# LArSoft technical details

Saba Sehrish, *Fermilab*
on behalf of SciSoft Team
LArSoft 2019 Summer Workshop

# Outline

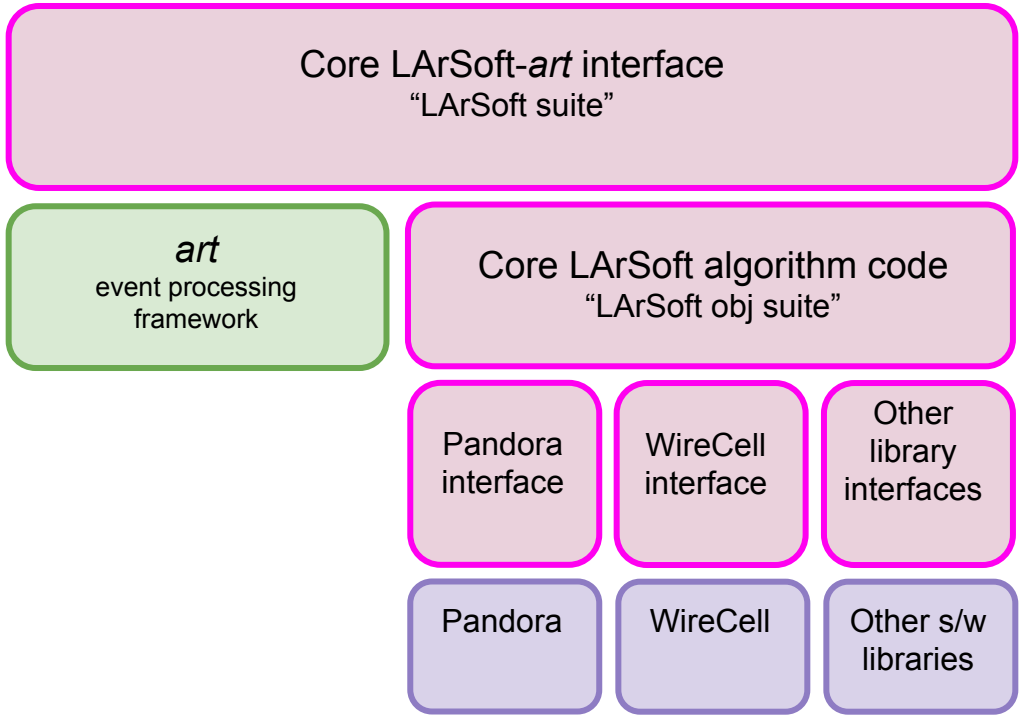
LArSoft repositories

LArSoft products

Setting up and running LArSoft

Contributing to LArSoft

# LArSoft conceptual design




There are 18 repositories containing LArSoft code.

# LArSoft repositories

The LArSoft code is organized into 18 different repositories that can be loosely grouped into three categories as shown in the conceptual design.

- Core LArSoft-*art* interface repositories
  - Modules, services, tools
- Core LArSoft algorithm repositories
  - Algorithms, providers
- Repositories with interface code to external software

In addition to these three types, every experiment has at least one code repository.

# LArSoft repositories

The LArSoft code is organized into 18 different repositories that can be loosely grouped into three categories as shown in the conceptual design.

- Core LArSoft *art* interface repositories
  - Mo
- Core LA
  - Algorithms, providers
- Repositories with interface code to external software

You will be using and contributing code to at least one of these repositories.

In addition to these three types, every experiment has at least one code repository.

# Core LArSoft repositories

| Name | Description |
|---|---|
| larcore | Low level utilities and functions e.g. Geometry services |
| lardata | Data products and other common data structures |
| larevt | Low level algorithm code that use data products |
| lareventdisplay | LArSoft based event display |
| larsim | Simulation code |
| larreco | Primary reconstruction |
| larana | Secondary reconstruction/analysis e.g. PID |
| larexamples | Examples of writing algorithms, data products, etc. |
| larsoft | Top-level repository |

# Interface code repositories

| Name | Description |
|---|---|
| larpandora | LArSoft interface to the pandora reconstruction package, includes *art* modules, etc |
| larwirecell | Interface to wirecell, includes *art* modules, etc |
| larpandoracontent | Algorithms and tools for larpandora |
| larg4 | Based on artg4tk, includes modules and services for Geant 4 |

# Core LArSoft algorithm repositories

| Name | Description |
|---|---|
| lardataalg | Algorithms shared between larsoft and gallery, larlite, etc. |
| lardataobj | Common data products for reconstruction, analysis, etc shared between larsoft and gallery, larlite, etc. |
| larcorealg | Core algorithms shared between larsoft and gallery, larlite, etc. |
| larcoreobj | Common data products for reconstruction, analysis, etc shared between larsoft and gallery, larlite, etc. |
| larsoftobj | Umbrella package/repository |

# LArSoft Products

# ups products

The build procedure creates and installs a **ups product** from the code in each repository.

What is **ups** (unix product support)?
ups is a tool that allows multiple concurrent versions of software libraries / products to co-exist on a single machine, and switching between them as needed

What is a **ups product**?
Collection of software, libraries, configuration files..., that define a single instance
Each product is self-contained, aside from dependencies

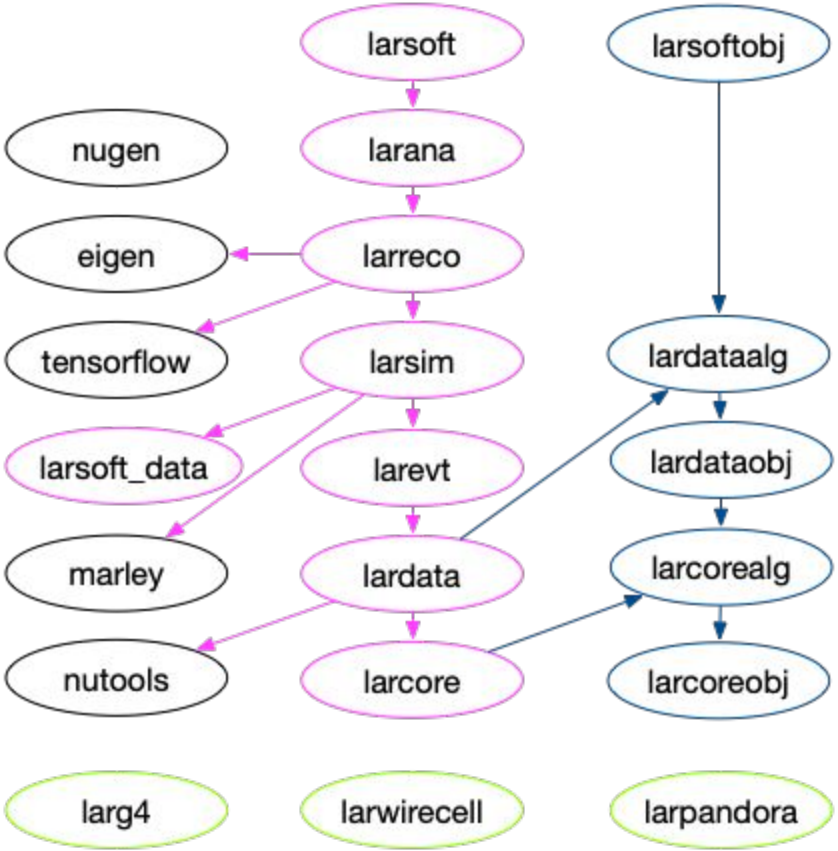
What is ups **setup** command?
Selects a single instance to use by **defining a set of environment variables** that point to the relevant software / libraries. e.g., <product>_DIR, <product>_INC, <product>_LIB, etc...
The “setup” command also **performs “setup” for any required dependencies**

```
setup -B <product name> <version> -q <qualifiers>
```

# larsoft ups products

- A LArSoft “release” is a consistent set of LArSoft products built from tagged versions of code in the repositories
  - Implicitly includes corresponding versions of all external dependencies used to build it.
- larsoftobj
  - An umbrella product for the larsoft algorithm repositories
  - Setting up larsoftobj sets up all the obj products and other dependencies: setup -B larsoftobj v08_15_00 -q …
- larsoft_data
  - A ups product (not a repository)
  - A place for large configuration files
- larsoft
  - A “larsoft” umbrella product binds it all together to give it one version, one command: setup -B larsoft v08_22_00 -q …
  - The only thing needed to run LArSoft is access to a tagged release
  - There is no need to checkout any code and build it

# Dependencies among larsoft products - simplified version



# Running LArSoft

# setup larsoft ups product

- First setup the ups product

  ```
  source <ups products dir>/setup
  ```

  - Experiments will have their own setup scripts, so users normally don't see this
- Then setup larsoft
  - `setup -B larsoft v08_22_00 -q +e17:+prof`
  - **Now you can use the lar command!**
- Some other useful ups commands are
  - `ups list -aK+ <product name>`
    - Lists available versions of the given product
  - `ups active`
    - Lists all the products that are currently setup
  - `ups depend <product name> -q <qualifiers>`
    - List of products dependencies (product doesn't need to be setup for that)
    - `ups depend larsoft v08_22_00 -q e17:prof`

# setup larsoft ups product

- First setup the ups product

  `source <ups products dir>/setup`

  - Experiments will have their own setup scripts, so users normally don't see this
- Then setup larsoft
  - `setup -B larsoft v08_22_00 -q +e17:+prof`
  - **Now you can use the lar command!**
- Some other useful ups commands are
  - `ups list -aK+ <product name>`
    - Lists available versions of the given product
  - `ups active`
    - Lists all the products that are currently setup
  - `ups depend <product name> -q <qualifiers>`
    - List of products dependencies (product doesn't need to be setup for that)
    - `ups depend larsoft v08_22_00 -q e17:prof`

Built with GCC v7.3.0, -std=c++17, -std=gnu (gfortran)
https://cdcvs.fnal.gov/redmine/projects/cet-is-public/wiki/AboutQualifiers#Primary-qualifiers

# The lar command

- An alias to *art* - allows LArSoft-customized build and configuration
- Get help: `lar -h`

```
lar ... -n <num events> -c <fcl configuration> -s <input art/ROOT>
```

- You need to provide a configuration file, you can use any installed fcl file or you can use your own fcl file and input root file.

# The lar runtime configuration

- How does *art* find the fcl file?

  FHICL_FILE_PATH environment variable: path to FHiCL directories defined by the ups products that are setup.
- How do I examine final parameter values for a given fcl file?
  - `fhicl-expand`

    Performs all “#include” directives, creates a single output with the result
  - `fhicl-dump`
    - Parses the entire file hierarchy, **prints the final state** all FHiCL parameters
    - **Using the “--annotate” option, also lists the fcl file + line number** at which each parameter takes its final value
    - Requires FHICL_FILE_PATH to be defined
- How do I tell the FHiCL parameter values for a processed file?
  - `config_dumper`

    Prints the full configuration for the processes that created the file

# The lar runtime configuration

- Information on configuration
- Best practices and guidelines explained in presentation by Kyle Knoepfel
  - Presentation from 2016 LArSoft Workshop
  - Not things that the typical user needs to know, but…
    - ...helps to answer why things are this way
  - It is required information for people who write modules or production workflows
    - E.g., fcl validation features
  - Basically calls for **highly nested structures that layer overrides**

- Bottom line: need good tools to help validate and debug

# Contributing code to LArSoft

# Where to find larsoft code?

LArSoft code lives in a set of git repositories hosted at Fermilab

All are publically accessible at:
http://cdcvs.fnal.gov/projects/<repositoryname>

For read/write access: ssh://p-<repository name>@cdcvs.fnal.gov/cvs/projects/<repository name>

# Inside a “lar*” repository

- Each repository has a similar organization, .e.g. listing on larreco shows:

```
> ls larreco
larreco
test
ups
CMakeLists.txt
```

For clarity in the include header paths

- Each lar* directory has a number of source code directories called “packages”.
- When a new package is added, the best practice is to add tests for the new code under test/package-name directory.
- If a package directory is in one of the lar* repositories, then it will have modules, services, tools. If it is in one of the larsoftobj repositories, then it will have algorithms code in it.

# Inside CMakeLists.txt

- The file CMakeLists.txt is the file used by the build system (cmake) to learn what steps it should do.
- There is a CMakeLists.txt in every directory/subdirectory; each contains additional instructions for the build system.
- The top level CMakeLists.txt includes
  - minimum version of cmake
  - `project()` name of the project
  - `include()` for additional macros
  - `find_ups_product()` for external dependencies
    - Checks if the product with at least the specified version is setup
  - `add_subdirectory()` for all the subdirectories

# More on CMakeLists.txt

In the CMakeLists.txt of subdirectories

- `simple_plugin` to build modules and services with different set of dependencies
- `art_make` is a utility that invokes simple_plugin on many modules, services, etc and it also makes one shared library
- `cet_test` to specify tests
- Use the following to install headers, fhicl and sources

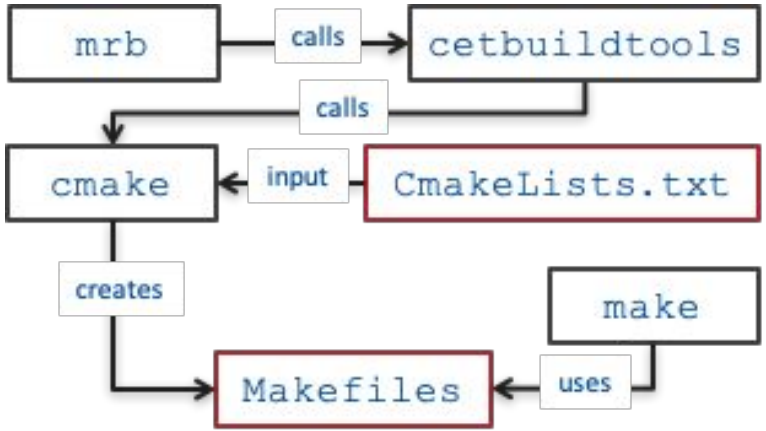
  `install_headers()`

  `install_fhicl()`

  `install_source()`

# Build process with make

- make is the standard build tool that determines dependencies, build order, and issues the commands.
- make uses Makefile(s)for configuration and construction.
- cmake is a tool with a simpler configuration language that will write all of the Makefile(s)for us.
- cetbuildtools are convenience macros for cmake(used by *art* framework).
- mrb for convenience to simplify the building of multiple products pulled from separate repositories.

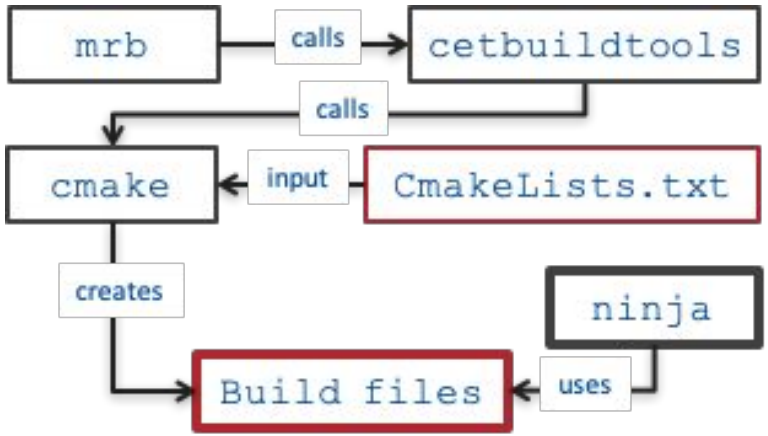





Fermilab

# Build process with ninja

- Ninja is a build system alternative to make.
- ninja works on all platforms.
- The advantage of ninja over make is that if you do an incremental build, ninja can determine what files need compiling in practically zero time.
- Cmake knows how to create the build files for building with ninja.



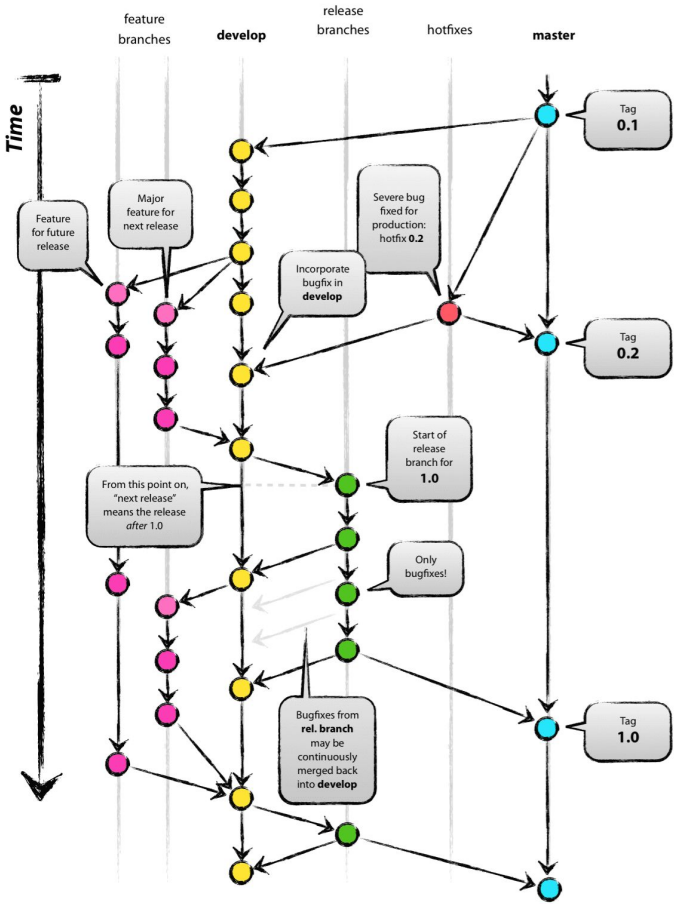
# mrb - multi-repository build system

- The purpose is to simplify the building of multiple products pulled from separate repositories.
- Use ups: `setup mrb`
- Define `MRB_PROJECT` e.g. export `MRB_PROJECT=larsoft`
- `mrb -h` will display a list of all commands that are available with a brief description
- `mrb <command> -h` will display help on a particular mrb command, e.g. `mrb newDev -h` or `setup mrb n -h`

# 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.

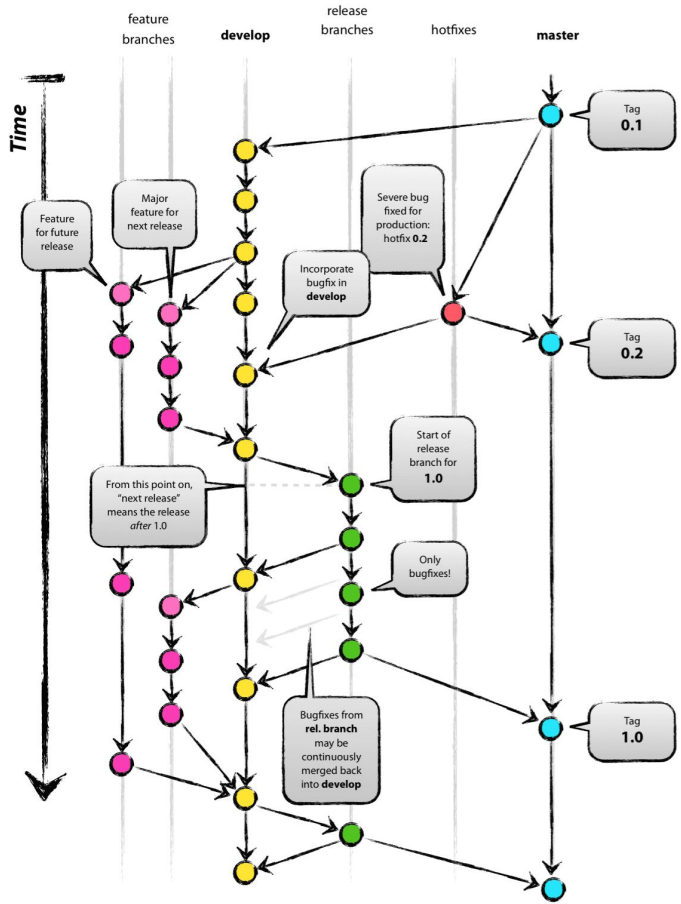


# branch model used by LArSoft

## Supporting branches

- An arbitrary set of feature branches for ongoing 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

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

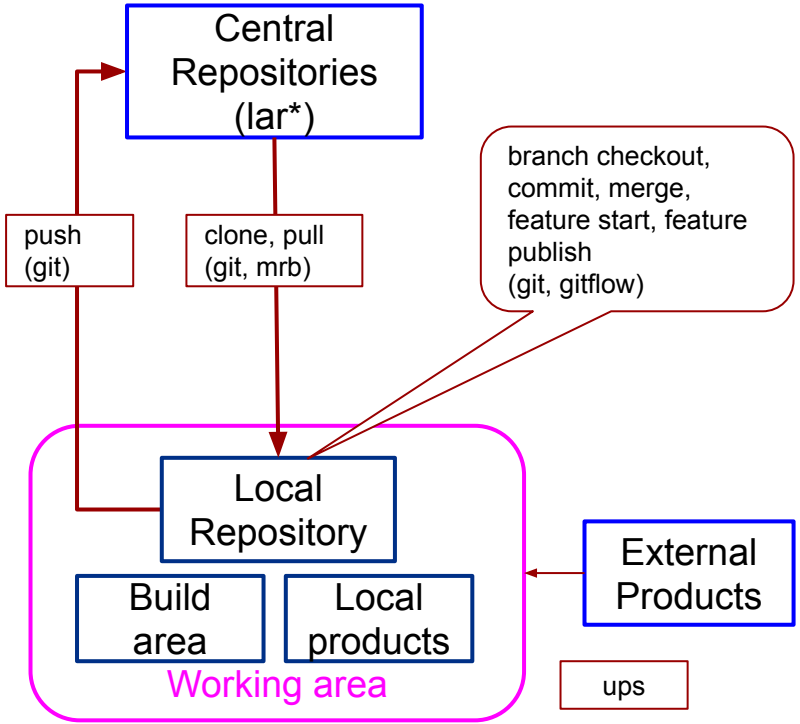


# Using Gitflow for LArSoft

- Gitflow is really just an abstract idea of a Git workflow described earlier.
  - It dictates what kind of branches to set up and how to merge them together.
- The git-flow toolset is an actual command line tool that has an installation process.
  - `gitflow` is provided as a ups product.
- When the command `setup mrb` is executed, `gitflow` gets setup as well.
- LArSoft developers, who will be developing for the project need to work with feature branches of their, can use gitflow to start and publish new features.

https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

# LArSoft development workflow



# Setting up your working area

Starting from a new login shell on a machine with ups products directory, set up the ups environment, and mrb.

- `source <products dir>/setup`
- `setup mrb`
- `mkdir <working dir>; cd <working dir>`
- `export MRB_PROJECT=larsoft`
- Make a new development area by creating srcs, build, and products directories in the `<working dir>`, this is default behavior. -S option can be used to specify source code directory and -T for build and localProducts directory

  `mrb newDev -v vx_x_x -q e17:debug`

# Setting up your working area

Starting from a new login shell on a machine with ups products directory, set up the ups e

- so
- se
- mk
- ex
- Ma

```
[ssehrish@grunt1 larsoft_workshop] mrb newDev -v v08_22_00 -q e17:prof

building development area for larsoft v08_22_00 -q e17:prof

MRB_BUILDDIR is /home/ssehrish/larsoft_workshop/build_slf7.x86_64
MRB_SOURCE is /home/ssehrish/larsoft_workshop/srcs

INFO: copying /products/larsoft/v08_22_00/releaseDB/base_dependency_database

IMPORTANT: You must type
    source
/home/ssehrish/larsoft_workshop/localProducts_larsoft_v08_22_00_e17_prof/setup
NOW and whenever you log in
```

cts
dir
on can be
us
oducts
dir

```
mrb newDev -v vx_x_x -q e17:debug
```

# Setting up your working area

- The following command will define several MRB environment variables and also the PRODUCTS variable

```
source localProducts_larsoft_vx_x_x_e17_debug/setup
```

- An example:

```
MRB_PROJECT=larsoft
MRB_PROJECT_VERSION=v08_20_00
MRB_QUALS=
MRB_TOP=<full-path-to-working_dir>
MRB_SOURCE=<full-path-to-working_dir>/srcs
MRB_BUILDDIR=<full-path-to-working_dir>/build_slf7.x86_64
MRB_INSTALL=<full-path-to-working_dir>/localProducts_larsoft_...
PRODUCTS=<full-path-to-working_dir>/localProducts_larsoft_:/products
```

# Setting up your working area

- The following command will define several MRB environment variables and also the P
  so
- An
  MR
  MR
  MR
  MR
  MR
  MRB_BUILDDIR=<full-path-to-working_dir>/build_slf7.x86_64

```
[ssehrish@grunt1 larsoft_workshop] source localProducts_larsoft_v08_22_00_e17_prof/setup

MRB_PROJECT=larsoft
MRB_PROJECT_VERSION=v08_22_00
MRB_QUALS=e17:prof
MRB_TOP=/home/ssehrish/larsoft_workshop
MRB_SOURCE=/home/ssehrish/larsoft_workshop/srcs
MRB_BUILDDIR=/home/ssehrish/larsoft_workshop/build_slf7.x86_64
MRB_INSTALL=/home/ssehrish/larsoft_workshop/localProducts_larsoft_v08_22_00_e17_prof

PRODUCTS=/home/ssehrish/larsoft_workshop/localProducts_larsoft_v08_22_00_e17_prof:/products
```

```
MRB_INSTALL=<full-path-to-working_dir>/localProducts_larsoft_...
PRODUCTS=<full-path-to-working_dir>/localProducts_larsoft_:/products
```

# Getting the source code

- Any specific repository, or whole suite can be checked out. In the following there are examples of both cases.

```
cd $MRB_SOURCE
```

- If you want to checkout larsoft and larsoftobj

```
mrb g larsoft_suite
mrb g larsoftobj_suite
```

- `mrb g` is the short form of `mrb gitCheckout`.
- Or alternately if you only have to work with one specific repository, .e.g. larreco

```
mrb g larreco
```

# Getting the source code

- Any specific repository, or whole suite can be checked out. In the following there are exa
  `cd $MRB_SO`
- If you want
  `mrb g lar`
  `mrb g lar`
- `mrb g` is the
- Or alternate .e.g. larreco
  `mrb g larre`

```
[ssehrish@grunt1 larsoft_workshop]$ cd $MRB_SOURCE
[ssehrish@grunt1 srcs]$ mrb g larreco
Cloning into 'larreco'...
remote: Counting objects: 59453, done.
remote: Compressing objects: 100% (27850/27850), done.
remote: Total 59453 (delta 44209), reused 43364 (delta 31507)
Receiving objects: 100% (59453/59453), 28.41 MiB | 4.17 MiB/s, done.
Resolving deltas: 100% (44209/44209), done.
Checking out files: 100% (796/796), done.
NOTICE: Adding larreco to CMakeLists.txt file
```

# Setting up the required ups products

Set up the required ups products necessary for building the code: `mrbsetenv`

```
[ssehrish@grunt1 srcs]$ mrbsetenv
The working build directory is
/home/ssehrish/larsoft_workshop/build_slf7.x86_64
The source code directory is /home/ssehrish/larsoft_workshop/srcs
----------- check this block for errors ----------------------
-------------------------------------------------------------
```

# Build the checked out code

- Set up the required ups products necessary for building the code.
  `mrbsetenv`
- Now from the build directory, run the mrb build command.
  `cd $MRB_BUILDDIR`

  `mrb b -jN`, where N is the number of cores you want to use for parallel build
- To use ninja, setup ninja first, e.g. `setup -B ninja <version>`
- Then run the build command
  `mrb b -jN --generator ninja`
- If the build succeeds, run tests, `mrb t -jN`

# Working with feature branches

If you want to add code to larreco or modify any existing code in there, you need to work in a feature branch.

You will need to create a new feature branch for every repository/package in which you are changing code. Do not change code in “develop” branch!

- Change to the correct directory

```
cd $MRB_SOURCE
cd larreco
```

- Start a new feature using git flow

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

- You can see all the feature branches by typing

```
git branch -a
```

- `git branch` will only show the local ones

# Working with feature branches

If you want to add code to larreco or modify any existing code in there, you need to work in a feat

You will nee ackage in which you are cha

- Chang
  `cd $MF`
  `cd lar`
- Start a
  `git fl`
- You ca
  `git bra`
- `git branch` will only show the local ones

```
[ssehrish@grunt1 srcs]$ cd larreco/
[ssehrish@grunt1 larreco]$ git flow feature start ${USER}_testFeature
Switched to a new branch 'feature/ssehrish_testFeature'

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

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

     git flow feature finish ssehrish_testFeature

[ssehrish@grunt1 larreco]$ git branch
  develop
* feature/ssehrish_testFeature
  master
```

# Modifying or adding new code to larsoft

- Create a new package directory `mkdir larreco/<pkg_dir>`
- Update CMakeLists.txt to include the <pkg_dir>
- Make changes and commit to the feature branch
  - Create a new file, e.g. my_file.cc, or make changes to an existing file
- Add the file first if it hasn't already been added to the repository:

  `git add my_file.cc`
- Commit your changes: `git commit -m "commit message"`
  - without -m option, it will open a text editor for a very long commit message
- Add a new directory or multiple files:

  `git add my_dir`

  `git add file1.cc file2.cc`

# Always write tests for your code

It is important to write new tests for your code and run existing 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)

# Building and running tests for your code

- You are encouraged to write tests in the test directory for your code.
  - Add your test using cet_test macro to CMakeLists.txt e.g.

```
include(CetTest)
cet_test(HitAnaAlg_test USE_BOOST_UNIT LIBRARIES larreco_HitFinder)
```

  - build and then run tests

```
cd $MRB_BUILDDIR
mrb test -jN
```

- For running a specific test, you can use `ctest <test name>`
- `ctest -help` lists all the options you can use,
  - `-V` for verbose output
  - `-R` to run tests matching regular expression
- Always test your feature branch for both debug and prof builds

# Making your feature branch public/available

Once your feature branch is ready to be merged into develop:

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

# Returning to your working area from a new login

- First, setup the ups product
  `source <products dir>/setup`
- Then setup mrb
  `setup mrb`
- Change directory to your existing working area
  `cd <working area>`
- This following command is needed to define all the MRB_* environment variables, and the PRODUCTS variable.
  `source local_products/setup`
- Need to setup the development environment
  `mrbsetenv`
- Ready to develop and build again!

# Update your feature branch when there is a new release

- Commit your local changes to your feature branch

```
git commit -am "commit message"
```

- Checkout the head of develop, and make sure you get the updated code

```
git checkout develop
git pull
```

- Then checkout your local feature branch, and merge develop into it

```
git checkout feature/${USER}_testFeature
git merge develop
```

- Do that for all the feature branches in all the repositories you are working with
- Resolve any conflicts and do a clean build

# A few useful commands

`mrb z` : Delete everything in your build area

`mrb zd`: Delete everything in both your build and localProducts areas

`mrb newDev` with -p and -f options:

-f = use a non-empty directory anyway

-p = just make the products area (checks that src, build are already there)

`mrb uc`: Update the master CMakeLists.txt file

`mrb uv`: Update a product version in product_deps

`unsetup_all`: unsetup all the products that were setup

# Recommended policy for adding new code to LArSoft

- Most changes are coordinated through bi-weekly coordination meeting to
  - make everyone aware of changes and behavior
  - make sure there are no conflicts
  - make sure there are no breaking changes
- Never merge a breaking change into develop!!!
- Always use feature branches
- Changes are merged by the release manager during the release process
  - Makes sure develop always works

# Recommended policy for adding new code to LArSoft

- Always discuss any new code
  - Ask questions, ask for help even before writing any code, do design discussions
- Some changes can be merged without discussion
  - Bug fixes, new code that nothing uses or depends upon
  - Other changes that have been agreed to on some other forums
- However it is a recommended practice to have a presentation of your code to be merged at the coordination meeting.

# Questions?