GROMACS - Task #845

Format white space correctly

11/25/2011 01:14 PM - Roland Schulz

Status:	Closed		
Priority:	Normal		
Assignee:			
Category:	build system		
Target version:	5.0		
Difficulty:	uncategorized		

Description

Discussion started on https://gerrit.gromacs.org/#/c/260/ but should have its own thread.

Related issues:

Related to GROMACS - Task #818: Clean up source file headers and set up autom...	**Closed**	**09/22/2011**
Related to Support Platforms - Task #648: Use of commit hook scripts	**Closed**	
Has duplicate GROMACS - Task #664: Update code formatting to follow coding co...	**Closed**	**01/13/2011**

Associated revisions

Revision 5df09bc1 - 01/13/2013 12:55 PM - Teemu Murtola

Remove unused bfunc.h.

In addition to being unused, it also contains a non-terminated comment.

Related to #845 (uncrustify doesn't like the comment).

Change-Id: I662d2ecde5e58b6babc211010d4364b27e58c564

Revision 3339ffd6 - 01/18/2013 09:22 PM - Roland Schulz

Code beautification with uncrustify

We are going to try to keep our eyeballs from bleeding when we
read prehistoric code from this project while maintaining it.
So we need a point in time where release-4-6 gets beautiful
in tandem with master. That time is now.

At some point we hope to be able to do this automatically so
we can have reviewers concentrate on code quality and not
formatting trivia. But we have to do a whole-of-project
fix first.

Refs #845

Change-Id: I8e99228cc118f75c0831e8297c171fcbbabc8f73

Revision 987b4f3a - 02/10/2013 08:04 PM - Teemu Murtola

Manual reformatting in preparation for uncrustify.

Reindent example code blocks in Doxygen comments to align as uncrustify
would align them. New enough Doxygen should be able to strip the leading
whitespace from the blocks. Disable uncrustify indentation in one
static data declaration where uncrustify can't keep the original manual
indentation.

Related to #845.

Change-Id: I585852db6fc08557ca55cba41a1349533e1b9c4d

Revision 74158191 - 02/25/2013 04:57 PM - Teemu Murtola

Uncrustify template.cpp and selection.cpp.

template.cpp was missed in the original reformatting, and selection.cpp

got rebased past the uncrustify commit and got missed in reformatting
that I did for such changes.

Part of #845.

Change-Id: I9753e6add8aed894b10e829b63dc729f0d3d9b4b

Revision a48cd30b - 02/26/2013 05:59 PM - Mark Abraham

Uncrustified code changes since 4.6

The group kernel python scripts have changed slightly to emit
code that uncrustify won't want to change. Group kernel
generation and uncrustify are now mutual null operations.

Recent changes to the declarations in gmx_order in baa65b60
may have exposed a possible bug in uncrustify. Those declarations
have been simplified so that there will be no future issue from
that possible bug.

Refs #845

Change-Id: I33495d57ca37317cef4bf12707fd77d67309d292

Revision 921e2c99 - 06/13/2013 09:45 AM - Teemu Murtola

Uncrustify template.cpp and selection.cpp.

template.cpp was missed in the original reformatting, and selection.cpp
got rebased past the uncrustify commit and got missed in reformatting
that I did for such changes.

Part of #845.

Change-Id: I9753e6add8aed894b10e829b63dc729f0d3d9b4b

Revision 78ccd172 - 06/13/2013 09:45 AM - Mark Abraham

Uncrustified code changes since 4.6

The group kernel python scripts have changed slightly to emit
code that uncrustify won't want to change. Group kernel
generation and uncrustify are now mutual null operations.

Recent changes to the declarations in gmx_order in baa65b60
may have exposed a possible bug in uncrustify. Those declarations
have been simplified so that there will be no future issue from
that possible bug.

Refs #845

Change-Id: I33495d57ca37317cef4bf12707fd77d67309d292

Revision 9f0d8f20 - 07/01/2013 12:50 AM - Teemu Murtola

Script for running uncrustify for modified files.

Comments in the script explain how to use it manually or automatically.
The script uses the git filter attribute to indentify which files to run
uncrustify for. This makes it possible to configure a git filter for
running uncrustify instead of using the script. This is also described
in the script comments.

The script is easily extensible to do other types of automatic
reformatting, e.g., to check/add copyright headers.

Also contains a pre-commit hook for running uncrustify. Currently only
checks the formatting; does not yet automatically reformat anything, as
it is a bit unclear what is the best way to do that.

Related to #845.

Change-Id: I389f2ebf43182d4827d1ec10cb6c1332bd4bf8de

Revision ea999126 - 01/29/2014 08:19 PM - Roland Schulz

Uncrustify all files

Add a script that can be used to do this again if necessary. The script also supports listing the files based on the filter attribute, and checking the copyright; can be extended if/when there is need.

Some manual reformatting in gpu_utils.cu, nbnxn_cuda_data_mgmt.cu, and nbnxn_cuda_kernel.cuh. Updated the Sparc kernel generator to generate code that is invariant under uncrustify, like the other kernels already do.

Copyright not updated, except for fixing readpull.c such that the script doesn't complain.

Part of #845

Change-Id: Ia77738ec781f75f1c4e7a264734aac884321f3e5

History

#1 - 11/25/2011 01:20 PM - Roland Schulz

(This is referring to the questions on Gerrit. Please check the above link for context)

Why do you prefer indent? Is the result as good? As long as we don't plan to add it to the user side git hooks it wouldn't matter that it is less common.

The style should follow: http://www.gromacs.org/Developer_Zone/Programming_Guide/Code_Formatting

I don't think it would be a good idea for Gerrit to change the commit automatically. I think it would be better to either 1) just relay on the Gerrit highlighting of whitespace errors + manual review, or 2) check the indentation automatically as part of the auto verification planned. That way the change would automatically be marked as incorrect white space in the comments. But the reviewers could still approve a change (in case the auto indent is wrong). But I'm not sure someone has the time to write the script integration with Gerrit.

#2 - 11/25/2011 11:14 PM - Mark Abraham

I preferred indent because it's a GNU tool and thus likely ubiquitous (and I personally don't care about Windows support!). astyle would require separate installation and its latest versions are not fully cross-platform and the dev who responded to my bug report didn't seem to care.

I think we should offer a user-side git hook that people can elect to use so that they know their code won't be rejected later because of formatting. At the very least we can approve a particular indent command line. I'll look at how that might work.

How do we configure gerrit to flag appropriate whitespace errors (point 1) above)? At the moment it notices trailing whitespace.

Do you mean we harness CTest or such for point 2)?

#3 - 11/25/2011 11:58 PM - Mark Abraham

Then again indent doesn't do C++, so scratch that.

#4 - 11/26/2011 12:27 AM - Roland Schulz

http://uncrustify.sourceforge.net/ seems quite configurable. Might be another good option.

#5 - 11/26/2011 12:32 AM - Roland Schulz

I don't think we can configure Gerrit regarding highlighting more/less whitespace errors. We could only use the ones it does show and the rest we would need to review manually with option 1.

Gerrit has an interface to interact with CI programs. We could either write a script to directly use that interface or we could write/use a plugin for the CI software. Currently it looks like we will use CDash and it doesn't support plugins. But in case we use Jenkins/Hudson it supports plugins and plugins already exist for static code analysis. Maybe someone has already done a plugin for format check or otherwise it would be easy to add one. So it depends a bit on what CI server we will use.

#6 - 12/05/2011 02:12 PM - Mark Abraham

I had a play with uncrustify (linked above by Roland) and I think it does a good job and is highly configurable. I have done some minor tweaks to the default configuration file to have it adhere to our minimal style guide (linked above).

I envisage we leave the config file in the admin directory and offer some voluntary git pre-commit hooks that will massage the code to suit the style. Also a Jenkins plug-in to report back on the adherence to the style. I haven't yet found a Jenkins plugin that works to check C/C++ code style, but it can't be too hard to hack something to do a in-place re-style on the CI machine. Then Jenkins can report some degree of error if the git diff is non-null, and show the offending code.

We will need to tweak the uncrustify config further for C++, and this is perhaps best done in consultation with Teemu.

Once Gerrit 310-312 have been accepted, I propose that I'll upload a commit with the beautified release-4-6 source and the config file and invite

comments on gmx-core.

This shouldn't create serious merge issues since the fallback of using uncrustify on the incoming branch will always exist.

#7 - 12/05/2011 06:00 PM - Teemu Murtola

I think that my comment to #818 applies to some extent here as well: Right now is about the worst time to implement things that will cause changes to many source files, since there are so many active branches. True, git merge tools could be good enough that merging would not become a nightmare, but at a minimum, automatic formatting should be applied to all branches to about the same time, including master. Right now, there is actually a need to do a merge from release-4-5-patches -> release-4-6 -> master to get the autobuild working, and large-scale formatting changes could cause issues.

About the formatting itself, is it possible to only configure e.g. uncrustify such that it will only enforce a certain set of rules, e.g., related to block indenting and brace placement? Our minimal style guide doesn't, at the moment, have any guidelines on, e.g., indenting continuation lines or placing line breaks in conditionals, and I don't know whether it would even be very productive to add strict rules on that.

#8 - 12/06/2011 01:41 AM - Mark Abraham

Teemu Murtola wrote:

> I think that my comment to #818 applies to some extent here as well: Right now is about the worst time to implement things that will cause changes to many source files, since there are so many active branches.

Sure, but there will never be a development freeze, so it's unlikely in the next year or more that there's any time with fewer branches under active development. Those dealing with fixing bugs in malformed code will appreciate no longer needing safety goggles. :) For example, (and off the top of my head) parts of src/gmxlib/xvgr.c or src/gmxlib/gmx_random.c have code that looks like somebody indented some new lines with a single dumb tab or something, regardless of surrounding indentation. These are by no means the worst such I have seen.

Another consideration is that such whole-scale reformatting touches the history of every development path, and so should take place as early as possible when lots of potentially-bug-introducing development will take place.

> True, git merge tools could be good enough that merging would not become a nightmare, but at a minimum, automatic formatting should be applied to all branches to about the same time, including master.

I agree, except that release-4-5-patches is for bug fixes only, and so will not be re-formatted.

> Right now, there is actually a need to do a merge from release-4-5-patches -> release-4-6 -> master to get the autobuild working, and large-scale formatting changes could cause issues.

Yes, master and 4-6 should be done together. Easy instructions need to be made available for authors of incoming branches so things will line up.

> About the formatting itself, is it possible to only configure e.g. uncrustify such that it will only enforce a certain set of rules, e.g., related to block indenting and brace placement? Our minimal style guide doesn't, at the moment, have any guidelines on, e.g., indenting continuation lines or placing line breaks in conditionals, and I don't know whether it would even be very productive to add strict rules on that.

Agreed, many of these things are the least of our worries. Uncrustify does support a minimalist approach. In most such cases you can tell it to add, remove or ignore the spacing of a feature.

#9 - 12/06/2011 02:24 PM - Teemu Murtola

Mark Abraham wrote:

> Sure, but there will never be a development freeze, so it's unlikely in the next year or more that there's any time with fewer branches under active development. Those dealing with fixing bugs in malformed code will appreciate no longer needing safety goggles. :) For example, (and off the top of my head) parts of src/gmxlib/xvgr.c or src/gmxlib/gmx_random.c have code that looks like somebody indented some new lines with a single dumb tab or something, regardless of surrounding indentation. These are by no means the worst such I have seen.

I agree that some of the files are quite a mess when it comes to indentation. And sure, there will not likely be any development freeze. But right now, in addition to the two release branches and master, there are also at least half a dozen mature branches ready to be merged to release-4-6, and any reformatting applies to them as well. Even more so if/when we encourage rebasing those branches before merging, which means that contributors need to reformat every single commit, after resolving possible conflicts caused by the reformatting. And that reformatting may cause additional merge conflicts in the downstream commits...

> Another consideration is that such whole-scale reformatting touches the history of every development path, and so should take place as early as possible when lots of potentially-bug-introducing development will take place.

But if you are introducing this kind of reformatting when several concurrent development branches already exist, you implicitly assume that merge and rebase tools in git are smart enough to deal with the arising issues (otherwise, a lot of work would pile up on others, not just on the person who did the first reformatting). So why wouldn't, e.g., git blame -w, also be smart enough?

Also, since development for 5.0 will also potentially touch quite a large part of the source, why not do this reformatting only in master before that work really begins, at a point when there is no longer need for large changes in other branches? That wouldn't mess up the history significantly more than the reorganization already does. I would think that, e.g., after 4.6 has been released and merged to master would be a much more suitable point to do this reformatting. And since this has been known for some time, it wouldn't be as likely to break others' work, or at least people would be more prepared.

> True, git merge tools could be good enough that merging would not become a nightmare, but at a minimum, automatic formatting should be applied to all branches to about the same time, including master.

> I agree, except that release-4-5-patches is for bug fixes only, and so will not be re-formatted.

I don't object, assuming that the merge tools actually work across whitespace changes well enough.

> Right now, there is actually a need to do a merge from release-4-5-patches -> release-4-6 -> master to get the autobuild working, and large-scale formatting changes could cause issues.

> Yes, master and 4-6 should be done together. Easy instructions need to be made available for authors of incoming branches so things will line up.

I can try to contribute to the discussion of formatting C++ code, but other than that, I don't really have much at stake here. I'm just trying to point out that this will likely introduce yet more additional work on the people working on merging their contributions to 4.6, and adding requirements on the incoming branches "on-the-go" doesn't help in trying to push the 4.6 release out quickly, as I think was the original plan...

#10 - 12/07/2011 10:08 AM - Mark Abraham

I did a test run of dealing with uncrustification using the qhop and release-4-6 (r46) branches (and I know nothing about the former). After some experimenting, I found a really easy workflow:

1. Suppose at some point, r46 was uncrustified with something like (temp=`mktemp`; find share src include -name *.[ch] > $temp; uncrustify -c admin/uncrustify.cfg -l C -F $temp --no-backup) and that commit tagged "uncrustify". Development on r46 then continued.

2. When it is time to merge qhop, check it out and git merge uncrustify~1. Resolve as normal.

3. The next commit in the r46 branch has only formatting changes. In principle, we should just be able to merge with it, but I found that git merge uncrustify would get a bit confused about changed brace indenting levels and introduced braces. However, there is a simple work around. git merge uncrustify -s ours --no-commit is nearly a null operation. We get left with the content of qhop in crusty form with the appropriate commit from each branch as a parent and the merge commit still pending. Now, repeat the uncrustify step, and git commit completes the merge. No fuss, no decisions. We get left with the result of step 2 in uncrustified form and it will fast-forward merge into r46 later.

4. Now git merge r46 to get the rest of the r46 development. Resolve as normal.

5. Switch to r46 and git rebase qhop and the merge is done.

Note that the work of 2, 4 and 5 are necessary whether or not uncrustification takes place. The overhead of uncrustify is

- installation (download, ./configure, make, make install)
- the bureaucracy of stopping the normal merge as in 2
- step 3 (about two minutes to uncrustify)
- the bureaucracy of starting merging again

So I estimate around an extra 20 minutes of work per branch incoming to r46 due to uncrustification if done by someone who has to install uncrustify and never uses it again. However step 3 is so easy that one option is that I tell people that if they do step 2 and upload, I'll handle step 3 overnight.

#11 - 12/07/2011 05:53 PM - Roland Schulz

Why would you want to first merge r46 before rebasing? I thought rebase ignores merges so this would have no effect.
So without uncrustify I would think one would only do 5.
And even with uncrustify I don't see how it would help to merge first. But of course than you can have the problem Teemu mentioned, that one can get merge conflicts on each individual commit which is rebased.

The only way I can see one could rebase ontop of a branch with the proposed uncrustify settings would be to use "git filter-branch" first on the branch. Using uncrustify as the filter. That way each commit in the branch would be fixed up before the rebase and the rebase should work.

Or an alternative would be to tell uncrustify to only fix white-spaces (assuming that this is possible) and not fix braces. Than it should be possible to rebase/merge without additional problem because git has options to ignore white-spaces.

#12 - 12/07/2011 06:35 PM - Teemu Murtola

Steps 1-4 in Mark's workflow are exactly how I would have done the merge if it were necessary and I wouldn't need to care about the multitude of merge commits introduced (in fact, this is how I did the original restructuring in master a year back, where I already had some of the work in a separate branch). But step 5 is incorrect: git rebase qhop (tries to) rebase the current branch on top of qhop, and not the other way around as

intended. And since step 4 already merged r46 to qhop, the rebase is a no-op without the -f flag. So as Roland mentions, I don't think that those merges help in the rebase problem (except perhaps in simple cases with git-rerere enabled).

I just learned about git filter-branch earlier today, which could be the solution here (as Roland already pointed out) if we absolutely want this done in release-4-6 branch. Even the whitespace-only fix may have issues: after the rebase, the rebased commits may have incorrect indentation where new lines are added. So if you can come up with a bullet-proof git filter-branch invocation whose result doesn't need manual inspection, and can sell the additional work to those who actually need to do it, I don't have objections. Except perhaps that if/when master needs to be reformatted soon, I can't promise to have much time available for the C++ issues.

#13 - 12/08/2011 01:44 PM - Mark Abraham

Roland Schulz wrote:

> Why would you want to first merge r46 before rebasing? I thought rebase ignores merges so this would have no effect.

I don't know what you mean by "rebase ignores merges". 2 and 4 could/should be done as rebases if the feature branch has not been pushed to a public repository.

> So without uncrustify I would think one would only do 5.

As above, there are cases where only a merge is acceptable. Step 5 is effectively a relabelling, but actually has to be done as with git merge because the branch is public.

> And even with uncrustify I don't see how it would help to merge first. But of course than you can have the problem Teemu mentioned, that one can get merge conflicts on each individual commit which is rebased.

If step 5 is a merge then by construction there is no way to get uncrustify-based merge conflicts. I've done this series of steps on a feature branch I knew nothing about, and it compiled first time :)

> The only way I can see one could rebase ontop of a branch with the proposed uncrustify settings would be to use "git filter-branch" first on the branch. Using uncrustify as the filter. That way each commit in the branch would be fixed up before the rebase and the rebase should work.

That would be a workable approach for a private-only branch. However you would still want to do step 2 as a rebase before git filter-branch with uncrustify, lest you get the same problem in reverse. It is necessary that both branches uncrustify "at the same time", that is, when both have all the commits from r46 before the uncrustify one.

> Or an alternative would be to tell uncrustify to only fix white-spaces (assuming that this is possible) and not fix braces. Than it should be possible to rebase/merge without additional problem because git has options to ignore white-spaces.

IMO inserting proper bracing is far more valuable than proper indenting, but if someone identifies a real problem, then it could be an acceptable part solution.

#14 - 12/08/2011 02:00 PM - Mark Abraham

Teemu Murtola wrote:

> Steps 1-4 in Mark's workflow are exactly how I would have done the merge if it were necessary and I wouldn't need to care about the multitude of merge commits introduced (in fact, this is how I did the original restructuring in master a year back, where I already had some of the work in a separate branch).

Doing steps 2, 3, 4 and 5 with git merge results in three merge commits per feature branch, one each from 2, 3 and 4. I've tested this with my qhop experiment. This is an increase of two on the minimum for a public branch. So I'm not sure where the multitude of merge commits comes from?

> But step 5 is incorrect: git rebase qhop (tries to) rebase the current branch on top of qhop, and not the other way around as intended. And since step 4 already merged r46 to qhop, the rebase is a no-op without the -f flag. So as Roland mentions, I don't think that those merges help in the rebase problem (except perhaps in simple cases with git-rerere enabled).

Indeed, as in my previous post, step 5 should be a git merge qhop - there is no rebase problem.

> I just learned about git filter-branch earlier today, which could be the solution here (as Roland already pointed out) if we absolutely want this done in release-4-6 branch. Even the whitespace-only fix may have issues: after the rebase, the rebased commits may have incorrect indentation where new lines are added. So if you can come up with a bullet-proof git filter-branch invocation whose result doesn't need manual inspection, and can sell the additional work to those who actually need to do it, I don't have objections. Except perhaps that if/when master needs to be reformatted soon, I can't promise to have much time available for the C++ issues.

git filter-branch amounts to an interactive rebase that runs the filter before each new commit. So it is not likely to be a good solution for the public branches that will need to be merged.

#15 - 12/10/2011 09:25 AM - Roland Schulz

Regarding mode-lines (a discussion started on https://gerrit.gromacs.org/#change,319):
I agree with Teemu that it doesn't make much sense to treat Emacs special. Also it is a bit much to have a modelines for every editor in every file. An approach for the whole project seems better. We could accept a config file per editor. Such a directory approach seems to work with the main editors. E.g.:
http://code.google.com/p/lh-vim/source/browse/misc/trunk/plugin/local_vimrc.vim
http://www.emacswiki.org/emacs/DirectoryVariables
Eclipse (by default per directory)
So I suggest to remove all modelines and instead adding a config file per editor in the root folder.

#16 - 12/10/2011 10:40 AM - Teemu Murtola

Mark Abraham wrote:

> Doing steps 2, 3, 4 and 5 with git merge results in three merge commits per feature branch, one each from 2, 3 and 4. I've tested this with my qhop experiment. This is an increase of two on the minimum for a public branch. So I'm not sure where the multitude of merge commits comes from?

Well, there is three merges instead of zero for a rebase. I'm not sure if that counts as a multitude. :) But based on earlier discussion on, e.g., gmx-core, I think that the mindset has been to avoid merges as much as possible to keep an easy-to-follow history.

> git filter-branch amounts to an interactive rebase that runs the filter before each new commit. So it is not likely to be a good solution for the public branches that will need to be merged.

I think we are talking a bit past each other here. Based on the discussion referred to above, I was assuming that we would like to rebase/squash/whatever as many branches as possible when adding them to 4.6 (to make review easier), even if they were semi-public previously. If we instead do merging, the workflow that you propose is probably sufficient. Or one could probably, as an additional step, squash the merges together after the above workflow to have just a single merge from the feature branch to the main branch with all the effects included.

#17 - 12/10/2011 04:01 PM - Mark Abraham

Teemu Murtola wrote:

> Mark Abraham wrote:
>
> > Doing steps 2, 3, 4 and 5 with git merge results in three merge commits per feature branch, one each from 2, 3 and 4. I've tested this with my qhop experiment. This is an increase of two on the minimum for a public branch. So I'm not sure where the multitude of merge commits comes from?
>
> Well, there is three merges instead of zero for a rebase. I'm not sure if that counts as a multitude. :)

There has to be a merge for a public feature branch, unless you commit to trashing the branch... Trashing could well be best, of course :) The idea of doing interactive-rebase commit-squashing trashes the feature branch also, as you note below.

> But based on earlier discussion on, e.g., gmx-core, I think that the mindset has been to avoid merges as much as possible to keep an easy-to-follow history.

I'm all for that. I do expect resistance to the idea that you have to trash the public feature branch, based on incomplete understanding of git workflows. The feature branch is actually dead weight after the merge, except inasmuch as the overall merge/rebase process introduces bugs and one needs a "known good" version to compare against. So the pre-rebase feature branch should probably be archived somehow.

> > git filter-branch amounts to an interactive rebase that runs the filter before each new commit. So it is not likely to be a good solution for the public branches that will need to be merged.
>
> I think we are talking a bit past each other here. Based on the discussion referred to above, I was assuming that we would like to rebase/squash/whatever as many branches as possible when adding them to 4.6 (to make review easier), even if they were semi-public previously. If we instead do merging, the workflow that you propose is probably sufficient. Or one could probably, as an additional step, squash the merges together after the above workflow to have just a single merge from the feature branch to the main branch with all the effects included.

Good point. So that suggests the overall workflow, for either public or private branches should be more like

1. r46 was uncrustified and that commit tagged "uncrustify". Development on r46 then continued.

2. When it is time to merge qhop, archive a "known good" copy somewhere safe, check it out, do any remaining commit-squashing with interactive rebase, and then git rebase uncrustify~1. Resolve as normal. Tell anyone who needs to know that any public copies of this branch are now broken and they should delete their branches.

3. Use git filter-branch to run uncrustify on the feature branch. Details TBA. Either DIY, or upload a newly named branch to the GROMACS repo and

email Mark to get him to run it for you. Or maybe we use the format-patch and send-email git features. Whatever.

4. Now git rebase r46 to get the rest of the r46 development. Resolve as normal.

5. Switch to r46 and git rebase qhop - which will now be a fast-forward and so a legitimate operation on a public branch. Done.

I'll test this on qhop tomorrow. I wonder how this workflow copes with any existing merges of r46 into qhop?

#18 - 12/11/2011 12:08 AM - Roland Schulz

I tend to agree by now to Teemu's original opinion that this is not the correct time to do it. We currently have a lot of confusion of how to correct merge/rebase feature branches into 4.6. And we have a lot of them (two of them in gerrit for review). As long as it isn't sorted out how to best prepare the feature branches for review, we shouldn't make it more complicated than it already is.
I think we should look at it again as soon as the process of how to integrate feature branches has been settled.

This problem doesn't effect modelines. So we could do them now.

And we should focus on getting 4.6 out and than having an easier life with these issues.

#19 - 12/11/2011 06:28 AM - Mark Abraham

Roland Schulz wrote:

> This problem doesn't modelines. So we could do them now.

http://www.gromacs.org/Developer_Zone/Programming_Guide/Code_Formatting has instructions for configuring emacs in lieu of mode lines, and also for using it as a batch tool to reformat multiple new files.
Hopefully someone can had whatever's appropriate for Eclipse or Visual Studio, or whatever.

#20 - 01/14/2012 01:44 AM - Mark Abraham

Christoph Jungans suggests find . -type f -exec sed -i 's/[[:space:]]$//' {} \; for removing terminal white space. Recorded here for when it becomes useful.

#21 - 03/06/2012 09:52 AM - Mark Abraham

... and while we're there, change all the

int func();

to

int func(void);

etc.

#22 - 03/12/2012 08:19 PM - Teemu Murtola

Mark Abraham wrote:

> ... and while we're there, change all the
>
> int func();
>
> to
>
> int func(void);
>
> etc.

I think this should only be done for C code, where there actually is a semantic difference. For C++, func() and func(void) declarations work identically, and the former is very commonly used at least for methods in classes.

#23 - 04/01/2012 01:03 AM - Roland Schulz

- Target version set to 4.6

Setting target to 4.6. As soon as the last big features (i.e. free-energy and gpu) are merged this should be fine. And this would make it easier to fix bugs in the future.

#24 - 04/20/2012 05:29 AM - Mark Abraham

Another idea for posterity: see if it is possible to have some script parse new patches uploaded to gerrit and fix trivial whitespace by uploading a new

patch. The less developer time has to be spent mentioning and fixing trivia the better our code quality and rate of progress.

#25 - 04/20/2012 05:46 AM - Roland Schulz

This would be easy to do. We could just create a Jenkins jobs which is triggered by Gerrit. The Jenkins job would auto-fix all whitespace and upload it back to Gerrit.

BTW: We need to finalize the decision on uncrustify and how the config should be. Mark could you upload the config you have (both for C/C++)? I would suggest you add them to the source repository (e.g. admin) but releng is ok too. I would suggest you also upload a commit to Gerrit with uncrustify run over it. We are not ready to run it yet, but if we have a version in Gerrit we could discuss whether we like the uncrustify config. And we can comment on lines we think are misaligned.

#26 - 04/20/2012 06:22 AM - Teemu Murtola

I seem to recall that automatic formatting was discussed in some issue, but can't find it right now. Anyway, I'm not in principle against it, but please consider the following points in thinking how it should work:

1. If there is a misconfiguration, an automatic formatter would be very difficult to override, as it would simply undo you changes for each patch set that you upload.
2. If a developer uploads multiple changes simultaneously, how would the automatic formatter work? Would it reformat each change independently? In that case, the developer would need to do a lot of rebasing (rebase automatically generated commits on top of other automatically generated commits) to get the dependencies right. There is more work in this than in rebasing a local branch, since the automatically generated commits don't have convenient names like the local branch. An alternative would be that the automatic formatter first reformats change 1, then change 2 and rebases that on top of change 1, then change 3 and rebases that on top of change 2 etc., but this gets already quite complex.
3. An alternative of the above, consider if developer has locally multiple changes, but only uploads the first one or two for review. Again, they need to rebase their remaining local changes on top of an automatically generated commit.
4. For point 2, also consider that currently Jenkins jobs are triggered in an indeterminate order if multiple changes are uploaded simultaneously.

If Jenkins would simply vote -1 for any issues it finds (and clearly identify what those are, even better if it would provide a diff/patch), the above issues would be avoided. The developer would need to do a bit more work for single changes, but multiple simultaneous changes I think that there could actually be less work for the developer.

#27 - 04/20/2012 07:38 AM - Roland Schulz

If we want to only once clean-up the code the automatic formatter doesn't have to be perfect. But if we want to check formatting or automatic fix the code, it has to be perfect so that one never needs to manual fix the result. Because even if we only use the formatter for checking, we couldn't make manual fixes because then Jenkins would vote -1 (thinking the manual fix is not OK). That's why I asked Mark to upload the result of uncrustify, so that we can see whether the result is OK without any manual fixes. I think e.g. the initialization of t_pargs is difficult for most formatters.

I think the rebasing/mutlti-commit issue is not really a problem. The auto-fixed change should just be viewed as a suggestion. If the developer. Jenkins would only upload a new change if it isn't OK. Thus if you would want to do it manually (probably could idea in the described cases (e.g. multiple commits)) you can do so and just ignore Jenkins change. And if you did everything OK Jenkins wouldn't upload a fixed one again (because nothing is to fix). I think this would make it easy for single commits and wouldn't complicate it for more complex cases. But of course it would only work if no manual fixes to the formatter are required.

#28 - 06/23/2012 08:45 PM - Roland Schulz

As discussed there would be a big advantage to do that before 4.6 because it will make merging bug-fixes between 4.6 and 5.0 much easier. If so we should do this as soon as the CPU and GPU nbnxn kernels from Szilard/Berk are merged. Thus we would need to finalize deciding on the process very soon. Mark, do you plan to upload a suggested uncrustify configuration?

#29 - 06/25/2012 02:56 AM - Mark Abraham

Can do, but am on holidays this week. Will look at it next week.

#30 - 10/18/2012 07:06 AM - Teemu Murtola

I can help with parts of this (at least tweaking the uncrustify configuration file and adding cmake targets to easily run it), but I really would not like to duplicate work that Mark has already done.

Just a point to remember: this needs to be applied to 4.6 and master at roughly the same time, which means that also C++-specific indentation needs to be considered before it can be merged to 4.6.

#31 - 11/10/2012 07:30 PM - Roland Schulz

@Mark: Ping. Can you upload your work in progress?

#32 - 11/12/2012 12:21 AM - Mark Abraham

- File gromacs-uncrustify.cfg added

Here's the file I have lying around. Haven't looked at it in ages, but I hope it will save someone some work when someone can prioritise this :)

#33 - 11/12/2012 05:06 AM - Roland Schulz

Should we also do line wrapping to 80 characters automatically? Based on Marks config I tried to add that but it seems it often looks rather silly (both comments and long conditionals). If we don't want to do line-wrapping, the simpler Artistic Style might be a good option.

I think in most cases Mark's config gives OK indentation. Something I don't like is that it changes

```
    const char *fit[efNR + 1] =
        { NULL, "rot+trans", "translation", "none", NULL };
```

to

```
    const char *fit[efNR + 1] =
    { NULL, "rot+trans", "translation", "none", NULL };
```

And I couldn't find a configuration to change that.

#34 - 11/12/2012 07:18 AM - Roland Schulz

- File uncrustify.cfg added

Using a http://sourceforge.net/projects/universalindent/ I created a possible configuration which is attached.

#35 - 11/13/2012 12:59 AM - Roland Schulz

- File deleted (uncrustify.cfg)

#36 - 11/13/2012 03:23 AM - Roland Schulz

I uploaded a suggestion to https://gerrit.gromacs.org/#/c/1680
This gives everyone a chance to comment on the changes without having to install uncrustify.

I used

```
uncrustify --replace -c admin/uncrustify.cfg `find src -not -name 'nb_kernel_Elec*' -not -path '*/external/*'
-not -path '*src/gromacs/linearalgebra/*' -not -path 'src/contrib/*' -not -path '*nb_kernel_adress_c*' \( -nam
e '*.c' -o -name '*.cc' -o -name '*.h' -o -name '*.cpp' \)`
```

#37 - 12/28/2012 11:57 PM - Erik Lindahl

- Target version changed from 4.6 to 5.0

Changing target to 5.0, since we're not going to do large overhauls of whitespace in the very last minute before 4.6 the release.

#38 - 12/29/2012 12:06 AM - Roland Schulz

- Target version changed from 5.0 to 4.6.1

We don't need to do it before the 4.6 release. But if we don't fix the whitespaces also in the 4.6 branch, it is going to be difficult to merge (bugfix) changes from 4.6 to master. Thus changing target to 4.6.1.

#39 - 12/29/2012 09:07 PM - Erik Lindahl

Just a quick comment about line lengths.

I strongly subscribe to Linus Torvald's opinion that max-80-character lines really made sense with the terminals used in the 1970s, but today when we have modern editors that can handle much longer lines it is a stupid limitation.

The problem is that it encourages people to write shorter comments and use too short variable and function names just to fit things within 80 characters, or you end up splitting a logical unit over several lines to make it fit.

For readability (IMHO) it is much better if people use 30-character function names and long variables, and sometimes keep a logical unit together for readability, even if we use 120+ chars.

#40 - 12/29/2012 09:11 PM - Erik Lindahl

PS: If we want to introduce something that changes pretty much every line in every file, we need it either for 4.6.0 or it will have to wait for 5.0.

Roland, perhaps you could post a mail on gmx-core asking more people to comment on your RFC-patch, and if nobody complains Mark could apply this as the last patch we do before 4.6.0.

#41 - 12/30/2012 08:56 AM - Roland Schulz

Answering some of Teemu's points from Gerrit (moved here because redmine seems better for discussions):

> I think that automatic alignment of variables, equality signs, continuation lines etc. should be left out.

If we would not automatically align or not indent continuation lines, then any alignment(/indentation for continuation lines) would look bad for files which needed fixed indentation, because the manual alignment/indentation wouldn't fit the fixed new indentation. Thus I don't think this would be a good idea even if it would be supported by any code beautifier. Instead, if we don't like the result of the automatic alignment/continuation-indentation, then we should apply the automatic indentation only selectively. E.g. the initial fixup could exclude all of the new C++ files which don't require fixup.

> That bad alignment would be fine if it would be an opt-in option; I would personally like to use a configuration where I configure my editor to allow running uncrustify for just code fragments that I want to indent with these opt-in settings (very easy with vi), instead of using a blanket commit hook.

The main disadvantage of that strategy is that we need to make sure we apply this consistently. If people don't use it enough, we end up again with badly formated code and if some people simply apply it to all modified files (e.g. by git hook), then whenever those people touch any file, the manual alignment is lost.

> Writing stuff in the first place is also generally easier if things are aligned at tab stops.

uncrustify has an option align_on_tabstop. Should we use that?

#42 - 12/30/2012 12:24 PM - Teemu Murtola

Roland Schulz wrote:

> > I think that automatic alignment of variables, equality signs, continuation lines etc. should be left out.
>
> If we would not automatically align or not indent continuation lines, then any alignment(/indentation for continuation lines) would look bad for files which needed fixed indentation, because the manual alignment/indentation wouldn't fit the fixed new indentation. Thus I don't think this would be a good idea even if it would be supported by any code beautifier. Instead, if we don't like the result of the automatic alignment/continuation-indentation, then we should apply the automatic indentation only selectively. E.g. the initial fixup could exclude all of the new C++ files which don't require fixup.

I think you are quoting/answering this issue out of context. I don't have much to add to what I have written into gerrit, and don't want to repeat everything I said there, but here are still some thoughts. My point was that if we want to have, e.g., a Gerrit/Jenkins-enforced automatic clean-up, then we really should strive for minimal set of rules. I have hard time believing that there isn't a single code beautifier in the world that could leave non-leading whitespace on a line alone. True, there are cases where this may result in suboptimal alignment, but I think those are much rarer than the issues that are clearly visible in the RFC change or that I have pointed out in my comments in gerrit. For automatic clean-up of incoming commits, I have hard time believing that we need a beautifying algorithm that converts any kind of mess into perfectly readable code. Instead, it would only need to check (and possibly fix) a few issues, assuming that the incoming stuff is somehow reasonably formatted. Ideally, it should be trivial to write code that follows the guidelines the beautifier enforces, and it should be rare that the beautifier needs to step in (unless one wants to give the responsibility to it).

For the initial clean-up, we may well want to use a different set of rules, possibly only processing part of the files, but this is a separate discussion from the automatic clean-up.

> > That bad alignment would be fine if it would be an opt-in option; I would personally like to use a configuration where I configure my editor to allow running uncrustify for just code fragments that I want to indent with these opt-in settings (very easy with vi), instead of using a blanket commit hook.
>
> The main disadvantage of that strategy is that we need to make sure we apply this consistently. If people don't use it enough, we end up again with badly formated code and if some people simply apply it to all modified files (e.g. by git hook), then whenever those people touch any file, the manual alignment is lost.

It isn't opt-in if some people's actions can force anyways everyone to use it. For this approach to work, the "manual" part should only consist of rules that people don't object to even if they are not enforced. And as such, there should be no incentive to put them into a commit hook. But if people want to make it so that they don't need to (and *can't*) influence any whitespace in their code, then I guess I just have to live with it...

> > Writing stuff in the first place is also generally easier if things are aligned at tab stops.
>
> uncrustify has an option align_on_tabstop. Should we use that?

I don't know whether turning that on will make things better or worse globally... I think that all the options uncrustify has to offer for alignment are bad, so I will not be happy no matter which of these is chosen.

#43 - 12/30/2012 08:20 PM - Roland Schulz

Teemu Murtola wrote:

> Roland Schulz wrote:
>
> > I think that automatic alignment of variables, equality signs, continuation lines etc. should be left out.
>
> If we would not automatically align or not indent continuation lines, then any alignment(/indentation for continuation lines) would look bad for files which needed fixed indentation, because the manual alignment/indentation wouldn't fit the fixed new indentation. Thus I don't think this would be a good idea even if it would be supported by any code beautifier. Instead, if we don't like the result of the automatic alignment/continuation-indentation, then we should apply the automatic indentation only selectively. E.g. the initial fixup could exclude all of the new C++ files which don't require fixup.
>
> I think you are quoting/answering this issue out of context.

Sorry I didn't mean to.

> I don't have much to add to what I have written into gerrit, and don't want to repeat everything I said there, but here are still some thoughts. My point was that if we want to have, e.g., a Gerrit/Jenkins-enforced automatic clean-up, then we really should strive for minimal set of rules. I have hard time believing that there isn't a single code beautifier in the world that could leave non-leading whitespace on a line alone.

I looked again and couldn't find anything. I didn't check those from editors (e.g. the Eclipse CDT one is pretty good) because it would be hard to use for anyone not using that particular editor.

> True, there are cases where this may result in suboptimal alignment, but I think those are much rarer than the issues that are clearly visible in the RFC change or that I have pointed out in my comments in gerrit. For automatic clean-up of incoming commits, I have hard time believing that we need a beautifying algorithm that converts any kind of mess into perfectly readable code. Instead, it would only need to check (and possibly fix) a few issues, assuming that the incoming stuff is somehow reasonably formatted. Ideally, it should be trivial to write code that follows the guidelines the beautifier enforces, and it should be rare that the beautifier needs to step in (unless one wants to give the responsibility to it).

I opened two bugs which would resolve most of these issues as far as I can see Extra indent if aligning isn't possible and Finer control when to align. Given that two other bugs I filed yesterday ('using namespace' causes incorrect indention in following lines and Segmentation fault introduced by commit 2cb2833 are already fixed, maybe those can also be addressed soon.

> For the initial clean-up, we may well want to use a different set of rules, possibly only processing part of the files, but this is a separate discussion from the automatic clean-up.

We could use the same set of files for both the initial clean-up and the automatic clean-up. If we have a script with the file pattern of those files included in the automatic clean-up it would be used consistently, and we would avoid doing any automatic aligning on any files which are in average better manual than automatic. If we would use that approach, would you be happy with the current rules which then would only affect the C files?

#44 - 12/31/2012 01:53 PM - Teemu Murtola

Roland Schulz wrote:

> I looked again and couldn't find anything. I didn't check those from editors (e.g. the Eclipse CDT one is pretty good) because it would be hard to use for anyone not using that particular editor.

The first one I looked at (astyle, also mentioned here earlier), doesn't have any options for aligning struct members or assignment signs (or aligning anything, more or less). And it leaves the manually added spaces in these cases intact. It has some other issues though (ran it for the C++ code in the analysisdata subdirectory with options -oOCS -m0 -M80): similar problems as with uncrustify with typedef indents and function parameters on a next line, and bad indentation of class declarations with base classes on multiple lines. These are still possible to work around (by changing the convention used), unlike uncrustify's alignment.

> I opened two bugs which would resolve most of these issues as far as I can see Extra indent if aligning isn't possible and Finer control when to align.

If there is an option to disable all alignment so that uncrustify doesn't touch it, I'm fine with minor issues that may be worked around by changing the code.

> We could use the same set of files for both the initial clean-up and the automatic clean-up. If we have a script with the file pattern of those files included in the automatic clean-up it would be used consistently, and we would avoid doing any automatic aligning on any files which are in average better manual than automatic. If we would use that approach, would you be happy with the current rules which then would only affect the C files?

I don't think that this approach addresses the needs for the automatic clean-up. My impression is that people want to be able to write code in any file, and not worry about the indentation, spaces/tabs, or trailing whitespace (and not burden code reviewers with that task either). If we start excluding more and more files from the clean-up, this doesn't work. Also, my complaint about the alignment is of more general nature: do we really want to enforce rules like "you are not *allowed* to align struct/enum definitions if you comment them, but must align them if you don't have long comments nor empty lines nor ...?" (I can make a lot of these examples, but hopefully this clarifies my point.)

#45 - 12/31/2012 11:43 PM - Roland Schulz

- Target version changed from 4.6.1 to 4.6

> If we start excluding more and more files from the clean-up, this doesn't work. Also, my complaint about the alignment is of more general nature: do we really want to enforce rules like "you are not allowed to align struct/enum definitions if you comment them, but must align them if you don't have long comments nor empty lines nor ...?"

I think this is a bit unfair, given that the rule can be stated much simpler as: you must only align adjacent lines.

I did try astyle too, but found that it had changed about as much in your well formatted code without good reason. But if the concern is less the amount of changes but the predictability, then I see how it might be a better option.
If we choose astyle I would suggest we use -oOCS -m1 -M60 -jbcH. I think particularly "-jbc" is very important for cleaning up the badly formatted code. And we need to decide what to do with the extern "C" in the headers, because the last released version of astyle 2.02.1 indents all inside. The options:
- add the #elif 0 { trick everywhere
- replace with EXTERN_C_OPEN (with once defining EXTERN_C_OPEN properly in one header)
- use the svn version which has this problem fixed

I'll upload to Gerrit a version with the suggested options and the svn version.

#46 - 01/01/2013 03:54 AM - Roland Schulz

Two things I much prefer of the uncrustify result:

- The alignment of comments. I uploaded a new commit with uncrustify with it enabled again. I disabled it before because it aligned the namespace comment for no reason, but that isn't a big deal. And in some files (e.g. tools/gmx_order.c) it looks extremely ugly without the comments alignment fixed.
- The code guidelines require spaces around some operators. While even with uncrustify we are not adding all, some of the important ones are added and I think this makes it quite a bit better.

#47 - 01/01/2013 12:19 PM - Teemu Murtola

Roland Schulz wrote:

> > If we start excluding more and more files from the clean-up, this doesn't work. Also, my complaint about the alignment is of more general nature: do we really want to enforce rules like "you are not allowed to align struct/enum definitions if you comment them, but must align them if you don't have long comments nor empty lines nor ...?"
>
> I think this is a bit unfair, given that the rule can be stated much simpler as: you must only align adjacent lines.

Well, the technical content of the rule can be written in a number of ways (although the current rule is several times more complex than what you state). But it still amounts to what I said: one is *forced* to remove any manual alignment if one adds a comment or an empty line somewhere. Even if the technical rule does not care what there is in between the lines to be aligned, it *still* means that one is not *allowed* to combine manual alignment with commented code. That deceptively simple technical description of the rule also hides the fact that if you have a stretch of aligned something, and then you add a comment or an empty line in between, you may need to realign the whole block...

I don't really care whether we want to force alignment somewhere. But I think that we should not force some arbitrary rules on where *not* to align. I don't know how to make my point any more clear. If we decide on using uncrustify and there is no option to ignore alignment, then I suggest that we either disable it completely, or set both an infinite span and an infinite threshold. Those are the only three options that don't lead to this kind of issues. We may want to pick a different alternative for different cases, though; e.g., align struct/class members through the whole file, but not assignments. For assignments, alignment within a single block could also be an option, but probably not supported by uncrustify.

> I did try astyle too, but found that it had changed about as much in your well formatted code without good reason. But if the concern is less the amount of changes but the predictability, then I see how it might be a better option.

I agree that it changes about as much as uncrustify (and mostly in the same places that uncrustify did with the first set of options we had). And in general, I like uncrustify's configurability and the style it produces more, but there is the alignment issue.

> If we choose astyle I would suggest we use -oOCS -m1 -M60 -jbcH. I think particularly "-jbc" is very important for cleaning up the badly formatted code.

Yes, I didn't put those in my example invocation since those did not affect my code. And also note that -oO causes some code to stay as it is that we would probably want to change in the initial clean-up. But I think it is overkill to force every accessor in C++ to take four lines instead of using one-liners.

> And we need to decide what to do with the extern "C" in the headers, because the last released version of astyle 2.02.1 indents all inside. The options:
> - add the #elif 0 { trick everywhere
> - replace with EXTERN_C_OPEN (with once defining EXTERN_C_OPEN properly in one header)
> - use the svn version which has this problem fixed

Using the svn version sounds like the best option of these, although that will also make it a bit difficult for anyone to use it as commit hook.

#48 - 01/01/2013 12:26 PM - Teemu Murtola

Roland Schulz wrote:

> Two things I much prefer of the uncrustify result:
>
> - The alignment of comments. I uploaded a new commit with uncrustify with it enabled again. I disabled it before because it aligned the namespace comment for no reason, but that isn't a big deal. And in some files (e.g. tools/gmx_order.c) it looks extremely ugly without the comments alignment fixed.
> - The code guidelines require spaces around some operators. While even with uncrustify we are not adding all, some of the important ones are added and I think this makes it quite a bit better.

Both of these are mostly relevant for the initial cleanup (adding the space around operators could be helpful later as well). And there is nothing preventing us running the code first through uncrustify in the initial cleanup (or using any combination of tools or multiple runs), as long as the end result is such that the automatic alignment rules no longer change it. And astyle doesn't touch alignment of comments. astyle has an -p option to add spaces around operators, but haven't tried whether that causes some issues.

For the namespace issue, some combination of align_right_cmt_mix and align_right_cmt_gap could help to exclude the namespace case (and setting the gap > 1 could also help in not forcing everything to get aligned).

#49 - 01/01/2013 12:28 PM - Teemu Murtola

- Tracker changed from Bug to Task

#50 - 01/01/2013 10:13 PM - Roland Schulz

I made a small change to uncrustify which disables all tightening (=removing of spaces) when aligning. I updated the gerrit commit. The modified version of uncrustify is at: https://github.com/rolandschulz/uncrustify. How do you like this solution?

#51 - 01/01/2013 10:37 PM - Roland Schulz

The "\code" issue (120) should also be easy to fix (in src/tokenize.cpp add "\code" the same as UNCRUSTIFY_OFF_TEXT).

#52 - 01/02/2013 06:34 AM - Teemu Murtola

Roland Schulz wrote:

> I made a small change to uncrustify which disables all tightening (=removing of spaces) when aligning. I updated the gerrit commit. The modified version of uncrustify is at: https://github.com/rolandschulz/uncrustify. How do you like this solution?

I like the end result (and the general level of control that remains for people writing the code) much more. The remaining issues I can live with, although there are probably still some things that we may want to tweak in the configuration. And we can also change the code to fix some issues: e.g., for the \code issue, we can simply indent the examples manually. The only reason why I didn't do that originally (it would look much nicer) is that the version of Doxygen I had when writing those did not strip the whitespaces.

#53 - 01/02/2013 07:29 AM - Roland Schulz

With that change is uncrustify your first choice or would you still prefer astyle? What remaining issues would like to have fixed before we do the initial clean-up?

#54 - 01/04/2013 01:17 PM - Teemu Murtola

Roland Schulz wrote:

> With that change is uncrustify your first choice or would you still prefer astyle? What remaining issues would like to have fixed before we do the initial clean-up?

Uncrustify seems better after this. If there is no option in uncrustify to remove tabs within comments, we may want to run astyle with the -c option as part of the initial clean-up (if it fixes that, not sure).

In the uncrustify configuration, I'd consider at least the following changes:

- indent_switch_case=4: it appears that this is not consistently used in the code, but the only harm I can see that some parts may become more indented.
- nl_assign_brace=ignore: as we in many other cases force the braces to a line of their own, this forcing one to remove the brace here seems a bit out of place. But I agree that =remove makes some places more consistent (like names.c). Another option is to set it as add.
- Some of the sp_* and nl_* options that are currently at ignore might be better as add at least for the initial clean-up, but that would require a bit of effort to find out which of these won't cause any issues. Non-exhaustive list of thoughts follows:
  - Adding space after if, while, for etc. before the parentheses should be relatively safe (sp_before_sparen=add), and would make the code more consistent.
  - Possibly also sp_inside_sparen=remove.
  - The same for nl_switch_brace, nl_do_brace and friends.
  - Also, if we want to force blocks to start with a brace on its own line, several C++-specific nl_* options should be enabled (at least related to exception handling and possibly also nl_class_brace and nl_namespace_brace).
  - Don't know if we want to force nl_struct_brace=add, and the same for enums and unions.
- nl_end_of_file=force and nl_end_of_file_min=1 would normalize newlines at the end of the file.
- Consider changing nl_after_brace_close=add to ignore or setting nl_brace_struct_var=remove (if that fixes the issue in the selection code that I commented on in gerrit, patch set 6).

Also, some comments for the alignment:

- I'm not sure how your change for uncrustify works, but it may be that now the meaning for the various *_thresh is "max. number of spaces that uncrustify will add to align". As such, it may be that a single run of uncrustify adds some space to align some variables, and the next run then detects more things to be aligned (that are now within the threshold after the first run) and adds even more space. So at least for automatic clean-up, we probably should set the threshold to infinity.
- Otherwise, I'm fine with the current alignment settings for the initial clean-up. For the automatic clean-up, we may want to reduce the spans for anything but align_var_struct_span to 1 or disable them to avoid potential surprises. But in general, the end result for the initial clean-up could be better with the current alignment settings.

Finally, as a minor comment, for the final clean-up commit, generated files that we don't have full control over should be excluded. This means at least the generated selection parser files (parser.cpp, parser.h, scanner.cpp, scanner_flex.h). Possibly the kernels (and their preprocessor source field) should be included, although it is good that they are not included yet as they would just generate noise in gerrit. Don't know about thread_mpi.

And another final comment, I agree with Erik that we shouldn't force any particular line length. We could still have a guideline something like "try to keep lines under 80 characters if it doesn't harm readability", as code that has a lot of very long lines is often hard to read. Even if using long identifiers, splitting long lines manually into logical chunks often makes it a lot better. It's not a coincidence that, e.g., latex typesets around 60-70 characters per line.

#55 - 01/04/2013 09:14 PM - Roland Schulz

Teemu Murtola wrote:

> In the uncrustify configuration, I'd consider at least the following changes:

I'm happy with all of those. If you have time to do that it would be great. Otherwise, I'll try to do it soon. I think we should make sure we still get this into 4.6.

> Also, some comments for the alignment:
>
> - I'm not sure how your change for uncrustify works, but it may be that now the meaning for the various *_thresh is "max. number of spaces that uncrustify will add to align". As such, it may be that a single run of uncrustify adds some space to align some variables, and the next run then detects more things to be aligned (that are now within the threshold after the first run) and adds even more space.

Both with and without my change their are a few changes when rerunning uncrustify. But after 3 times no further changes happens. So I suggest the initial cleanup uses 3 runs (+ a 4th run to check that further runs don't cause further changes).

> So at least for automatic clean-up, we probably should set the threshold to infinity.

I would suggest not to do that so that we don't align to much.

> - Otherwise, I'm fine with the current alignment settings for the initial clean-up. For the automatic clean-up, we may want to reduce the spans for anything but align_var_struct_span to 1 or disable them to avoid potential surprises. But in general, the end result for the initial clean-up could be better with the current alignment settings.

That's a good idea.

> Finally, as a minor comment, for the final clean-up commit, generated files that we don't have full control over should be excluded. This means at least the generated selection parser files (parser.cpp, parser.h, scanner.cpp, scanner_flex.h). Possibly the kernels (and their preprocessor source field) should be included, although it is good that they are not included yet as they would just generate noise in gerrit. Don't know about thread_mpi.

> And another final comment, I agree with Erik that we shouldn't force any particular line length. We could still have a guideline something like "try to keep lines under 80 characters if it doesn't harm readability", as code that has a lot of very long lines is often hard to read. Even if using long identifiers, splitting long lines manually into logical chunks often makes it a lot better. It's not a coincidence that, e.g., latex typesets around 60-70 characters per line.

I agree with all that.

#56 - 01/06/2013 11:10 PM - Erik Lindahl

- Status changed from New to In Progress

#57 - 01/09/2013 02:51 AM - Erik Lindahl

I did a little bit of comparison with astyle (that I've used this far), but uncrustify seems to be at least as good, and possibly more configurable.

When it comes to special files (for the kernels), I think the best solution would be to apply the formatting only to the template (not the generated output), which will then mean the kernels will be correct next time they are generated. Otherwise we'll end up in a format war where the kernels will yet again be incorrect every time they are regenerated.

Did you settle on a decision for the C++ header blocks?

Altogether, I think this is good enough now to be applied as one of the very last patches before making the release, and then directly merging to 5.0...

#58 - 01/09/2013 04:27 AM - Roland Schulz

Erik Lindahl wrote:

> Did you settle on a decision for the C++ header blocks?

Do you mean the 'extern "C"'? This was only a problem with astyle. Uncrustify does it automatically correctly.

#59 - 01/09/2013 01:04 PM - Erik Lindahl

OK, that great. However, there's a related problem: Even if uncrustify does it automatically, many editors we use don't, and then we're going to end up with incorrect indentation.

The obvious solution is to manually add either the macro solution or the ifdef when creating new files (which I guess does not have to be part of uncrustify), but perhaps we should standardize on one of them?

#60 - 01/09/2013 04:15 PM - Roland Schulz

We haven't decided but my impression was that many (e.g. Szilard, Mark) are in favor of running uncrustify automatically (either from Jenkins or git hook). That way the indentation of the different editors people use wouldn't matter anymore because it would always be fixed by uncrustify. So if we decide to do that we wouldn't need the marco or the ifdef.

#61 - 01/12/2013 12:31 AM - Erik Lindahl

I think we still do - it will get pretty irritating when the editor wants to change the indentation any time you hit tab, even if jenkins will revert it later.

I'm all for the macro since that's short and sweet.

#62 - 01/12/2013 01:01 PM - Teemu Murtola

I agree with Erik that even if we have automatic formatting, we should strive to make it as easy as possible to write correct (or close-to-correct) code that will not be changed by uncrustify (and vice versa, that editors don't over-eagerly change the indentation produced by uncrustify). That probably means also tuning the uncrustify configuration after we get a bit of experience on what people find irritating. But I'm not sure we want to go the path of adding more and more hacks into the code just to please some editors.

The extern "C" case is one where we don't indent after a brace, but in C++ code, namespaces are another, and those are much more common (often occurring several times in each and every file). Is it really so that we want to add macros for such standard C++ constructs (and force everyone to use them throughout the code), just to please some editors? Isn't it possible to configure those editors to be a bit less eager in reformatting everything? I know that, e.g., Visual Studio may by default reformat the whole block after typing a closing brace, but it's simply impossible to please every possible configuration of every possible editor out there. Even if we fix these opening braces with some preprocessor magic, where is this then going to end? Do we add such hacks everywhere where someone complains that their editor by default doesn't produce exactly the same indentation as uncrustify?

For vim, the default configuration is such that this issue only affects the first declaration after the opening brace. Reindenting that manually isn't really that much work. I haven't had any problems with this or other cases where I may need to adjust the indentation manually. And I've been perfectly capable of producing code where perhaps 1% of the lines gets changed by uncrustify (and probably less if we go with less stringent alignment criteria for the automatic reformatting than for this batch work).

PS. I'm a bit pointed on purpose; I'm all for a compromise that doesn't make anyone's life too hard. But possibility of changing the configuration of the editors people use should also be on the table...

#63 - 01/14/2013 08:41 PM - Erik Lindahl

- Priority changed from Normal to High

Marking as "high" priority, just to make sure nobody pushes this to 4.6.1 by mistake. Together with the patch touching all heads, this should be the very last stuff to go in before the release.

#64 - 01/14/2013 09:32 PM - Mark Abraham

Yep, noted. Can Roland please post his latest uncrustify invocation so we're all on the same page?

A workflow of

0. Merge r46 up to master now-ish (and ideally right before step 1)
1. Uncrustify r46 right before the version bump
2. Uncrustify master
3. Merge r46 up to master

seems workable. If there's issues with an ugly merge commit (which I doubt) then we have time to play with 2 and 3.

#65 - 01/14/2013 10:09 PM - Roland Schulz

The kernel templates cannot be easily auto-formatted because of the extra parenthesis for template variables. So I suggest to leave out both the template and the kernels for now.

Marks workflow sounds good. I don't think it is critically to do 0 before 1. One can always merge an earlier version of r46, so step 0 can be done after step 1.

Required to run: https://github.com/rolandschulz/uncrustify/tree/gromacs, https://gerrit.gromacs.org/#/c/1680/10/admin/uncrustify.cfg

The command is:
uncrustify --replace -c admin/uncrustify.cfg `find src -not -name 'nb_kernel_Elec*' -not -path '*/external/*' -not -path '*src/gromacs/linearalgebra/*' -not -path 'src/contrib/*' -not -path '*nb_kernel_adress_c*' -not -name 'parser.*' -not -name scanner.cpp -not -name scanner_flex.h \(-name '*.c' -o -name '*.cc' -o -name '*.h' -o -name '*.cpp' \)`

One needs to run it 3 times.

#66 - 01/16/2013 07:59 PM - Teemu Murtola

Just as a comment for potentially reducing extra work: for 4.6, the scanner.cpp should be replaced with scanner.c in Roland's command (a few other replacements are also necessary for the paths). I would also suggest using --no-backup instead of --replace to avoid creating a mass of back-up files.

#67 - 01/23/2013 04:53 PM - Mark Abraham

Update: you want to check out the gromacs branch from Roland's git repo (I found master was checked out by default).

I believe the command line Roland used was close to:

uncrustify --no-backup -c admin/uncrustify.cfg `find src -not -name 'nb_kernel_Elec*' -not -path '*/external/*' -not -path '*src/gromacs/gmx_lapack/*' -not -path '*src/gromacs/gmx_blas/*' -not -path 'src/contrib/*' -not -path '*nb_kernel_adress_c*' -not -name 'parser.*' -not -name scanner.c -not -name scanner_flex.h \(-name '*.c' -o -name '*.cc' -o -name '*.h' -o -name '*.cpp' \)`

Note that this assumes a file layout suitable for master branch - remove some "gromacs" strings to suit release-4-6.

#68 - 01/23/2013 05:10 PM - Roland Schulz

My change to uncrustify has been merged into the official version. When Teemu's patch is in too, we can use the official version.

We should decide whether we want to recommend/require people to install a git commit hook and/or whether Jenkins should run uncrustify for verification/fixing.

#69 - 01/23/2013 07:16 PM - Mark Abraham

Roland Schulz wrote:

> My change to uncrustify has been merged into the official version. When Teemu's patch is in too, we can use the official version.

Great.

> We should decide whether we want to recommend/require people to install a git commit hook and/or whether Jenkins should run uncrustify for verification/fixing.

Yeah. I think making the former available and doing the latter is the most sound option. If people want to run a manual uncrustify, or opt in to use the

commit hook, then that's fine and the automated uncrustify will be a null-op. This will mean those people can have a workflow of local editing, uncrustify, upload and if there's a bug to fix from Jenkins verification failure and/or review, their local repo might still match what's in gerrit. Or, if people want to upload random crap or avoid installing software, they will have to do revisions on their crappy version (which will get re-uncrustified when they upload) or download the gerrit version before revising.

#70 - 01/23/2013 07:38 PM - Roland Schulz

The only disadvantage I see with that is we might increase the load on Jenkins. If we run our current jobs and the new uncrustify job in parallel, and the uncrustify job uploads a new patch because something changed, then we would have 2 patches getting verified, potentially doubling the load on Jenkins. It might be not an issue if only very patches needs to be fixed, or it could be improved in Jenkins by either running uncrustify first (but that would delay the other tests) or interrupt the other tests if uncrustify found any problems. The question would be if we need to worry about that at the beginning or whether we hope most patches won't need fixing and simply monitor the load.

#71 - 01/23/2013 07:48 PM - Mark Abraham

Something's different about our setups, Roland. I'm rebasing up some BlueGene/P kernel stuff for 4.6.1, so I need to rebase over the uncrustify, which is best done by uncrustifying my squashed commit just before the rebase over the uncrustify commit.

In src/tools/gmx_trjcat.c, the command in comment 67 produces

```
        fprintf(
            stderr,
            "          File             Current start (%s)  New start (%s)\n"
            "------------------------------------------------------\n",
            output_env_get_time_unit(oenv), output_env_get_time_unit(oenv));
```

rather than indented to the level of the parentheses. I see similar behaviour in a handful of other places, too. Any ideas?

#72 - 01/23/2013 07:54 PM - Teemu Murtola

I suspect that initially, most patches will actually need fixing unless people immediately start using a commit hook as well. There is an element of unpredictability/complexity in what uncrustify does (and I'm sure that some options could be improved to make it more consistent), so initially it will be difficult to write code that follows all the rules that get enforced. But I agree with Mark that if we want to have a commit hook approach, then it would be best enforced on the server side one way or another, because otherwise things will get messy if the commit hook starts to reformat parts of the files completely unrelated to the change at hand.

For the commit hook, I would split the reformatting part into a separate script that can be applied also independently to the working tree, as that would probably be useful in more complex git workflows than simple write - commit-all - push - forget.

If done cleverly (only reformatting files that actually changed), I don't think that uncrustify should take more than a few seconds to run in most cases. If it takes longer, it isn't very practical for the pre-commit hook either.

I can try to polish my patch to be able to submit it upstream, but it requires 1) fixing a completely unrelated problem in uncrustify to make the tests pass, and/or 2) rethinking/evaluating whether it would actually be better to make indent_continue more flexible, either in addition to my change or instead of it. So it may take some time...

The behavior that Mark is seeing seems to come from cases where my patch has an effect (it doubles the indent for code like this), so it may be related to that.

#73 - 01/23/2013 07:55 PM - Mark Abraham

Roland Schulz wrote:

> The only disadvantage I see with that is we might increase the load on Jenkins. If we run our current jobs and the new uncrustify job in parallel, and the uncrustify job uploads a new patch because something changed, then we would have 2 patches getting verified, potentially doubling the load on Jenkins. It might be not an issue if only very patches needs to be fixed, or it could be improved in Jenkins by either running uncrustify first (but that would delay the other tests) or interrupt the other tests if uncrustify found any problems. The question would be if we need to worry about that at the beginning or whether we hope most patches won't need fixing and simply monitor the load.

I envisaged gerrit running uncrustify via a **gerrit** hook on the set of files touched by the commit (modulo files it shouldn't touch), before Jenkins sees anything. That seems very lightweight to me.

#74 - 01/23/2013 08:02 PM - Mark Abraham

Thanks for the hint, Teemu. It seems I had compiled from the gromacs uncrustify branch (i.e. with Teemu's commit) but not installed it. Sorry for the spam.

#75 - 01/23/2013 08:03 PM - Mark Abraham

uncrustify --no-backup -c admin/uncrustify.cfg `find src -not -name 'nb_kernel_Elec*' -not -path '*/external/*' -not -path '*gmx_lapack/*' -not -path '*gmx_blas/*' -not -path 'src/contrib/*' -not -path '*nb_kernel_adress_c*' -not -name 'parser.*' -not -name scanner.c -not -name scanner_flex.h \(-name '*.c' -o -name '*.cc' -o -name '*.h' -o -name '*.cpp' \)`

is a more flexible command - by omitting "src/gromacs", it works on both release-4-6 and master (didn't actually test the latter).

#76 - 01/23/2013 08:11 PM - Mark Abraham

Teemu Murtola wrote:

> I suspect that initially, most patches will actually need fixing unless people immediately start using a commit hook as well. There is an element of unpredictability/complexity in what uncrustify does (and I'm sure that some options could be improved to make it more consistent), so initially it will be difficult to write code that follows all the rules that get enforced. But I agree with Mark that if we want to have a commit hook approach, then it would be best enforced on the server side one way or another, because otherwise things will get messy if the commit hook starts to reformat parts of the files completely unrelated to the change at hand.

Agreed. Also, we do not want to require all developers to install a particular piece of software, particularly if its a non-standard form of that software. That would be a barrier to participation from casual contributors that we do not need to impose.

> For the commit hook, I would split the reformatting part into a separate script that can be applied also independently to the working tree, as that would probably be useful in more complex git workflows than simple write - commit-all - push - forget.

Indeed. I'd suggest making available a pre-commit hook that just calls that script, so that people can call .git/hooks/uncrustify.sh and know they're applying the standard solution (unless we've gone and changed it on them :-S)

> If done cleverly (only reformatting files that actually changed), I don't think that uncrustify should take more than a few seconds to run in most cases. If it takes longer, it isn't very practical for the pre-commit hook either.
>
> I can try to polish my patch to be able to submit it upstream, but it requires 1) fixing a completely unrelated problem in uncrustify to make the tests pass, and/or 2) rethinking/evaluating whether it would actually be better to make indent_continue more flexible, either in addition to my change or instead of it. So it may take some time...
>
> The behavior that Mark is seeing seems to come from cases where my patch has an effect (it doubles the indent for code like this), so it may be related to that.

It seems so. For the moment, we can make an uncrustify tarball available for GROMACS use that has your fix - people have to opt in whether or not it's "standard" uncrustify. And use the same tarball to build the code that gerrit runs.

#77 - 01/23/2013 08:19 PM - Teemu Murtola

Mark Abraham wrote:

> > For the commit hook, I would split the reformatting part into a separate script that can be applied also independently to the working tree, as that would probably be useful in more complex git workflows than simple write - commit-all - push - forget.
>
> Indeed. I'd suggest making available a pre-commit hook that just calls that script, so that people can call .git/hooks/uncrustify.sh and know they're applying the standard solution (unless we've gone and changed it on them :-S)

The pre-commit hook needs to do a bit different thing than what normal users would expect the script to do (because it must deal with stuff like partial commits from a dirty working tree etc.), so we may need to make it a bit more complex to cater for both use cases. But in general, yes; people should be able to run this script and then do git commit -a and see no further changes.

#78 - 01/23/2013 09:22 PM - Roland Schulz

Mark Abraham wrote:

> I envisaged gerrit running uncrustify via a **gerrit** hook on the set of files touched by the commit (modulo files it shouldn't touch), before Jenkins sees anything. That seems very lightweight to me.

Gerrit doesn't have any pre-... hook. Only patchset-created hook could be used. So the patch would be accepted and then if a anything changes a new patcheset would be created. This would mean Jenkins would see both the original and the changed one. And unless we do something extra, it would verify both. Also for security reasons I would suggest not to run uncrustify on the gerrit server but instead by Jenkins. We really don't want the Gerrit server to be compromised. And I assume it would be very easy to find a buffer overflow in uncrustify which could be used to upload a patch which triggers it.

#79 - 01/24/2013 09:50 PM - Mark Abraham

Roland Schulz wrote:

> Mark Abraham wrote:
>
> > I envisaged gerrit running uncrustify via a **gerrit** hook on the set of files touched by the commit (modulo files it shouldn't touch), before Jenkins sees anything. That seems very lightweight to me.
>
> Gerrit doesn't have any pre-... hook. Only patchset-created hook could be used. So the patch would be accepted and then if a anything changes

> a new patcheset would be created. This would mean Jenkins would see both the original and the changed one. And unless we do something extra, it would verify both. Also for security reasons I would suggest not to run uncrustify on the gerrit server but instead by Jenkins. We really don't want the Gerrit server to be compromised. And I assume it would be very easy to find a buffer overflow in uncrustify which could be used to upload a patch which triggers it.

Bugger, I thought there was already a pre-patchset hook in gerrit. That's a PITA.

It seems that Jenkins uses a plugin that lets it poll gerrit, rather than having gerrit signal Jenkins via a post-patchset hook. So we can't force uncrustify to run just before Jenkins.

Maybe a post-patchset gerrit hook can terminate Jenkins runs if it detects that uncrustify needs to change the code?

#80 - 01/24/2013 11:05 PM - Roland Schulz

- Priority changed from High to Normal

Mark Abraham wrote:

> It seems that Jenkins uses a plugin that lets it poll gerrit, rather than having gerrit signal Jenkins via a post-patchset hook. So we can't force uncrustify to run just before Jenkins.

No we don't use polling. We use the Gerrit+Trigger plugin. It uses gerrit stream-events to get notified.

> Maybe a post-patchset gerrit hook can terminate Jenkins runs if it detects that uncrustify needs to change the code?

Again I wouldn't do it in gerrit, because of the security issue. But we could cancel the one Jenkins job from the other Jenkins job. It is a bit tricky because several of each could be running, so one would need to query the correct job based on the changeset number. Not sure how easy that is.

I found another solution: Gerrit+Trigger provides an option "Build Current Patches Only". It cancels previous builds of the same change. If a new patchset is uploaded the build of the previous patchset of the same change is automatically canceled. I just activated this, because it is anyhow useful. If we notice any problems we can undo it again. If it works fine this would solve this issue, because as soon as the uncrustify job uploads a new patch, the previous job would be automatically canceled.

#81 - 02/14/2013 02:31 PM - Teemu Murtola

Some observations from experimenting with pre-commit hook scripts (with the script in https://gerrit.gromacs.org/#/c/2155/).

- Nearly every commit that I tried it on required at least some reformatting with the current configuration. Mostly this was alignment issues, and the code was written before the uncrustification was applied, but I still think this highlights how often the reformatting needs to run.
- pre-commit hooks are *not* run for commits created implicitly by git rebase --continue. Didn't try extensively different rebasing approaches, but this means that for rebases, one potentially needs to manually run the script for each commit.

#82 - 02/20/2013 04:43 PM - Mark Abraham

Berk noted that we need to consider the question of indenting labels (e.g. case: in C/C++) and access specifiers (public/private/protected). These can be managed separately in uncrustify.

Currently uncrustify uses indent_case_switch=1, which produces

```
switch(blah)
{
    case 1:
        code_here();
}
```

I like the general idea of indentation reflecting a scope change (which makes it easy to find the end of a block). For case statements, that suggests indent_switch_case=0. I don't care too much, because they don't get nested much. (IIRC case statements do start their own scope blocks.)

If we might get nested classes, then a whole indent level for access specifiers seems a bit heavy to me. Spamming the upper indent level with access specifiers that don't strictly change scope is not too palatable either. There are styles that use a partial indent for access specifiers (I'd suggest 2 spaces), so that there's a visual clue for changes of access, but that does not affect the visual clue for change of scope. That corresponds to indent_access_spec=-2 and indent_access_body=false.

Thoughts?

#83 - 02/20/2013 08:37 PM - Teemu Murtola

Mark Abraham wrote:

> Berk noted that we need to consider the question of indenting labels (e.g. case: in C/C++) and access specifiers (public/private/protected). These can be managed separately in uncrustify.

Just to clarify, his point was that he prefers the way the emacs "stroustrup" mode works, and that happens to not indent the case labels. Don't know what it does with access specifiers, as I haven't used emacs in years. And since I don't know what the stroustrup style exactly does, very likely the code written by me does not follow all the rules there, but that code was the only easy reference for me to use when judging the uncrustify result.

> I like the general idea of indentation reflecting a scope change (which makes it easy to find the end of a block). For case statements, that suggests indent_switch_case=0. I don't care too much, because they don't get nested much. (IIRC case statements do start their own scope blocks.)

The reason why I picked indent_switch_case=4 was that I think that a rule "everything within a brace gets indented" is the most straightforward, and the extra indent can only improve readability. Both styles (either indented or not indented case labels) were mixed throughout the code. Just based on a quick look, I couldn't judge which one was used more.

(PS. Case statements do not, by themselves, start a new block. One has to add an extra set of braces if one wants to, e.g., declare variables. Also for this reason, I find indent_switch_case=4 more clear: the following is more readable than pulling the whole switch contents left one indent level (this example is what the current uncrustify configuration produces):)

```
switch (condition)
{
    case 1:
    {
        int a = 2;
        ...
        break;
    }
    case 2:
        ...
        break;
}
```

> If we might get nested classes, then a whole indent level for access specifiers seems a bit heavy to me. Spamming the upper indent level with access specifiers that don't strictly change scope is not too palatable either. There are styles that use a partial indent for access specifiers (I'd suggest 2 spaces), so that there's a visual clue for changes of access, but that does not affect the visual clue for change of scope. That corresponds to indent_access_spec=-2 and indent_access_body=false.

As above, this goes with the simple rule that everything within braces gets indented. I don't see how a partial indent would improve things, as it will be harder to write without some custom configuration in an editor. I haven't been bothered by the limitation of having 4 characters more indent within class declarations (and thus 4 characters less before hitting the 80 char limit). Partial indent also probably doesn't correspond to any editor's default style (for reference, the current one is more or less the default on vi, but is also my personal preference from times when I did not yet use vi). If nested classes would get indented too much, it is always possible to move their definition outside the main class to get it start back from indentation level zero.

> Thoughts?

The above said, I don't really have any hard opinions on these things. If people generally want some other style, then that's fine by me. But we should try to keep things as simple as possible such that it is possible to easily configure editors that people use to actually follow that style.

#84 - 02/20/2013 10:21 PM - Berk Hess

I guess almost no one has strong opinions on this (neither do I). But it might be useful to come up with a simple, clear set of rules, such that the code people write doesn't need to be reformatted.

#85 - 02/20/2013 10:42 PM - Mark Abraham

Teemu Murtola wrote:

> Mark Abraham wrote:
>
> > Berk noted that we need to consider the question of indenting labels (e.g. case: in C/C++) and access specifiers (public/private/protected). These can be managed separately in uncrustify.
>
> Just to clarify, his point was that he prefers the way the emacs "stroustrup" mode works, and that happens to not indent the case labels. Don't know what it does with access specifiers, as I haven't used emacs in years. And since I don't know what the stroustrup style exactly does, very likely the code written by me does not follow all the rules there, but that code was the only easy reference for me to use when judging the uncrustify result.

Sure, the point is to pick a style that is not onerous to configure editors to produce and which conveys meaning visually in a way we think is useful.

> I like the general idea of indentation reflecting a scope change (which makes it easy to find the end of a block). For case statements, that

> suggests indent_switch_case=0. I don't care too much, because they don't get nested much. (IIRC case statements do start their own scope blocks.)

The reason why I picked indent_switch_case=4 was that I think that a rule "everything within a brace gets indented" is the most straightforward, and the extra indent can only improve readability. Both styles (either indented or not indented case labels) were mixed throughout the code. Just based on a quick look, I couldn't judge which one was used more.

Sure, the status quo in any GROMACS code is not normative :-)

> (PS. Case statements do not, by themselves, start a new block. One has to add an extra set of braces if one wants to, e.g., declare variables. Also for this reason, I find indent_switch_case=4 more clear: the following is more readable than pulling the whole switch contents left one indent level (this example is what the current uncrustify configuration produces):)
> [...]
>
> > If we might get nested classes, then a whole indent level for access specifiers seems a bit heavy to me. Spamming the upper indent level with access specifiers that don't strictly change scope is not too palatable either. There are styles that use a partial indent for access specifiers (I'd suggest 2 spaces), so that there's a visual clue for changes of access, but that does not affect the visual clue for change of scope. That corresponds to indent_access_spec=-2 and indent_access_body=false.
>
> As above, this goes with the simple rule that everything within braces gets indented.

We-e-ell namespace braces currently don't get indented by uncrustify. And that's consistent with past GROMACS practice of avoiding indenting for extern "C" {} blocks.

> I don't see how a partial indent would improve things, as it will be harder to write without some custom configuration in an editor.

Unless we can pick a style that's supported with a "button press" in (at least) emacs, vi and Eclipse then there are people who have to configure something regardless. I'm happy if we just pick some useful, widespread and editor-supported style. **But we should do that now before people start developing C++ habits for GROMACS.**

> I haven't been bothered by the limitation of having 4 characters more indent within class declarations (and thus 4 characters less before hitting the 80 char limit). Partial indent also probably doesn't correspond to any editor's default style (for reference, the current one is more or less the default on vi, but is also my personal preference from times when I did not yet use vi). If nested classes would get indented too much, it is always possible to move their definition outside the main class to get it start back from indentation level zero.

Good point. Definitions of class and functions often should not be inline anyway. Now I don't care about access specifier indenting enough to bother about.

> Thoughts?
>
> The above said, I don't really have any hard opinions on these things. If people generally want some other style, then that's fine by me. But we should try to keep things as simple as possible such that it is possible to easily configure editors that people use to actually follow that style.

Yep.

#86 - 02/21/2013 06:11 AM - Teemu Murtola

Mark Abraham wrote:

> > As above, this goes with the simple rule that everything within braces gets indented.
>
> We-e-ell namespace braces currently don't get indented by uncrustify. And that's consistent with past GROMACS practice of avoiding indenting for extern "C" {} blocks.

True, these are the two exceptions to the rule. I think they are very reasonable, as otherwise more or less every single line would be indented by at least 4 or 8 characters extra, in particular for the namespace indent.

> > I don't see how a partial indent would improve things, as it will be harder to write without some custom configuration in an editor.
>
> Unless we can pick a style that's supported with a "button press" in (at least) emacs, vi and Eclipse then there are people who have to configure something regardless. I'm happy if we just pick some useful, widespread and editor-supported style. **But we should do that now before people start developing C++ habits for GROMACS.**

That's a bit late for me. ;) And I haven't seen many comments on the C++ style, which probably means that people will not have one until they actually start writing code. I can only speak for vi(m), and I checked that stuff suggested here is easy to configure either way (both the case labels and access specifiers). In general, it is probably easiest if we avoid partial indents (those may be more difficult to configure). Other than that, I see no other way than people using different editors actually complaining what they find difficult to write in the current or proposed styles (or what they would prefer to have changed for other reasons). If we want to use the emacs "stroustrup" style as the guideline, then it would be great if someone who is using emacs would actually spell out how it works for C++ code.

#87 - 02/21/2013 11:43 AM - Berk Hess

I found this page with some examples of emacs C++ indentation styles:
http://davidha.wordpress.com/2009/05/15/emacs-cc-modes-built-in-styles-gallery/

#88 - 02/21/2013 08:18 PM - Teemu Murtola

Berk Hess wrote:

> I found this page with some examples of emacs C++ indentation styles:
> http://davidha.wordpress.com/2009/05/15/emacs-cc-modes-built-in-styles-gallery/

Unfortunately, the example code is Java, so it doesn't help much (in particular, those access specifiers are not present). If the page is right, the stroustrup style seems to have some funny indentation for braces around class member functions bodies, but don't know whether it really is so for C++.

#89 - 03/17/2013 12:06 PM - David van der Spoel

Code formatting .emacs code for C++ is here: http://www.gromacs.org/Developer_Zone/Programming_Guide/Code_Formatting

#90 - 04/29/2013 07:06 PM - Mark Abraham

- Target version deleted (4.6)

#91 - 12/09/2013 06:36 AM - Roland Schulz

I uploaded a commit which uncrustifies all files since we ran it the last time. I think we should avoid doing this from time to time because it is annoying when using blame. On the other hand if we don't do it from time to time then running the script on files modified is annoying for code review because files outside the ones changed get changed by the uncrustify.

Thus I think we should do it on each commit with Jenkins. I looked at the discussion from a year ago and couldn't find any objections. Should we go ahead and do it now?

#92 - 12/09/2013 07:13 PM - Teemu Murtola

I support such an approach. I added the script and the pre-commit hook to support this, but not much has happened since. I suspect it got dropped since no one was interested enough in making it actually happen. Probably the only things remaining to consider for Jenkins is:

- Whether Jenkins should just vote, or also do the reformatting.
- If the latter, there should be a check that running uncrustify more than once does not produce additional changes. And a safeguard that this doesn't end up in an endless loop (at least with some uncrustify configuration settings, it can end up adding whitespace indefinitely).
- There should be a mechanism to suppress automatic reformatting (and copyright checks, if we want to add those to Jenkins as well). For example, a tag in the commit message that skips this step in Jenkins. Otherwise, it will be a pain to do stuff like experiment with uncrustify configuration and ask for comments, or to do bulk updates to copyright headers (which we hopefully don't need to do in the near future).

#93 - 12/09/2013 08:33 PM - Roland Schulz

- In the previous discussion some people were for reformatting and no one against. So I assume we do that.
- Good point about the loop. We should first auto-reformat and then rerun to check that it is OK and stable.
- If we don't want to change the copyright for trivial changes, then we shouldn't automatically fix the copyright but only check it (with the vote with can be overwritten manually)
- From those examples you gave I don't see why we would need the suppression of automatic reformatting. In the first case it wouldn't hurt the review to have a new commit which undoes the change. In the 2nd the automatic reformatting shouldn't do anything. Thus I would suggest we only add the suppression if the practice shows it is needed.

#94 - 12/10/2013 12:11 PM - Mark Abraham

I side-step some issues by using things like admin/uncrustify.sh --copyright=update --rev=HEAD^ update-workdir, but the glitches are still annoying.

Ideally, we would avoid situations where we upload, Jenkins-test triggers, Jenkins-uncrustify works out it should reformat, and does so, triggering Jenkins again. Sometimes the Jenkins queues stall until there's a complete flush, which is a PITA at high load times

One way to do that would be to have a pipeline where Jenkins-uncrustify acts first (and set up all the slaves with uncrustify so the job can float to a free slave). If the code changes stably, then we upload with a special Jenkins-did-uncrustify author and report failure (and a new Jenkins job will be triggered by the upload). If the code was already OK, then we move to the next stage of the pipeline. The Jenkins-uncrustify job is a no-op if the committer was the Jenkins-did-uncrustify author, or perhaps if it had some magic tag.

IIRC Szilard didn't like the idea of tags in commit messages passing on into history. That doesn't bother me. An alternative would be to tell people who need to bypass uncrustify to set GIT_AUTHOR_EMAIL/NAME to that of the Jenkins-did-uncrustify author (e.g. http://schacon.github.io/git/git-commit-tree.html) and now we only need one predicate leading to the no-op.

We should sync up deploying that along with the full-source uncrustify.

#95 - 12/10/2013 04:11 PM - Roland Schulz

Mark Abraham wrote:

> I side-step some issues by using things like admin/uncrustify.sh --copyright=update --rev=HEAD^ update-workdir, but the glitches are still annoying.

What glitches are you referring to?

> Ideally, we would avoid situations where we upload, Jenkins-test triggers, Jenkins-uncrustify works out it should reformat, and does so, triggering Jenkins again. Sometimes the Jenkins queues stall until there's a complete flush, which is a PITA at high load times

I haven't noticed it stalling. Is there some pattern of when this happens? And it seems to work pretty well that if a commit is updated before the build finishes that it cancels the build to not waste build time.

> One way to do that would be to have a pipeline where Jenkins-uncrustify acts first (and set up all the slaves with uncrustify so the job can float to a free slave). If the code changes stably, then we upload with a special Jenkins-did-uncrustify author and report failure (and a new Jenkins job will be triggered by the upload). If the code was already OK, then we move to the next stage of the pipeline. The Jenkins-uncrustify job is a no-op if the committer was the Jenkins-did-uncrustify author, or perhaps if it had some magic tag.

The standard way to do pipelines in Jenkins is defining downstream jobs. The problem is that this doesn't work correctly with Jenkins Trigger: https://issues.jenkins-ci.org/browse/JENKINS-11409. One could do the uncrustify as pre-build step but that is a problem because we have several jobs. There is the https://wiki.jenkins-ci.org/display/JENKINS/Multijob+Plugin plugin. But I haven't tested it to see whether that would work for us.

> We should sync up deploying that along with the full-source uncrustify.

yes

#96 - 12/11/2013 12:13 AM - Mark Abraham

Roland Schulz wrote:

> Mark Abraham wrote:
>
> > I side-step some issues by using things like admin/uncrustify.sh --copyright=update --rev=HEAD^ update-workdir, but the glitches are still annoying.
>
> What glitches are you referring to?

Sorry, I just meant that crusties from past patches touching the same file as my commit would get fixed by the above uncrustify, and neither fixing them in my commit or avoiding them manually are desirable.

> > Ideally, we would avoid situations where we upload, Jenkins-test triggers, Jenkins-uncrustify works out it should reformat, and does so, triggering Jenkins again. Sometimes the Jenkins queues stall until there's a complete flush, which is a PITA at high load times
>
> I haven't noticed it stalling. Is there some pattern of when this happens? And it seems to work pretty well that if a commit is updated before the build finishes that it cancels the build to not waste build time.

Sorry, I was again too fast. A cancelled Jenkins job sometimes delays future jobs being considered for dispatch to slaves (e.g. a grey status circle in the queue) until some magic condition is satisfied, particularly under high load. I saw this most recently ~9 days ago, during the lead up to 5.0-beta1.

> > One way to do that would be to have a pipeline where Jenkins-uncrustify acts first (and set up all the slaves with uncrustify so the job can float to a free slave). If the code changes stably, then we upload with a special Jenkins-did-uncrustify author and report failure (and a new Jenkins job will be triggered by the upload). If the code was already OK, then we move to the next stage of the pipeline. The Jenkins-uncrustify job is a no-op if the committer was the Jenkins-did-uncrustify author, or perhaps if it had some magic tag.
>
> The standard way to do pipelines in Jenkins is defining downstream jobs. The problem is that this doesn't work correctly with Jenkins Trigger: https://issues.jenkins-ci.org/browse/JENKINS-11409. One could do the uncrustify as pre-build step but that is a problem because we have several jobs. There is the https://wiki.jenkins-ci.org/display/JENKINS/Multijob+Plugin plugin. But I haven't tested it to see whether that would work for us.

There seem to be many attempts to build pipelines in Jenkins, e.g. https://wiki.jenkins-ci.org/display/JENKINS/Build+Pipeline+Plugin and http://antagonisticpleiotropy.blogspot.com.au/2012/02/implementing-real-build-pipeline-with.html seem plausible. The former looks good to me. Multijob also seems possible.

> We should sync up deploying that along with the full-source uncrustify.

yes

#97 - 12/11/2013 06:17 AM - Teemu Murtola

I'm fine if we don't want to implement a mechanism to override the formatting; those cases should be exceedingly rare. I'm fine to apply the copyright years also for trivial changes. The only place where I would not like to see them is trivial, non-content changes that affect every file, such as rewording something in the copyright header or running uncrustify with completely different settings.

I would also be fine with Jenkins just voting if the formatting/copyright does not match. Based on the discussion and limitations in the plugins we need to use, it seems that would be a lot easier to setup.

Overriding the committer may work, assuming that Gerrit allows you to push stuff where your ssh key/registered e-mail does not match the identity in the commit header (also for non-admins; there could be a permission required for this).

#98 - 12/11/2013 07:15 AM - Roland Schulz

I clearly prefer having Jenkins just vote now, over having Jenkins also fix - in a year or so.

If I understand the build-pipleline plugin correctly it main aim is to visualize the pipeline. It doesn't change the dependencies. So both build-pipleline plugin and the solution described by the blog post, use the "Parameterized Trigger Plugin" for the dependencies (with optional other plugins improving the views / conditions / ...). One problem with the parameterized plugin is that we have to use the "Block until the triggered projects finish their builds" option (otherwise Gerrit doesn't get the aggregated result), and in that case the triggering job is using an executor while waiting (e.g. https://issues.jenkins-ci.org/browse/JENKINS-12290). One can increase the number of executors per VM (they don't need to be a fixed ratio to the cores), but that might make it more difficult to avoid oversubscribing VMs. I'm not sure whether the multi-job plugin has the same issue.

Given that this is non trivially and no one probably has time to look into this in detail anytime soon; I think we should just go with the vote-only option. And as long as developers once install the post-commit hook, git does usually everything automatically so it wouldn't be any extra work for the developer.

Having said that. It would probably still be good if we had some kind of build pipeline in the long run - not just for the uncrustify (e.g. running several (parameterized - e.g. number of threads) tests on the same build).

#99 - 12/11/2013 07:40 PM - Teemu Murtola

In addition to whole-source uncrustification, we should also merge https://gerrit.gromacs.org/#/c/2847/ and its dependencies before we take this into use. In particular if we want to include the copyright headers in the check, but also if we don't, since the last commit changes the uncrustified files (to include CUDA files, which were previously excluded, also from the mass uncrustification).

#100 - 01/25/2014 03:03 AM - Roland Schulz

- File uncrustify added

Attached binary should work on most 64bit Linux distributions. For those who don't want to compile from source.

#101 - 01/28/2014 06:25 PM - Roland Schulz

I added a Jenkins job but didn't mean for it to actual vote. Not quite sure why it votes even though it is disabled. Let me know if you don't like that it is voting and I check again.

#102 - 01/28/2014 10:42 PM - Mark Abraham

It doesn't downvote if there's just crusties. But it inhibits the upvote that would otherwise occur. So in lieu of some kind of auto fix that seems ok

#103 - 01/29/2014 09:10 PM - Roland Schulz

- Status changed from In Progress to Closed

I think we can finally close this. Please reopen if you see any further issues.

#104 - 07/15/2014 01:46 PM - Teemu Murtola

- Category set to build system

- Target version set to 5.0

#105 - 10/30/2014 12:02 AM - Roland Schulz

Now that the canceling of jobs works fine (since we installed gerrit-trigger 2.12.0-beta-5). Should we change the Jenkins uncrustify job that it updates commits automatically not just vote? It might help new developers because they don't need to install uncrustify (e.g. https://gerrit.gromacs.org/#/c/4170/)

#106 - 10/30/2014 12:06 AM - Roland Schulz

- Description updated

Files

gromacs-uncrustify.cfg	62.7 KB	11/11/2012	Mark Abraham
uncrustify	511 KB	01/25/2014	Roland Schulz