# Introduction to LArSoft under git / mrb / ups

Erica Snider
*Fermilab*

LBNE FD Simulation and Reconstruction Meeting
Fermilab
Jan 31, 2014

# Outline

- Overview
  - The git repositories
  - mrb
  - The working area
  - Building, installing, and products
  - Installed product structure
  - The branching model
- Working with the system
  - A walk through example
  - Examples of a few other common tasks
- Nightly builds

# Documentation

- Primary resource
  - LArSoft redmine wiki :  https://cdcvs.fnal.gov/redmine/projects/larsoft/wiki
    - See user environment and developer environment overviews
    - Quick-start guide (being revised to better delineate example tasks)
    - Information on releases
  - Links to documentation on tools
- Code browsers
  - Redmine browsers (see “Repository” tab on project pages)
  - lxr cross-referencing browser :  http://cdcvs.fnal.gov/lxr
    - Searches will find hits across repositories (though not quite working yet...)

# Introduction

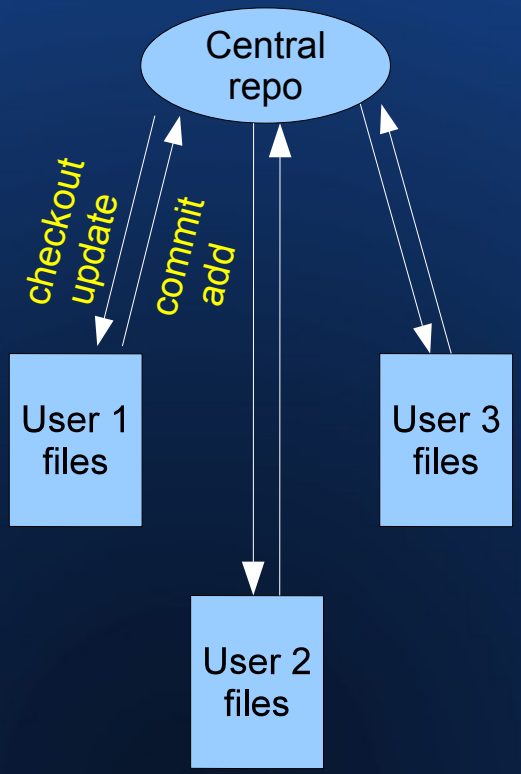
- Goals of the transition
  - Provide the tools necessary to create a more stable development environment
  - Utilize better supported and more modern development and build tools
- Basic strategy
  - Isolate code development from the head of the repository
  - Better control merging of changes into the head
  - Move away from SoftRelTools build system

# Introduction

- Major elements of the new system
  - git repositories for version control (replaces svn)
    - Will also call these “products”, since there is a one-to-one correspondence
  - mrb  (replaces SRT)
    - Creates working area, repository skeletons
    - Drives the build procedure (with cetbuildtools)
      - Configure + cmake + various make phases
  - ups (replaces setup_larsoft_*.sh for user environment)
    - setup <product> <version> -q <qualifiers>
  - git flow (new)
    - A tool that Implements a git development workflow / branching model
    - Using it is optional, but basic branching model should be used

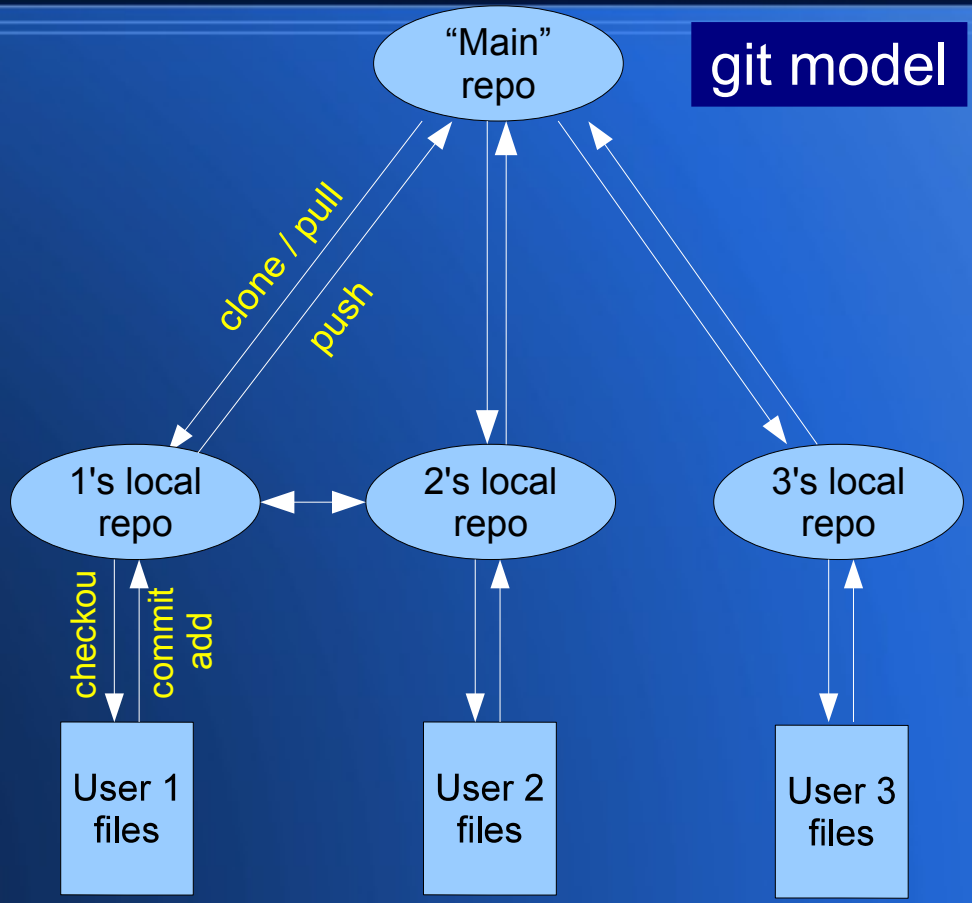
# git vs. svn

## svn model




- Commits shared with everyone instantly
- Conflicts resolved at commit time

## git model




- Create a local copy of the “central” repository
- Commit to local local repository first
- Push commits to central repository to share with everyone
- Conflict resolved at time of local “merge” / “rebase”
- Local repositories:  many operations are very fast

# git repositories

- The svn repository has been factored into smaller git repostories
  - Each contains code at similar level in class hierarchy, functionality
  - The core LArSoft repositories:
    - larcore : Geometry, SummaryData, SimpleTypesAndConstants
    - lardata : data products, utilities, RecoBase, etc.
    - larevt : Filters, CalData
    - larsim : EventGenerator, Simulation, DetSim, LArG4, etc.
    - larreco : RecoAlg, HitFinder, ClusterFinder, *Finder, etc.
    - larana : Calorimetry, OpticalDetector, ParticleIndentification
    - larpandora : Pandora modules and interfaces
    - lareventdisplay
    - larexamples

Full list at https://cdcvs.fnal.gov/redmine/projects/larsoft/wiki/_LArSoft_repositories_packages_and_dependencies_

# git repositories

- Experiment-specific repositories:
  - lbnecode
  - uboonecode
  - lariatsoft
- Where to find the repositories
  - All in Redmine, so url's look like:
    - RW:  ssh://p-<repo_name>@cdcvs.fnal.gov/cvs/projects/<repo_name>
    - Read-only:  http://cdcvs.fnal.gov/projects/<repo_name>
- larsoft_data product
  - Contains data files extracted from svn repositories – needed to run
  - No repository for this

# Structure of repositories

- Internal structure of repositories
  - Core LArSoft repositories.
    - Take larcore as an example:

```
CMakeLists.txt

Geometry/
   CMakeLists.txt
   ...
SimpleTypesAndConstants/
   CMakeLists.txt
   ...
SummaryData/
   CMakeLists.txt
   ...
ups/
   CMakeLists.txt
   product_deps
   ...
.git/
```

cmake configuration files

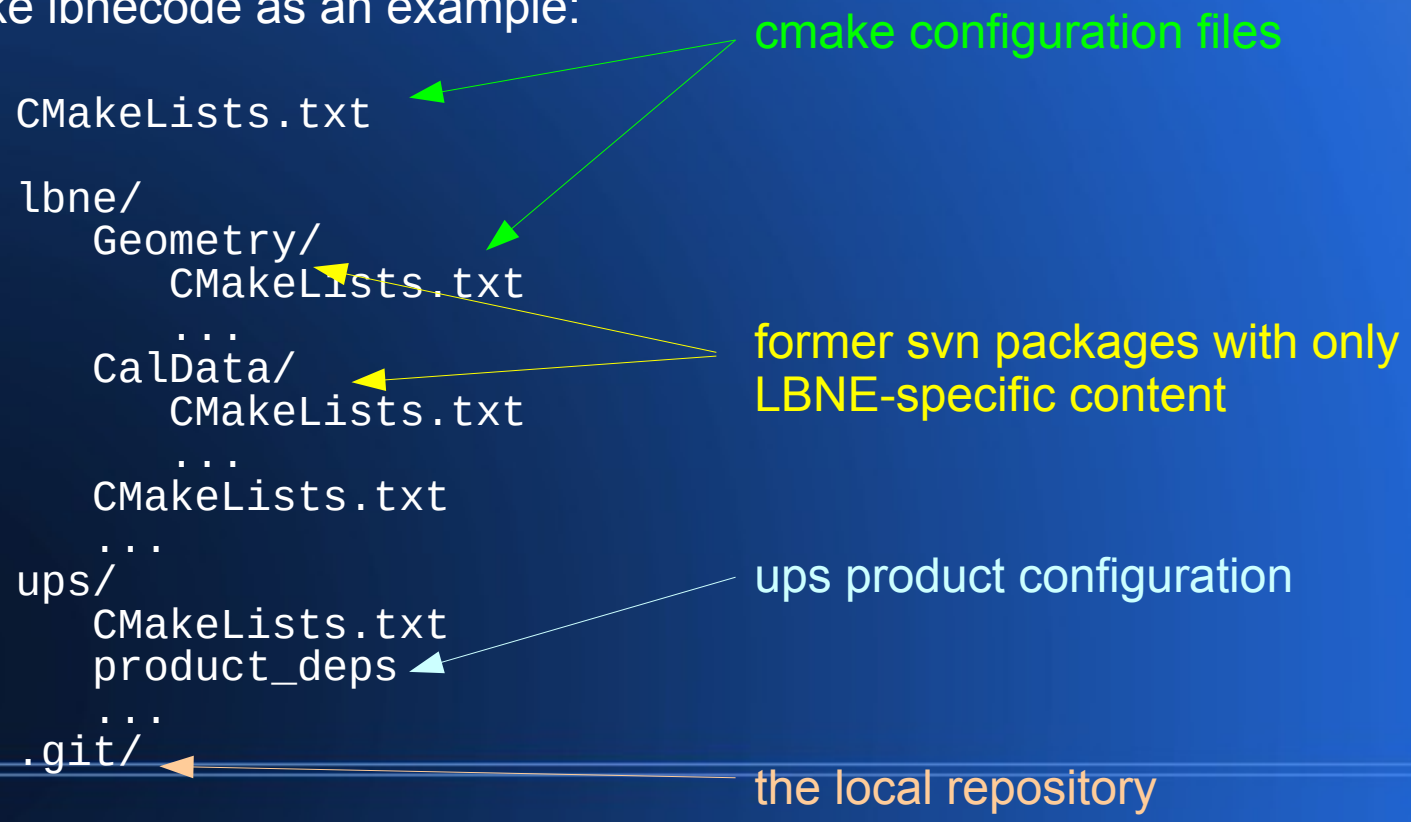
former svn packages with (mostly) same content (since most of the code is detector-agnostic)

ups product configuration:
- what it will depend on
- where source, header, fcl files installed
- product version number

the local repository

# Structure of repositories

- Internal structure of repositories
  - Experiment repositories.
    - Take lbnecode as an example:



# Structure of repositories

- Internal structure of repositories
  - Experiment repositories.
    - Take lbnecode as an example:

```
CMakeLists.txt

lbne/
   Geometry/
      CMakeLists.txt
      ...
   CalData/
      CMakeLists.txt
      ...
   CMakeLists.txt
   ...
ups/
   CMakeLists.txt
   product_deps
   ...
.git/
```

One more layer in directory hierarchy:
Allows packages with same name as in core LArSoft (e.g., Geometry)

#include “lbne/Geometry/...”

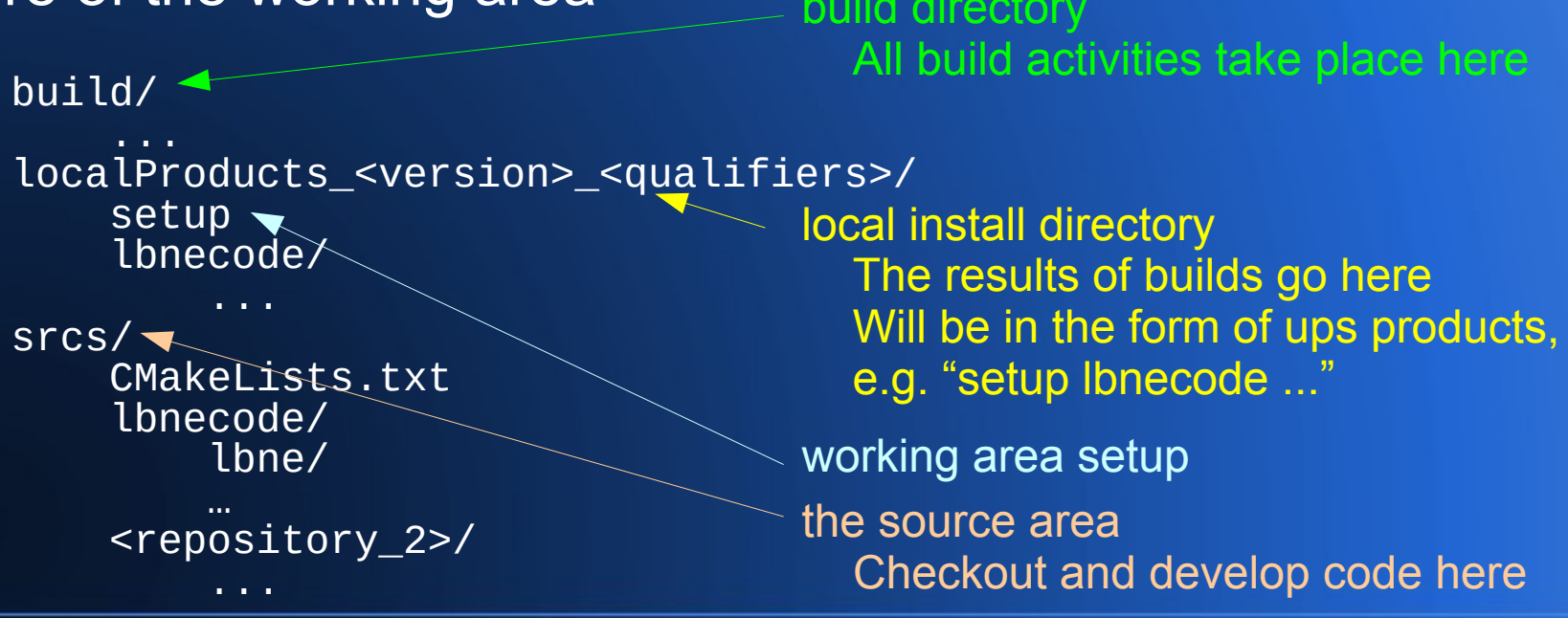
# mrb

- Multi-repository build tool
  - Has several responsibilities
    - Creates working area skeleton (similar to “newrel” under SRT)
    - Assists in checking out code (similar to “addpkg_svn” under SRT)
      - Modifies top-level CMakeLists.txt file appropriately
      - Makes sure you are on the develop branch
    - Configures environment for the build (similar to srt_setup under SRT)
    - Top-level driver for the build/install procedure across multiple repositories
      - Calls buildtool/cmake to do the actual work
      - Supports parallel builds
    - Creates new product / repository skeleton

Information on mrb commands: “mrb help”, “mrb <command> -h”

# The working area

- Code development / build / install takes place in a “working area”
  - Like a “test release” under SRT
  - Create using “mrb newDev” command
- Structure of the working area



# The working area

- Code development / build / install takes place in a “working area”
  - Like a “test release” under SRT
  - Create using “mrb newDev” command
- Structure of the working area

```
build/
    ...
localProducts_<version>_<qualifiers>/
    setup
    lbnecode/
        ...
srcs/
    CMakeLists.txt
    lbnecode/
        lbne/
        …
    <repository_2>/
        ...
```

checked-out repositories
“mrb gitCheckout <repo> [tag]”

# Basic work pattern

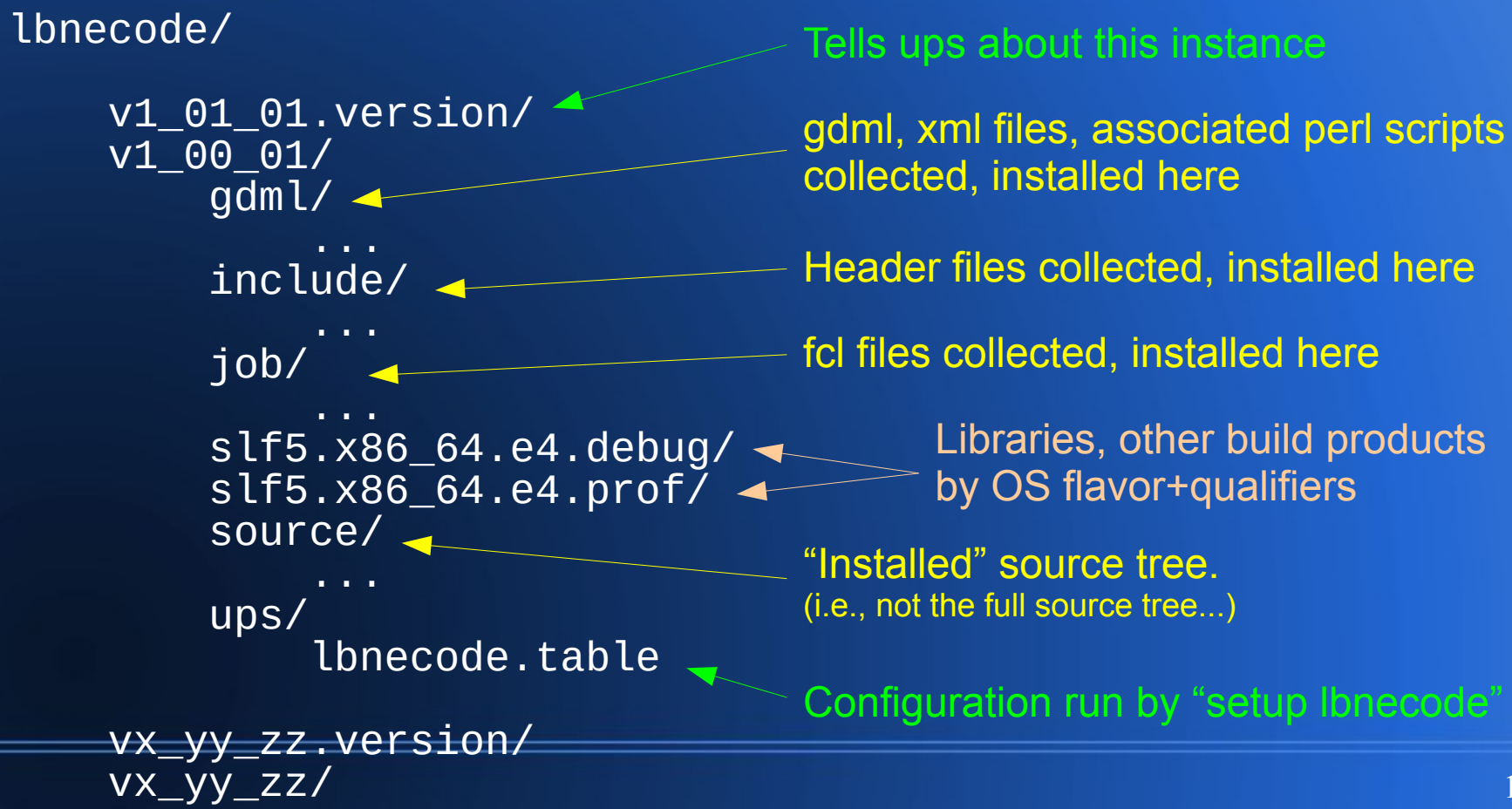
- The general pattern of work
  - Check out and develop code in “srcs”
  - Move to “build” and build the code
    - “Out-of-source” build
      - Keeps all build configuration, intermediate files, results out of source area
      - In principle allows multiple build configurations associated with same source tree
    - In principle can be any directory (except the source tree)
  - Install the build results in “localProducts”
    - Packaged as ups products
    - No install takes place if the build fails
      - Local products don't break when the build fails.
      - Check your build logs!
  - Can then “setup” and run from anywhere
    - “setup <product> ...” will always find the local version first

# Build and install commands

- Two commands available to run the build
  - “mrb build”
    - Runs cmake + build phases
  - “mrb install”
    - Runs cmake + build + install phases
  - Under some circumstances, can use “make” or “make install” instead
    - See the quick-start guide for details
- Many options available to change details of the build
  - For example: `-I <install directory>` to change target installation directory
  - “mrb build -h” for detailed list

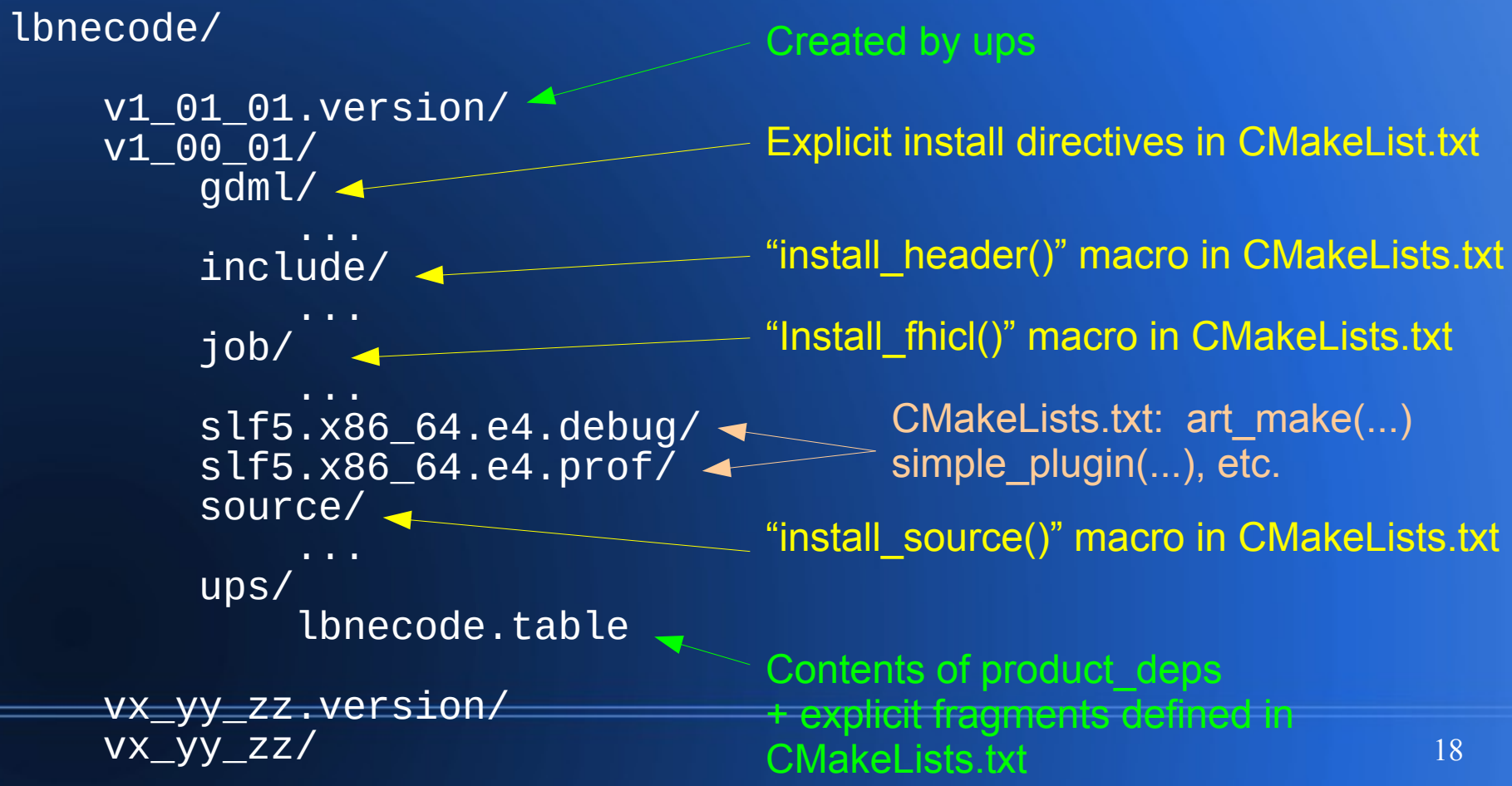
# Structure of installed product

- Starting from localProducts directory:



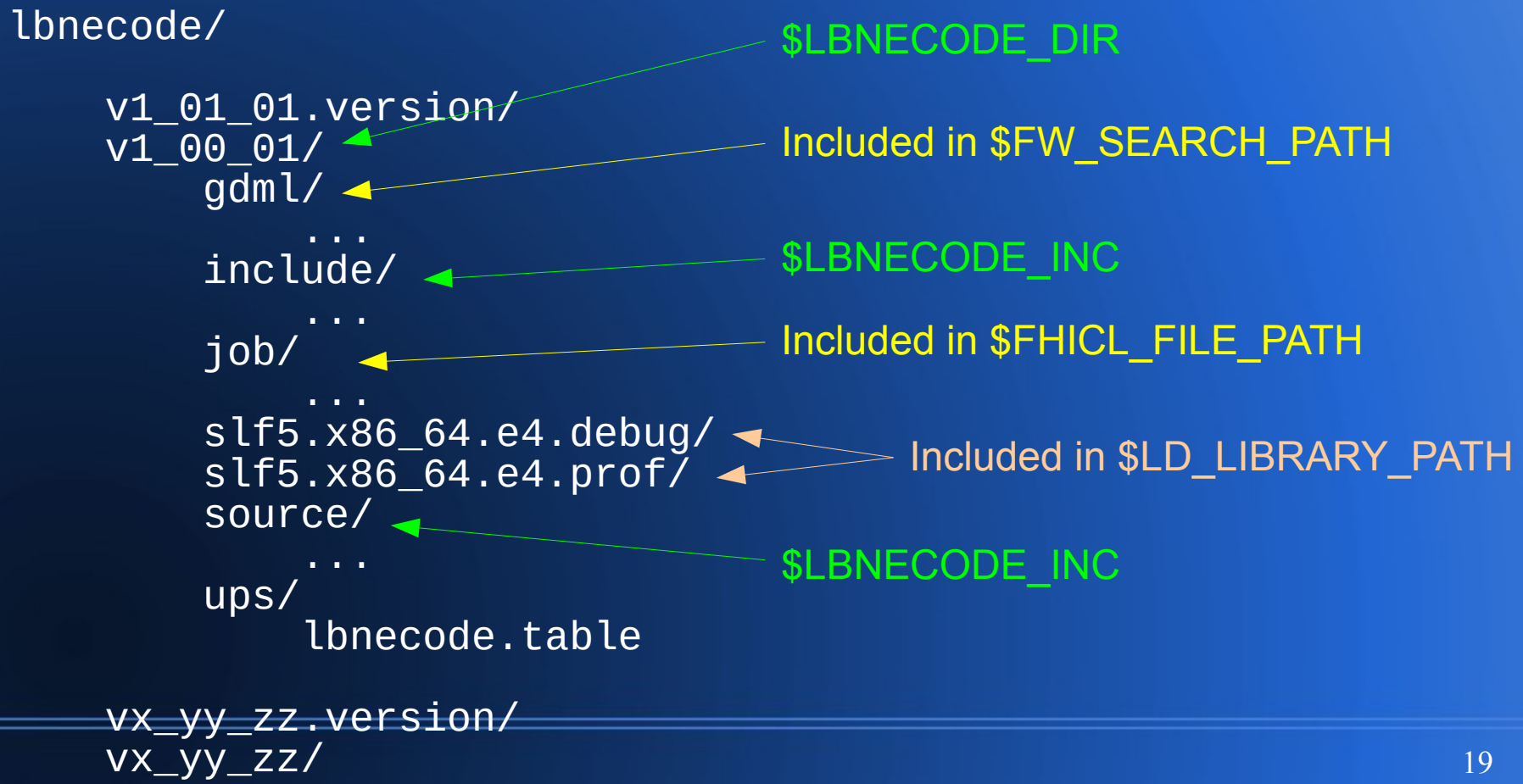
# Structure of installed product

- What determines which files get installed where?



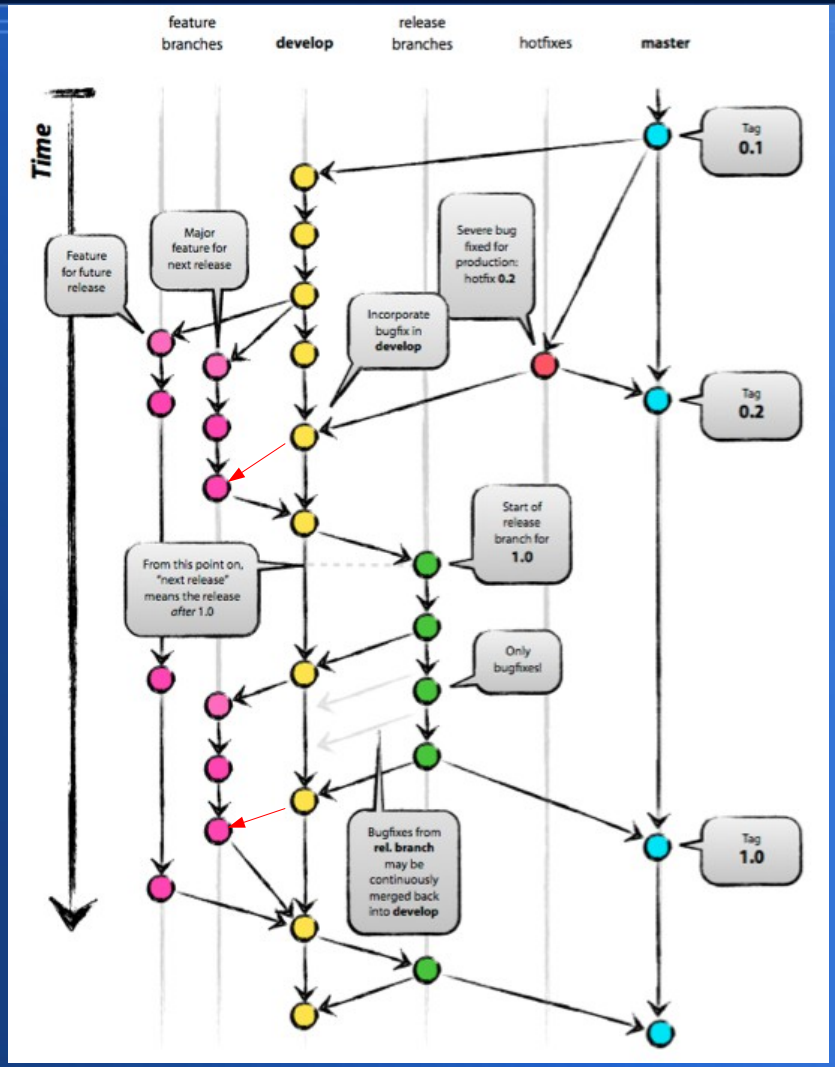
# Structure of installed product

- “setup lbnecode” environment



# git flow and the branching model

- The goal
  - To protect the head of develop branch
    - In the “main” *and* local repositories!
- Strategy
  - Develop code in “feature branches”
  - Merge into develop only after testing
    - Can make the feature branch look exactly like develop
- Use of git flow is optional
  - Strongly encourage use of branching model for development
  - Be sensible




circles = commits = state of the respostory
lines = branches

# Working in the new system:  an example

# Working in the new system

- Walk through an example
  - set up basic environment
  - create / initialize a working area
  - check out, modify code
  - build and install the changes
  - run the new version
- Other sample tasks
  - add / remove a package from repository
  - add / remove a repository from srcs area
  - clean build

More example tasks on the LArSoft wiki and quick-start guide:
https://cdcvs.fnal.gov/projects/larsoft/wiki
https://cdcvs.fnal.gov/redmine/projects/larsoft/wiki/_Quick-start_guide_to_using_and_developing_LArSoft_code_

# Check out and make a change

- From a fresh login

```
#  Create a working area based on larsoft v1_00_01
#
source /grid/fermiapp/lbne/software/setup_lbne.sh
mkdir workdir
cd workdir
mrb newDev -v v1_00_01 -q e4:debug
#
#  Set up the environment for this work area
#
source  localProducts_larsoft_v1_00_01_e4_debug/setup
```

…but wait

# Check out and make a change

```
#
#  Set up the environment for this work area
#
source  localProducts_larsoft_v1_00_01_e4_debug/setup
```

- What does this do? Go back to working area:

```
build/
    ...
localProducts_larsoft_v1_00_01_e4_debug/
    setup
    ...

srcs/
    CMakeLists.txt
    lbnecode/
```

$MRB_BUILDDIR

Added to $PRODUCTS path
Used by ups to find product instances.
Ensures that local version found first

$MRB_SOURCEDIR

# Check out and make a change

- continuing...

```
#  Check out code
#
cd srcs
mrb gitCheckout lbnecode     # abbrev to “mrb g lbnecode”
cd lbnecode
#
#  Suppose this is a major change, so prepare a feature branch
#
git flow feature start rs_newThing

#  ...
#  Now assume changes are ready to build and test
#
cd $MRB_BUILDDIR
source mrb setEnv
mrb install
```

# Check out and make a change

- continuing...

```
#  Oops, typo causes compilation error
#
cd ../srcs/lbnecode
#
# ...fix
#
cd ../../build
make install
#
#  Can use this shortcut if mrb setEnv has been called
#  already in this login session, cmake stage completed,
#  and no new files or repositories have been added/removed
#
#  Assume this succeeds
#
cd ..
setup lbnecode v1_00_01 -q e4:debug
lar -c  some_path/my_job.fcl
```

# Check out and make a change

- continuing...

```
#  Everything works. Now want to push it to the
#  main repository. But first, get any intervening
#  changes to the main repository. Then put those
#  changes on your branch ahead of any of your changes.
#
cd srcs/lbnecode
git pull origin develop
git rebase develop

#  Build and test again!!
cd ../../build
mrb i      # safer at this point than make unless are
           # sure you know what “pull” did
cd ..
lar -c some_path/my_job.fcl
#
#  Merge back into develop and push
#
git flow feature finish rs_newThing
git push origin develop
```

# Add / remove a “package” from a repository

- Want to add DoThisAlgs package

```
cd srcs/lbnecode/lbne
mkdir DoThisAlgs
vi CMakeLists.txt
#  Add “add_subdirectory(DoThisAlgs)” line at end
#
cp Geometry/CMakeLists.txt DoThisAlgs
vi DoThisAlgs/CMakeLists.txt
#  modify appropriately
#  Probably takes only minor changes to library list
#
#  Build as usual. Will need to use “mrb i”
```

- To remove the DoThisAlgs package

```
#  Remove DoThisAlgs package
#
cd srcs/lbnecode/lbne
rm -r DoThisAlgs
vi CMakeLists.txt     # remove “add_subdirectory(DoThisAlgs)”
```

# Remove existing repository from working area

- Remove larcore repository

```
cd srcs
rm -rf larcore
#
#  Need to re-make the top-level CMakeLists.txt in srcs
mrb uc
cd ../build
source mrb setEnv
mrb I
```

# Create a new repository / product

- Add a “newthing” repository / product
  - Assume you already have somewhere to push it once you're done

```
#  Create a repository/product skeleton
cd srcs
mrb newProduct newthing
#
#  Now need to:
#  1) Add content
#  2) Modify top-level CMakeLists.txt
#  3) Create CMakeLists.txt files throughout
#  4) Modify ups/product_deps
#
cd $MRB_BUILDDIR
source mrb setEnv
mrb i
```

# Clean build

- Just delete everything!

```
cd build
rm -rf *
#
#  Can also use “mrb zapBuild”
#
#  Now need to set up environment again
#
source mrb s
mrb install
```

# Nightly builds

- Want to create a version every night from head of repositories
  - Version is “nightly”
  - “setup lbnecode nightly -q e4:debug”
  - New feature:  if a build breaks, previous nightly install remains unchanged
- Status
  - Creating larsoft nightly by hand since transition
  - Script for lbnecode and uboonecode almost working
    - Needed to fix some infrastructure to deal with “nightly” version name
  - Will be under cron jobs in a day or two
  - Scripts can be found in laradmin repository

# Brought to you by...

- The LArSoft team
  - Dave Dykstra
  - Lynn Garren
  - Mike Kirby
  - Gianluca Petrillo
  - Ruth Pordes
  - Brian Rebel
- Many thanks to the beta testers and others who contributed or provided valuable feedback
  - Eric Church
  - Chris Green
  - Herb Greenlee
  - Tom Junk
  - Wes Ketchum
  - Marc Paterno
  - Bill Seligman
  - Tracy Usher
  - Brett Viren
  - And anyone else I forgot...