

EMC Subversion Seminar Series

6. Branching and Merging Best Practices

Paul van Delst

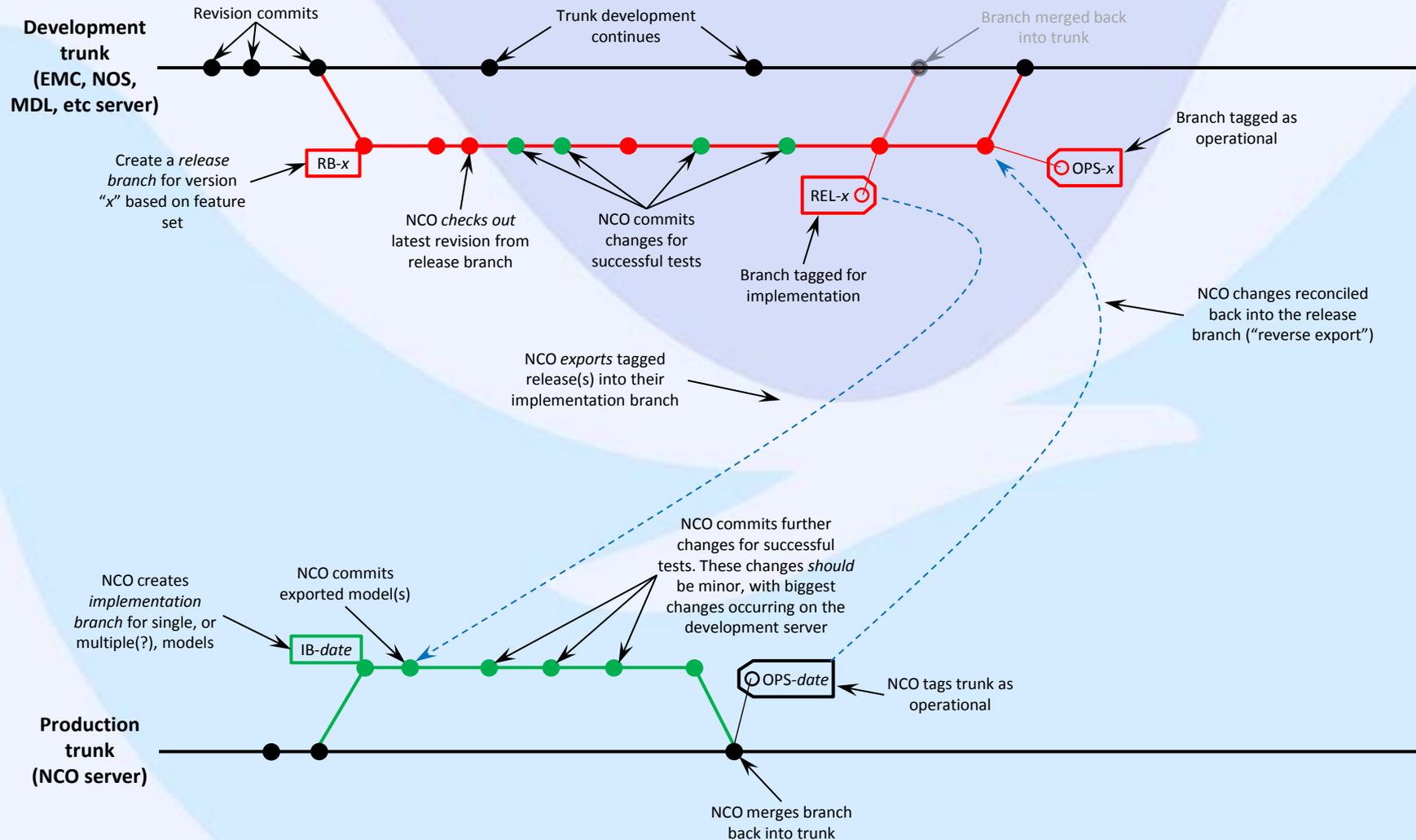


Introduction

- This is a sort-of refresher of the the branching and merging seminars based on usage of the version control setup over the last year or so.
- Much of what you'll see here is a amalgam/compression of previous branching and merging seminars.
 - Only covering synchronisation merges
- I will repeat three things from the intro seminar about what Subversion is *not*:
 - Magic
 - It is not a substitute for management
 - **It is not a substitute for developer communication. Which leads me to...**
- **Document the branch creation, code modifications, sync merging commands and results in the trac ticket you created for your work.**
- What would you like to see presented regarding using Subversion?
 - E.g. conflict resolution?



Context: Subversion usage scenario



Outline

- Branching
 - Guidelines
 - Creating and checking out a branch
 - Externals definitions
 - How not to create a branch
 - Why should I create a branch? (some common patterns)
 - When should I create a branch?
- Merging
 - Guidelines
 - Sync merge command syntax
 - Sync merge setup
 - Possible merge problems



Branching guidelines

- *Preface: These are general guidelines and aren't written in stone. **Context is key** – your team should determine any strict policy-driven methodologies.*
- Branch the entire trunk if you can.
 - This way your branch will be a complete representation of the project.
 - It will also simplify merging.
- Keep your branching as shallow as possible. Bugfix branches aside,
 - Branch from the trunk.
 - Try not to create branches from existing branches.
- Keep your branch lifetimes as short as possible.
 - Remember, the trunk is (should be) the mainline of development, not the branches.
- Avoid the “crawl-in-a-hole” strategy.
 - For situations where branches do exist for a long time (for a suitable definition of “long”), merge the trunk into the branch (aka “synchronisation”) as frequently as possible.
 - Frequent synchronisation of a branch will minimise the number of conflicts that occur at any particular merge, be it trunk→branch, or branch→trunk.



Creating and checking out a branch

- A branch is simply a copy of a particular revision of the source filesystem, *but with history* (important for successful merging).
- To create a branch, you use the **svn copy** command,

```
$ svn copy <FROM> <TO>
```

where the **<FROM>** is the trunk URL, and the **<TO>** is the branch URL, e.g.

```
$ svn copy -m "Creating EXP-MyBranch from trunk" \  
https://.../projects/modelX/trunk \  
https://.../projects/modelX/branches/EXP-MyBranch
```

- When the branch has been created, you use the **svn checkout** command to get a working copy on disk:

```
$ svn checkout [--ignore-externals] \  
https://.../projects/modelX/branches/EXP-MyBranch
```

This optional switch will prevent the checkout of any externally linked projects.



Externals definitions in GFS and GSI

- The **svn:externals** property is a way of linking in other separate and distinct projects.
- If you want to checkout everything, great. If not, use **--ignore-externals**.

The left screenshot shows the Subversion web interface for the GFS project. The URL is `https://svnemc.ncep.noaa.gov/trac/gfs/browser/trunk/projects`. The page title is "GFS Global Forecast System". The source is `trunk/projects`. A table lists the `svn:externals` property:

Name	Size	Rev	Age
Property <code>svn:externals</code> set to			
o enkf			
o gsi			
o obsproc_dump			
o obsproc_prep			
o obsproc_prep_post			
o post			
o verif			

The right screenshot shows the Subversion web interface for the GSI project. The URL is `https://svnemc.ncep.noaa.gov/trac/gsi/browser/trunk/lib`. The page title is "Gridpoint Statistical Interpolation Analysis". The source is `trunk/lib`. A table lists the files in the trunk:

Name	Size	Rev	Age	Author	Last Change
Property <code>svn:externals</code> set to					
o CRTM_REL-2.1.3					

Below the table, a detailed view of the `GSD` file is shown. The `svn:externals` property is set to `CRTM_REL-2.1.3`. The last change is described as: "Trunk: Merged sliu350 branch at r35790 to the trunk. This update adds a ...".



Aside about externals

- Because of project dependencies using externals, typically it's A Good Thing™ to:
- Create externals that link to a specific release or implementation tag.
 - This will ensure that a created branch in the “master” project contains exactly what was in the trunk when the branch was created.
 - If the external link is to the **HEAD** of trunk, or some branch, then that is also what will be checked out or updated in a working copy.
 - Once a branch is created, you can update the externals to whatever dependent project versions you want to point at.
- Communicate important changes in a dependent project to the development team/code manager of the “master” project, and test those changes.
 - This may seem like common sense, but it is not always done.
 - Example: CRTM changes that were self-consistent (and, ehem, seemed like a good idea at the time) caused some consternation in the GSI a few years back.
 - Now the CRTM group maintains a GSI branch where we test CRTM changes simply, e.g. single cycle tests just to make sure data files are in the right place, have the expected names etc.



How *not* to create a branch

- Subversion has no concept of **trunk**, **branches**, or **tags**. Any special meaning attributed to them via their names is simply because we choose to give them that meaning.
- Thus, you could create a “branch” like so:

```
$ cd branches
$ svn mkdir my-new-branch
$ cd my-new-branch
$ svn add ...a whole bunch of files...
$ svn commit ...
```

Please do not do this.

- Because there was no **svn copy** performed, a branch created like that above **has no history**.
- No history means merging is much more difficult – and it can “pollute” the history of whatever you are merging into.
- If you know what you are doing, **and can convince the project and code managers to agree**, fine. Otherwise, boo!



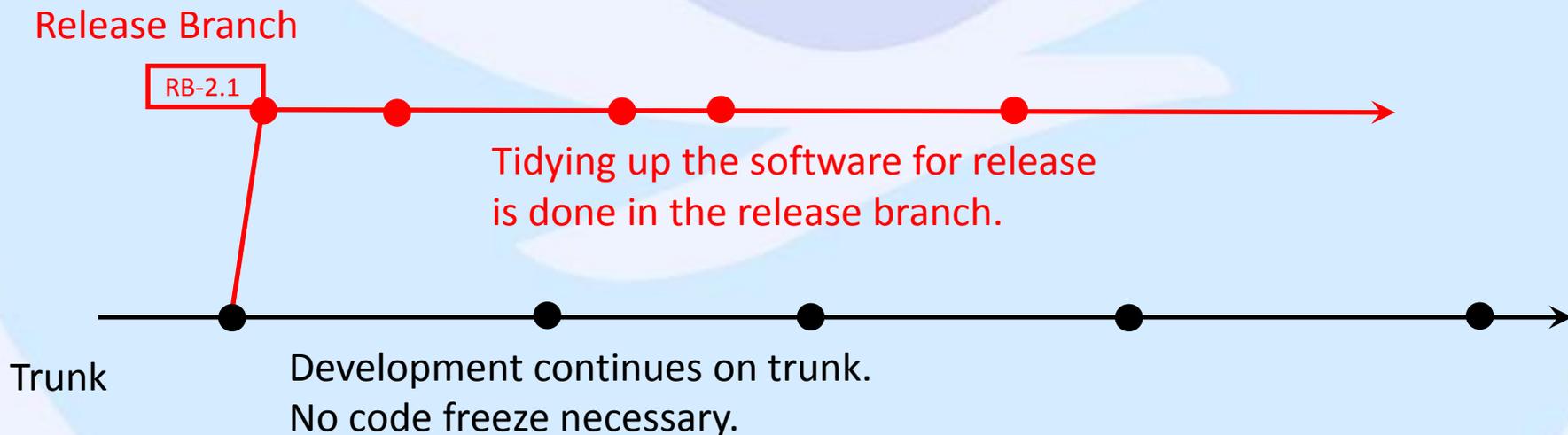
Why create a branch? (1)

- The trunk is the mainline of development - it should always pass a standard set of tests and be “nearly ready” for release or usage. As such, we always should try to ***Protect The Trunk.***
- Branching is a way to isolate yourself, or others, from change.
- There are three typical scenarios to create a branch:
 1. **Release Branch**: You want to release the code and work with NCO SPAs.
 - a. Development is “frozen” for the release.
 - b. Release-specific bug-fixes may be needed.
 2. **Feature Branch**: You want to introduce a new feature into the mainline
 - a. It’s not unreasonable to assume you will break the code (syntax errors, new bugs, etc) in the course of implementing and testing the new feature.
 - b. You still want to be able to commit unfinished code.
 3. **Bugfix branch**: You need to fix a complicated bug
 - a. We won’t cover this type in this seminar.



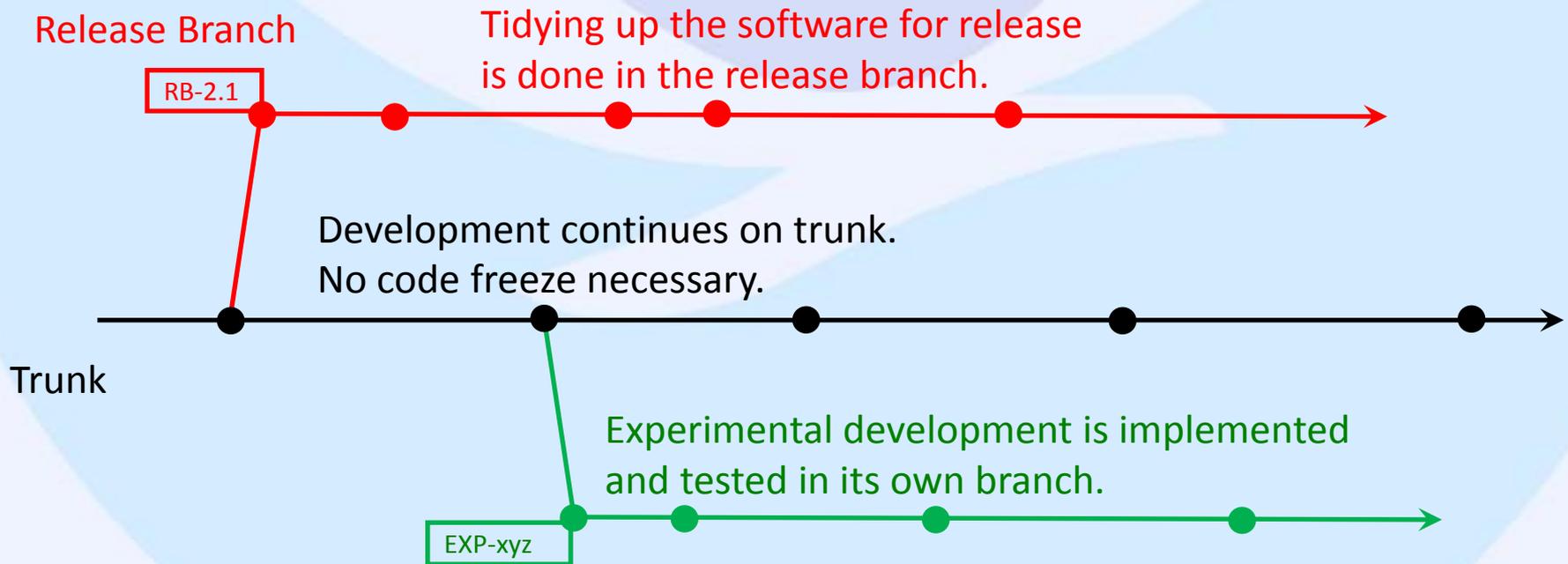
Why create a branch? (2)

- For releases.
 - At some point the development team needs to switch from adding new features to getting ready for the next implementation.
 - Creating a branch for this allows other developers to continue adding new features to the trunk.
 - *The release branch is also what NCO SPAs can work on prior to an implementation release.*



Why create a branch? (3)

- For experimental development.
 - If proposed changes will take some time to implement, or break the software build, create a branch to isolate these changes from the rest of the development team.
 - If the experiment is successful, the branch can be merged to the trunk. If it is not, it can be “deleted”.
 - Most of the EMC project branches are these development branches.



When to create a branch?

- Before creating branches, ensure your entire team is aware the branch is going to be created.
 - Why? So they have the opportunity to merge or commit their working copy changes to the trunk making their changes available for branching.
- Create release branches from the trunk when you get to the point where you wish to start collaborating with NCO SPAs for an implementation.
 - This avoids code “freezes” as work can always be committed to the repository in other branches and the trunk.
- Experimental and bugfix branches can be created anytime based on developer needs. But always *always* try to ensure the rest of the development team is aware of the experiment.
 - The proposed changes may affect someone else’s current work later on.
 - Other developers can suggest potential solutions.
 - The work may already have been done somewhere else.

Merging guidelines

- *Preface: Again, these are general guidelines intended to simplify the merging process, both sync and reintegration. **Context is key** – your team should determine any strict policy-driven methodologies.*
 - Frequent synchronisation of trunk changes into your branch minimises the likelihood of “surprise” conflicts when the branch is reintegrated back into the trunk.
 - Each development team needs to determine the “best” frequency of synchronisation merges. E.g. Once a day? Week? Month?
 - This is where developer communication is important.
 - Are there changes in the trunk you should merge?
 - Are those trunk changes compatible with your changes?
 - Perform merges on the root of the branch, not the subdirectories.
 - Otherwise **svn:mergeinfo** property is written at the subdirectory location.
 - Ensure your merge target working copy contains no “sparse” directories.
 - Similarly with subdirectory merges, **svn:mergeinfo** property is written at subdirectory location.
- Don't use the **--ignore-ancestry** switch unless you discuss with your code manager
 - Never, *ever* edit the **svn:mergeinfo** property directly.



Sync merge command syntax

- Because subversion tracks merge information, the syntax for regular synchronisation of the trunk to a branch is quite simple:

```
$ svn merge ^/modelX/trunk [my-branch-working-copy]
```

Note the caret syntax.
Saves you typing out the
full URL.

Yay!

The working copy target is
still optional. Defaults to
current directory if not
specified.

- You can only merge into a working copy, not directly into the repository.
- How does subversion “know” what revisions need to be merged?
 - That’s what the **svn:mergeinfo** property is for. Storing the merge information.



Sync merge setup

1. Go to the root of your *branch working copy*.

2. Execute an **svn status** command,

```
$ svn status --show-updates
```

to determine if:

- There are later versions of files in the repository (***** in column 9 of output)
- You have any local modifications (**M** in column 1 of output)

3. If (2a) is the case, you *must* issue an **svn update** command,

```
$ svn update
```

to bring your working copy up-to-date. *You should do this anyway as part of your workflow to make sure all the metadata is also up-to-date.*

4. If (2b) is the case, you *should* commit your local modifications prior to performing the merge,

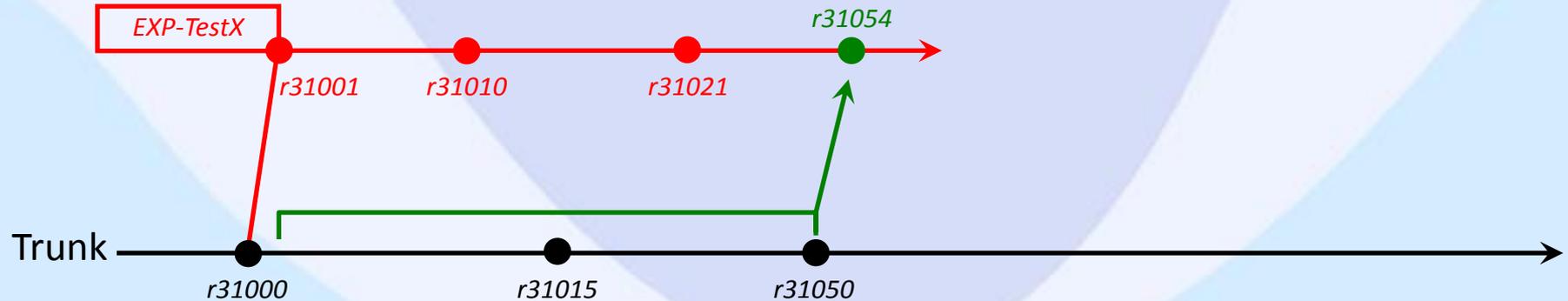
```
$ svn commit
```

5. At this point you have a “clean” working copy. It’s o.k. if you still have unversioned files in there (**?** in column 1 of status output).



Sync merge scenario

- Let's say this is the merge scenario:



1. Issue the **merge** subcommand in the *branch working copy* (use **--dry-run** first!),

```
$ svn merge ^/modelX/trunk .
```

2. Deal with any conflicts. (Ha! Easy to say. Separate seminar?)
3. Commit the changes.

```
$ svn commit -m \  
    "EXP-TestX branch: Sync'd trunk to branch"
```

Possible merge problems (1)

- Commit fails with an “directory out-of-date” error.

```
$ svn commit -m "my log message"
svn: Commit failed (details follow):
svn: Directory '.' is out of date
```

Issue an “`svn update .`” to get all the directory metadata up-to-date.

- The sync merge was performed into a branch subdirectory, not the root directory.
 - This is not really a “problem”, but it does mean there will be mergeinfo associated with that subdirectory rather than the root. Can be confusing for subsequent reintegration.

global_fcst.fd in branches/EXP-spptfix/para/sorc - GFS

File	Size	Download	Revision	Age	Author	Comment
wrt3d_hyb.f	13.3 KB	⬇	31495	6 months	nicole.mckee	TRUNK UPDATE: Committing code clean up fo
wrtg3d.f	10.1 KB	⬇	31495	6 months	nicole.mckee	TRUNK UPDATE: Committing code clean up fo
wrtg3d_hyb.f	10.0 KB	⬇	31495	6 months	nicole.mckee	TRUNK UPDATE: Committing code clean up fo
wrtst.f	72.5 KB	⬇	31495	6 months	nicole.mckee	TRUNK UPDATE: Committing code clean up fo
wrtout.f	199.8 KB	⬇	34436	3 months	nicole.mckee	TRUNK UPDATE: Bug fix in wrtout.f and updat
wrtsfc.f	98.7 KB	⬇	31495	6 months	nicole.mckee	TRUNK UPDATE: Committing code clean up fo
ysminv.f	3.9 KB	⬇	23579	15 months	kate.howard	Committing code clean up fo

Property svn:mergeinfo set to (toggle deleted branches)

/branches/EXP-restart-fix/global_fcst.fd	merged	eligible
/branches/EXP-tic107/para/sorc/global_fcst.fd	merged	eligible
/branches/EXP-tic96/para/sorc/global_fcst.fd	merged	eligible
/branches/EXP_tic132/para/sorc/global_fcst.fd	merged	eligible
/branches/moorthi/src/global_fcst.fd_Dec11	merged	eligible
/branches/moorthi/src/global_fcst.fd_Nov19	merged	eligible
/branches/moorthi/src/global_fcst.fd_ticket_46	merged	eligible
/branches/moorthi/src/global_fcst_r32604.fd	merged	eligible
/branches/ticket93B/para/sorc/global_fcst.fd	merged	eligible
/trunk/para/sorc/global_fcst.fd	merged	eligible

Lots of mergeinfo at subdirectory level. Some from branches with different named directories. (Yoicks!)

BTW, not singling out this branch in GFS...just a random selection. These merges proly weren't done in this branch.

Possible merge problems (2)

- The sync merge was performed into a “sparse” checkout.
 - A sparse checkout is where you specify the directory depths you want in your working copy via the “`--depth`” switch to `svn checkout`, and the “`--set-depth`” depth switch to `svn update`.
 - Again, not really a problem, but it does modify mergeinfo in directories where no changes were made. But, it can be confusing when you see subsequent commits modifying directory properties you *know* you didn’t change because you didn’t even check them out.

```
#421 (netCDF coefficients) - CRTM
• Performing another trunk sync to get those changes into the EXP-netCDF branch. Command:
EXP-netCDF: svn merge -r37592:37675 ^/crtm/trunk .
--- Merging r37593 through r37675 into '.':
U   test/Main/timing_test/timing_test.sh
A   test/Main/timing_test/emc-lw-pvandel.crontab
A   test/Main/timing_test/wandaboos.crontab
D   test/Main/timing_test/timing_test.crontab
--- Merging r37593 through r37675 into 'src':
U   src/Utility/Timing_Utility.f90

Committed at r37679.

It looks like my sync merge the other day at home (into a sparse copy) has caused some
mergeinfo issues that will undoubtedly come back to bite me in the ass upon reintegration.
The list of changed files in the commit message editor contains a bunch of items where only
the properties have been changed, in particular those directories that were not updated with a
--set-depth=infinity:
_M .
_M test/Main/timing_test/timing_test.sh
A + test/Main/timing_test/emc-lw-pvandel.crontab
A + test/Main/timing_test/wandaboos.crontab
D test/Main/timing_test/timing_test.crontab
_M doc
_M externals
_M fix/EmisCoeff/IR_Land/UWIREMIS
_M fix/EmisCoeff/IR_Water/netCDF
_M fix/ACCoeff
_M fix/ACCoeff/ACCoeff.nc
_M fix/SpcCoeff/Big_Endian
_M fix/SpcCoeff/Band_Correction_Statistics
_M fix/SpcCoeff/Little_Endian
_M fix/BeCoeff
_M fix/Be_LUT
_M fix/TauCoeff/ODPS/netCDF
```

These are the actual changes I made merging into my sparse copy.

These are the actual changes committed to my branch. The “M” means only properties have been changed.

Possible merge problems (3)

Changeset 37679 for branches/EXP-netCDF - CRTM

Unmodified Added Removed

branches/EXP-netCDF

- Property **svn:mergeinfo** changed
`/trunk` merged: `37624,37628,37638,37675`

branches/EXP-netCDF/doc

- Property **svn:mergeinfo** set to (toggle deleted branches)
`/branches/EXP-ScatteringIndicator/doc` merged eligible
`/branches/RB-2.1/doc` merged eligible
`/trunk/doc` merged **non-inheritable** eligible

branches/EXP-netCDF/externals

- Property **svn:mergeinfo** set to (toggle deleted branches)
`/branches/EXP-ScatteringIndicator/externals` merged eligible
`/branches/RB-2.1/externals` merged eligible
`/trunk/externals` merged non-inheritable eligible

Funky new mergeinfo status!

Subsequent merges still modify the subdirectory mergeinfo – “with no actual effect”. Sheesh.

Changeset 37839 for branches/EXP-netCDF - CRTM

Unmodified Added Removed

branches/EXP-netCDF

- Property **svn:mergeinfo** changed
`/trunk` merged: `37734,37737,37742,37824,37838`

branches/EXP-netCDF/doc

- Property **svn:mergeinfo** changed (with no actual effect on merging)

branches/EXP-netCDF/externals

- Property **svn:mergeinfo** changed (with no actual effect on merging)

branches/EXP-netCDF/fix/ACCoef

- Property **svn:mergeinfo** changed (with no actual effect on merging)

- Update a sparse working copy for merging using:

```
$ svn update --set-depth=infinity [branch root dir]
```

Summary, final thoughts

- **Branching**
 - Branch the entire trunk, not subdirectories.
 - Keep the branching shallow.
 - Keep branch lifetimes short.
- **Merging**
 - Sync your branch with the trunk frequently.
 - Merge into the root directory of your branch, not subdirectories.
 - Don't merge into a sparse checkout of your branch.
 - Don't use the **--ignore-ancestry** switch.
 - Don't edit the mergeinfo directly.
- Document the branch creation, code modifications, sync merging commands and results in the trac ticket you created for your work. (You created one, right?)
- If you have any questions or worries about branching or merging, please don't hesitate to contact me, Nicole, Kate, or Mike for help.
 - Particularly if you've encountered conflicts – they should be dealt with as they arise.
 - Just like personal ones. Otherwise they fester away... 😊