `test/ci/ci_tests.cfg`
  - test_runner will read this file to help it find tests and test suites
  - test/CMakeLists.txt
    - `install_scripts(AS_TEST)`
      - Will install all unit tests and `ci_tests.cfg` and all binaries in `test/` that run those tests and suites of tests into `localProductsXYZ/product/<version>/test/`. The CI system will find them there.

`ci_tests.cfg`

- Write stand alone test scripts
  - Pass/fail output
- Either write a new or modify the `ci_tests.cfg` file
  - Add test definition block for each of the stand alone scripts
    - `[test my_test]`
    - `script= …`
    - `args= …`
  - Running individual tests can lead to long command lines, so we recommend instead creating a test suite.
    - `[suite my_suite]`
- Now you can run `test_runner my_suite`
- See the following link for more information

  https://cdcvs.fnal.gov/redmine/projects/lar-ci/wiki/Test_Runner_Introduction

Multiplatform continuous integration for LArSoft

Multiplatform continuous integration for LarSoft

Select builds:

From build:

# of builds:

Select platforms:

- Darwin 13.4.0
- Darwin 14.3.0
- faketest
- Linux 2.6.18-371.11.1.el5
- Linux 2.6.18-371.12.1.el5
- Linux 2.6.18-400.1.1.el5
- Linux 2.6.32-431.20.3.el6.x86_64
- Linux 2.6.32-431.23.3.el6.x86_64
- Linux 2.6.32-431.29.2.el6.x86_64
- Linux 2.6.32-504.1.3.el6.x86_64
- Linux 2.6.32-504.23.4.el6.x86_64
- Linux 2.6.32-504.3.3.el6.x86_64
- Linux 2.6.32-504.8.1.el6.x86_64
- Linux 2.6.32-504.el6.x86_64

Select test suites:

- default

Select date range:

From:

To:

Update

Build	Start Time	Platform	checkout	build	make_test	install	ci_tests
lar_ci_beta/1230	2015-08-05 13:50:44.649416	Linux 2.6.32-504.23.4.el6.x86_64					
	Unknown	Unknown					
lar_ci_beta/1229	2015-08-04 13:15:11.329521	Linux 2.6.32-504.23.4.el6.x86_64					
	Unknown	Unknown					
lar_ci_beta/1228	2015-08-03 19:45:45.048902	Linux 2.6.32-504.23.4.el6.x86_64					
	Unknown	Unknown					
lar_ci_beta/1227	2015-08-03 15:49:53.355631	Linux 2.6.32-504.23.4.el6.x86_64					
	Unknown	Unknown					
lar_ci_beta/1226	2015-08-02 22:51:13.629634	Linux 2.6.32-504.23.4.el6.x86_64					
	Unknown	Unknown					
lar_ci_beta/1225	Unknown	Unknown					
	2015-07-31 18:15:14.270565	Linux 2.6.32-504.23.4.el6.x86_64					
lar_ci_beta/1224	Unknown	Unknown					
	2015-07-31 16:23:03.439165	Linux 2.6.32-504.23.4.el6.x86_64					
lar_ci_beta/1223	2015-07-30 23:36:59.261071	Linux 2.6.32-504.23.4.el6.x86_64					
	Unknown	Unknown					
lar_ci_beta/1222	2015-07-30 18:24:33.416596	Linux 2.6.32-504.23.4.el6.x86_64					
	Unknown	Unknown					

Progress Legend
Running
Pending
Succeeded
Failed
Skipped

Fermilab

Add new code to LArSoft

- Make changes and commit to feature branch
  - Create a new file `my_file.cc`, or make changes
  - Commit this change
    - Add the file first if it hasn't already been added to the repository

      `git add my_file.cc`
    - Commit

      `git commit -m` "commit message"

      if you do not use -m, it will open a text editor and allow you to make a very long commit message
  - Add multiple files

    `git add file1.cc file2.cc`
  - Add a directory

    `git add my_dir`

Things to remember!

- Use `git status` frequently
- Keep your feature branch up-to-date with the develop branch
  - Make frequent commits
  - Update and rebase before pushing the commits

    ```
    cd $MRB_SOURCE/<lar repo>
    git fetch origin
    git rebase origin/develop
    ```

    Fix rebase conflicts, continue rebase

    ```
    git rebase --continue
    ```

    Commit those changes
- You can undo git rebase
  - `git rebase --abort`

Merge, declare your changes

- Merging your changes back into the repository

```
cd $MRB_SOURCE/<lar repo>/<package>
git fetch origin
git rebase origin/develop
git flow feature finish
```

  - That will merge your feature branch back into ***your*** develop branch and delete your feature branch (remember, you have your own repository!).

  – Push your changes in develop to the main (origin) repository:

```
git push origin develop
```

- In LArSoft world you need to follow the guidelines on next slide before you do so

  – To publish a feature branch for collaboration (before feature finish):

```
git flow feature publish ${USER}_testFeature
```

  You can not rebase after publishing

Managing changes to develop branch

- Most changes are coordinated through bi-weekly coordination meeting to
  - make sure there are no conflicts
  - make everyone aware of changes and behavior
  - make sure there is no breaking changes
    - Never merge a breaking change into develop!!!
    - Changes are merged by Software manager during release process
      - Makes sure develop always works
  - discuss any new code
- Some changes can be merged without discussion
  - Bug fixes, new code that nothing uses or depends up on
  - Other changes that have been agreed to on some other forums
- However it is a general practice to have a presentation of your code to be merged at the coordination meeting.

Summary

- Branch model used by LArSoft
- `gitflow`
- Use feature branch
  - to modify existing code
  - add new code
- Testing for your code