Gromacs - Feature #969

Generating man pages, html help etc. from Options

07/12/2012 12:32 PM - Teemu Murtola

Status:	Closed
Priority:	Normal
Assignee:	Teemu Murtola
Category:	core library
Target version:	5.0

Description

parse_common_args() implements a hidden command-line option -man, that can be used to produce the list of options and the description of the program in a variety of formats, including e.g. man pages, html help and shell completion specifications. This is not yet implemented for command-line parsing using gmx::CommandLineParser and friends, which is used by the new trajectory analysis framework. This is the only remaining big feature that is missing; with it, most of the command-line option handling could be easily rewritten to use the new system.

The old system, mainly implemented in wman.c(pp), uses Gromacs-specific markup syntax in all longer text blocks, and has special code to write out the option descriptions in a variety of formats. While it is possible to replicate this exactly, it would be worthwhile to consider the alternatives at this point to avoid unnecessary work and a lot of code to maintain. As most of the markup is in tools, and #665 requires heavy modifications to them, it isn't much more work to modify the markup at the same time.

There are two different choices to make. The first is between these:

1. Keep the current approach and have separate code paths for all different output formats.
2. Only support console output for g_tool -h, g_ana help etc., and one external format. Use an external tool to convert the external format to each required output format. Need to investigate which external format is most suitable, and whether the same input can be used to produce all the required output formats reasonably. txt2tags has been suggested in #690, there are probably several other alternatives as well. David suggested xml-based format in some e-mail chain.

For either case, we should consider which of the current output formats are really necessary.

A second, somewhat orthogonal choice is for the markup syntax used:

1. Keep the current markup syntax.
2. Use a different markup syntax, preferably a well-known one. If combined with the second choice above, the markup syntax would preferably be as close as possible to the external format to make it possible to pass it to the external output without much modifications in code.

Both of these alternatives require us to implement a parser for the markup syntax (to provide the console output), unless we either select a markup syntax that can be printed unmodified to console, or we accept that the console output is read by the tools from a text file. The last alternative would make it possible to convert the text with markup (that could still be part of the source files) into plain text at development time using an external utility, and the release binaries could read the text files (e.g., from somewhere under share/gromacs/) instead of requiring the external utility to be present (e.g., txt2tags requires Python).

Please add if I've forgotten anything.

I can implement one of the alternatives, but for making the decision I would really need some discussion on the possible external formats and/or markup alternatives.

Related issues:

Related to Gromacs - Bug #690: options program needs to be built by default	**Closed**	**01/28/2011**
Related to Gromacs - Task #666: Improve help printing in the command-line runner	**Closed**	**01/14/2011**
Related to Gromacs - Feature #1410: Future of shell completions	**Closed**	**12/27/2013**
Related to Gromacs - Feature #1437: Online help formatting improvements	**New**	**02/15/2014**
Blocks Gromacs - Task #679: g_select documentation should be included in the ...	**Closed**	**01/22/2011**

Associated revisions

Revision 1eb59d9a - 07/15/2012 03:48 PM - Teemu Murtola

Draft support for multiple help output formats.

- Added a HelpWriterContext class. An instance is now passed to all methods that write help text. In addition to keeping track of the output format, it also allows other context information to be kept (e.g., for formatting reasonable cross-references; earlier I had to remove those that were in the selection help because static text was not appropriate for all situations).
- Moved current output format specific methods to be members of this class.
- Adjusted using methods to work as before for console output, but did not yet implement any additional output formats. All methods with console-specific code now throw a NotImplementedError to make it easy to find them once #969 is decided.

Once there has been some discussion on the markup and output formats in #969, this approach can be extended to actually implement it.
No matter which of the alternatives is chosen, the parts in this commit are useful, but the decisions made in #969 affect the direction to which the implementation should be extended.

Prerequisite for #969.

Change-Id: I33cd59e6f3b5450db99e0e1afba4a2d1b9e30e29

Revision 5145db01 - 08/31/2012 05:07 PM - Teemu Murtola

Reorganize CommandLineHelpWriter implementation.

Instead of separate classes to write out descriptions and different types of options, there is now three main local classes:
- OptionsFilter: has output format independent logic to do the filterting that was previously implemented in each class separately.
- OptionsFormatterInterface: interface that OptionsFilter uses to do output format specific formatting for the options.
- OptionsConsoleFormatter: implements the above interface to produce the console help that was previously done by the separate classes.

Prerequisite for #969 independent of the chosen solution.
May need additional work as part of that issue.

Change-Id: Ica506f3567b4527f210c48c1b43069d4b64007ec

Revision 3ffcfd69 - 09/07/2012 06:11 AM - Teemu Murtola

More features for TextLineWrapper.

Add more control over formatting the wrapped lines:
- Add possibility to indent the wrapped lines.
- Add possibility to indent the first line after an explicit line break differently from other lines.
- Add possibility to mark line continuation using an explicit character.
- Add possibility to either remove or keep leading whitespace after an explicit line break.
All except the last are already useful for the current console output. Depending on the chosen solution, some or all of these are also needed for #969.

Also expose the lower-level interface used internally that makes it possible to write code that iterates over the wrapped lines and does something else than just format them into a buffer. May be useful for some cases in #969, but is necessary to use the wrapper for #838 (line wrapping in error messages).

Change-Id: I905ba29856773656bf000c4b2e14d1ed2ba4de31

Revision cd965ad6 - 03/07/2013 07:57 PM - Teemu Murtola

More flexible handling of enum option descriptions.

Instead of always appending the list of allowed values to the description of enum options, expose the list through StringOptionInfo and construct the final description in cmdlinehelpwriter.cpp.

This gives better control for printing the option lists in different formats. Also allows removal of AbstractOption::createDescription(), streamlining the core option implementation slightly.

Prerequisite for #969.

Change-Id: I26494f79757ad6894f1930b1bff2f2c74cc26f9c

Revision 1ee29850 - 06/13/2013 09:45 AM - Teemu Murtola

More flexible handling of enum option descriptions.

Instead of always appending the list of allowed values to the description of enum options, expose the list through StringOptionInfo and construct the final description in cmdlinehelpwriter.cpp.

This gives better control for printing the option lists in different formats. Also allows removal of AbstractOption::createDescription(), streamlining the core option implementation slightly.

Prerequisite for #969.

Change-Id: I26494f79757ad6894f1930b1bff2f2c74cc26f9c

Revision 12c3bfd3 - 09/13/2013 09:10 PM - Teemu Murtola

Remove unused -man output formats.

Currently, only HTML, LaTeX, man pages, and console output are actually used.

Related to #969.

Change-Id: I32e5cec872527a779fad9fe742135bcfb30a13a7

Revision f107cc6f - 09/19/2013 05:16 AM - Teemu Murtola

Framework for help export from wrapper binary.

- Add a separate CommandLineHelpContext. This class layers extra information on top of a HelpWriterContext, specific for command-line help export.
- Add a global instance of the above to be able to pass it into write_man() through functions unaware of its existence. Make write_man() use the instance if present.
- Add -export option for 'gmx help', and an interface that needs to be implemented to export a particular format.

Related to #685 and #969.

Change-Id: Ica16895f8136a09bc5995812c4da5363d097c2b1

Revision 1c73f59f - 09/19/2013 05:28 AM - Teemu Murtola

Uniform code path for writing out console help.

Now both 'gmx help <module>' and 'gmx <module> -h' trigger exactly the
same code path, making things a lot easier to work with.
Some notable things:
- Moved the responsibility to parse the -hidden argument into the
wrapper binary, from where it gets passed down in
CommandLineHelpContext. Quite a few files are touched by this.
- The -h option now causes all other options to the module get ignored.
g_tune_pme requires some other approach to deal with option
validation, but even adding a separate command-line option for only
this purpose is probably better than the multiple code paths that
were there before this change.
- Related to the above, the help output could be significantly
simplified, since it no longer depends on the command-line.
- Removed the -verbose option that caused the options to be printed
during a normal run. Possible to add back if people want it, but it
simplifies things if it isn't needed.

Related to #685 and #969.

Change-Id: Ibe735711f650eafaecf28ffb1ed92da97dcb81b6

Revision 031d6d80 - 10/03/2013 05:25 PM - Teemu Murtola

Man page export from the wrapper binary.

Implement nroff output format for 'gmx help -export'.
Move the header/footer generation out of wman.cpp.
The generation is now off by default. Can be reinstantiated if someone
comes with a good approach for the case where the compiled binaries
don't run on the build host (e.g., compiling AVX256 binaries on a SSE4.1
host).

Related to #685 and #969.

Change-Id: I17725e3a487470bea8f6db2a59da31139e70621e

Revision abc93fe9 - 10/07/2013 04:31 PM - Teemu Murtola

HTML export from the wrapper binary.

Implement HTML output format for 'gmx help -export'.
Fix an issue that caused 'mdrun -man html' to segfault.
Move the HTML header and footer generation out of wman.cpp and clean it
up (close incomplete tags, remove the date, etc.).

Clean up the layout of the online manual (on the file system) by putting the generated files into a separate directory. Also generate an alphabetical list of programs into its own file instead of directly on the front page. By-topic list is currently missing.

Left out installation rules for now; can be reinstantiated once #1242 and the general approach to the whole online manual is clearer.

Will fix hyperlinks etc. in the generated output in separate change(s).

Related to #685, #969, and #1242.

Change-Id: Iaf8fc28d563f05a8e00c7c52d58b91cd1dabf369

Revision caeb87ea - 10/07/2013 05:07 PM - Teemu Murtola

Improve markup substitution in HelpWriterContext.

- substituteMarkup() replaced with an interface that also does line wrapping at the same time. Allows more complex line wrapping that can also depend on the input markup, and clarifies responsibilities in the code.
- Make TextTableFormatter use the new interface, removing explicit substituteMarkup() calls from elsewhere in the code.
- Structure the substitution to allow easy addition of more output formats.

Related to #685 and #969.

Change-Id: Id34be9489aa3a90d94cd87f8936b030bc21f1c98

Revision 0b9fcab2 - 10/16/2013 08:40 PM - Teemu Murtola

Remove latex help export.

The decision in I7e56a011 was to drop Appendix D from the manual. This commit removes the latex help export code that supported that. This makes it easier to manage change in wman.cpp, since now all the markup substitution code operates in the same environment.

Related to #685 and #969.

Change-Id: Iaeb5b82e288e3120ebe6ae38c0e5c3aca892194a

Revision 5e4af5d1 - 10/30/2013 04:45 PM - Teemu Murtola

Markup substitution through HelpWriterContext.

Make all calls to process the in-source strings to help output go
through HelpWriterContext. This makes it easy to implement common
functionality using C++ mechanisms.

Moved all the HTML link processing stuff into HelpWriterContext as well
and cleaned up links.dat. The generated HTML pages should no longer
contain broken links (but links between program pages are not there
yet).

The markup substitution itself is still done by functions in wman.cpp;
will move those into private functions in helpwritercontext.cpp as a
separate change.

Related to #685 and #969.

Change-Id: I37d45549214e49a6edab82fb64ec7b38b9e72094

Revision 88d213b1 - 11/16/2013 11:31 PM - Teemu Murtola

Add listing of programs by topic to HTML output.

Now the HTML-exported help contains also a list of programs by topic,
similar to what used to be generated from programs.txt. Removed the
mkhtml script, since it is now fully replaced by the mechanism in the
wrapper binary.

The same mechanism could also be used to replace the gromacs.7 man page,
but in its current form, it contains so much boilerplate code that I
didn't want to copy-paste that all into the source file. And it could
also be used to make the output of 'gmx help' more structured.

Related to #685, #969, and #1242.

Change-Id: I6c2efe10c53f10f7fde90b3386ddea7fbea34b89

Revision d84127aa - 01/20/2014 06:45 PM - Teemu Murtola

Rewrite help output from Options

(Partially) rewrite the way command-line help is printed for an Options
object. Now man pages and HTML output are supported in addition to
console output. The console output is formatted in a simpler manner,
since there is no longer any need to cater for user-provided values in
the output. There is some extra machinery now in cmdlinehelpwriter.cpp,
but left it there as it may still be useful in the future, and it does
structure the code a bit more cleanly.

As an extra bonus, the commandline module should no longer depend on the
selection module.

Things more or less work, but the following could still be addressed,
either in this change or later:
- Re-add selection option help formatting tests somewhere else.
- Add unit tests for various things about non-console help formatting.
- Refactor the way the help information is fetched from OptionInfo
objects: it would be cleaner to have a single method that returns all
the information in a struct, instead of multiple virtual methods that
return different pieces of information, many of which are not
required for anything else except the help output.
- Add proper synopsis that lists all the options.

Part of #969.

Change-Id: Iadf1b5df15f278547d5db6407f3e3c689fb88645

Revision 5bb0f5d8 - 01/21/2014 04:42 AM - Teemu Murtola

Improve file name option help

- Add brackets for cases where the file name is actually optional.
- Add [...] for cases where multiple file names can be specified.

As supporting changes, extend the options machinery to be able to get
this information for the options, and fix an issue in the table
formatter that was causing some ugly formatting.

Related to #969.

Change-Id: I5b38d7ac6d7c58c3decce64ddfe60fff0a5797ed

Revision d6f5bc31 - 02/09/2014 01:37 PM - Teemu Murtola

Refactor new help formatting implementation

- Merge formatFileOption() and formatOption() into one.
- Remove unnecessary generality by splitting description formatting into its own class, and make OptionsFilter only do option filtering. All output handling is now in OptionsListFormatter (which was renamed from OptionsExportFormatter).
- Split formatting basic option name and value placeholders into a separate function.

There should be no functional change. Preparation for a proper synopsis implementation.

Part of #969.

Change-Id: I900883eb39853ed67d5f6177ae06847283348d9e

Revision bb257109 - 02/09/2014 01:38 PM - Teemu Murtola

Implement synopsis for help output

Print all the options into the synopsis, for all the output formats (console, man pages, and HTML). Formatting is not the nicest possible, but surely better than what it was (in wman.cpp, the synopsis is only printed into the man pages, with no wrapping whatsoever).

Part of #969.

Change-Id: Ic09ac91b3d5b5e2d42d41b83935d54c894eb8e97

Revision 7d9eb7f6 - 02/09/2014 01:40 PM - Teemu Murtola

Remove old help formatting code

Make parse_common_args() format its help using the new CommandLineHelpWriter class. To support this:
- Add conversion functions that map t_filenm and t_pargs options to corresponding Option classes.
- Add support for arbitrary file types from the filenm.h enum to FileNameOption.
- Add support for writing a list of known issues from CommandLineHelpWriter.
- Some minor extensions to the Option classes to make it easy to do the conversion.
- Fix gmx_hbond to not specify the same option twice.
- Remove wman.* as unused after the change.

Related to #969.

Change-Id: I25143ddfa4226abd5ae5b95172f69c8fc615b720

Revision 12781054 - 02/09/2014 06:32 PM - Teemu Murtola

Shell completions through Options

- Implement shell completion generation for command line options
specified through an Options object.
- Use this support to generate the list of options for the wrapper
binary instead of hardcoding them.
- Use this support and the conversion from t_pargs/t_filenm to Options
to generate the existing completions. The only differences in the
generated completions are in the order of the options and in changing
"$n == 1" to "$n <= 1" and ".(xtc|trr|...)" to "(.xtc|.trr|...)".
- Extend some of the options to expose information necessary for this.

Related to #969 and #1410.

Change-Id: Ib77543367c38803ef186f6024a1af14feb806d80

History

#1 - 07/12/2012 05:12 PM - Teemu Murtola

My personal preference would be some variant of the third point. My understanding is that there is agreement on libxml2 being a required component for 5.0, so an xml-based markup syntax would be easy to parse (and is easy to make as flexible as needed), and would avoid the need for complex text manipulation code. As most of the text with markup is in tools, and these need to be heavily modified in any case for #665, it isn't much more work to modify the markup at the same time.

I don't now have time to delve into different possibilities for the markup syntax (e.g., docbook, there must be others as well) and whether it would be possible to generate directly a single xml document in one of these formats and then use external tools to generate all the wanted formats.

If not, a flexible approach that still wouldn't require much code is to

- use our own XML markup syntax (preferably one as close as possible to a known markup syntax),
- have a libxml2-based code that converts the XML markup into console output,
- have another libxml2-based code that writes the list of options etc. into an XML file, just writing any text with markup syntax directly into the output,
- have XSLT files to convert the XML file into the required output formats, and
- use libxslt to transform, either directly from the binaries or by cmake script that calls xsltproc.

Note that the dependencies on libxslt are only development-time dependencies: release binaries do not need this functionality, as the generated files can be distributed in the source packages.

#2 - 07/16/2012 05:48 AM - Teemu Murtola

- Description updated

#3 - 07/26/2012 08:58 AM - Roland Schulz

My preference would be that the source format isn't XML. XML makes it unnecessary difficult to be written by humans.

Either we keep the current format. Why can we not just keep using wman.cpp with the new tools? Using a new will be better (probably even if using XML). But I'm not sure the advantages justify implementing something new.

If we choose a new format I would suggest we use something like txt2tags, asciidoc, or Markdown. All can be converted to XML/XHTML if XSTL processing is wanted. And both are easy enough to read that they can be printed to console if at development time the tool is missing (as you mention the source archive can contain the processed files).

I think one requirement (and in very basic way fulfilled by the current format) is that basic equations are supported. I think most of the formats have no native support for it but most support embedding of Latex/Mathml. In our community most probably know Latex and also Latex is easier to read, and for simple equation the latex source format is even OK for direct text output (console/man). For HTML output we could use MathJax to have embedded equations in either format rendered nicely. A good tool to convert Markdown (and also a couple of other formats) into all required output format is Pandoc .

#4 - 07/27/2012 03:45 AM - Mark Abraham

Points I think are important:

1. documentation in the code file is good
2. being able to produce (at least) man, -h, LaTeX and HTML formats is good (but I can live with a subset)
3. making it easy for humans to write documentation is good
4. having custom markup languages and writing parsers are bad
5. making it easy for developers to see how the documentation turns out is good
6. being able to embed (simple) LaTeX and have it turn out OK in plain text is good
7. not having a huge documenatation conversion burden during the transition to 5.0

So something like Markdown with Pandoc looks OK at first sight to me, but the Haskell dependency of Pandoc takes the shine off because of point 5 above. Perhaps multimarkdown http://fletcher.github.com/peg-multimarkdown/ is better in this regard, but even it has some GTK+ dependencies.

Generating the output of g_tool -h at run time meant you could fit things to the terminal, but that's really optional.

We need to cater for three kinds of builds:

1. user with a tarball (should have pre-processed docs probably built with CPack as part of the tarball process so that there are no dependencies; either CMake can splice the pre-processed docs into the code for -h output, or they can be read from an installed location as required)
2. developer in a git repo doing day to day work (should be able to choose to use pre-processed docs in the source tree with some kind of acknowledgement if CMake can tell these are out of date, or build from the real documentation source)
3. developer doing a release (having to update pre-processed docs in source tree and/or tarball, and the small number of relevant users means I'd be willing to tolerate almost any dependencies here if it makes other stuff easy)

1 and 3 are easy to implement, but 2 is harder. It's not acceptable to frustrate developers into not documenting their code because doing so requires them installing some new software to get their code to compile with the (new) documentation.

A particularly awkward aspect is building the other g_tool options into the documentation. Some kind of trickery is required for an enumerated option to be written only once and yet generate both code and documentation (for example). This seems like a deal breaker. Unless something that can already parse C/C++ (like doxygen?) can be made to produce the documentation, we're stuck with writing some kind of parser ourselves, and I think we should keep the existing machinery rather than change much of anything in that case.

#5 - 07/27/2012 09:09 PM - Teemu Murtola

I don't have time right now to comment on all the issues raised, but here are some comments.

In my mind, the ideal situation would be something like this:

1. The code has blocks of documentation text (e.g., tool descriptions) as strings with some markup A.
2. The code specifies the options it understands and their descriptions. The descriptions may contain markup A.
3. When, e.g., "g_ana tool -h" is run,
    1. the code parses markup A in the text blocks (or just passes it through if that is feasible) and produces the console output, with minimal amount of code and minimal external dependencies, and
    2. the code prints out description of the options to the console.
4. It's possible to run "g_ana help -writeext" or similar, and it will produce one output file B. Markup A is passed unmodified to the output. This file will again contain
    1. the blocks of documentation text and
    2. list of options with their descriptions and any other properties specified in the code.
5. CMake has rules to produce all the necessary external documentation from file B. The less code we need to write ourselves for this, the better.
6. No generated source files are stored in git, but a necessary set may be distributed in source tarballs.

And the essential part of this issue is that current code in wman.cpp does not do any of 4.2. for the new tools, because they use different data structures to specify their options. Of course I can either:

1. Write code to convert the new options to those understood by wman.cpp, but there are things allowed by the new option handling that are impossible with the old, and these will then need some special treatment.
2. Duplicate all the code from wman.cpp to write out the option part in all the possible output formats for the new option structures.

But before doing one of these, I wanted to have a bit more general discussion to avoid a lot of unnecessary work.

#6 - 08/05/2012 07:04 AM - Teemu Murtola

Here are some random thoughts on the markup format. I don't actually have any strong feeling for any of the alternatives for markup, but I still think that we should consider them.

About the old markup format:

- There is currently no validation, and it's not particularly easy to implement: if someone adds a mismatching tag into a documentation string, one of the documentation formats may break silently (so that only, e.g., the manual build fails). The console output typically works fine in these cases.
- It requires concatenating the string array and replacing the markup in one go. It would be simpler if these could be done independent of each other, but I can also add extra (not-so-simple) C++ code to keep the string array information intact until the markup substitution is done if we want to keep the old format.
- I think there are some places that try to produce a list using something like

  ```
  "[BR]",
  "  * Item 1[BR]",
  "  * Item 2[BR]",
  ```

  which doesn't really look nice in any output format (except the console, if the item descriptions don't span more than one line).
- There is no support for something like [PRE], and that would not be very easy to implement either. This could be very useful, e.g., for the

selection online help (which currently also uses the [BR] trick, leading to unintentional whitespace stripping in the C++ alternative).

About XML:

- <tt></tt> isn't much harded to type than [TT][tt], and it's more clear which is the start and which is the end tag. For more complex cases (like the [SUM] construct), an XML-based format could also be clearer.
- I agree that the need to use entity references (e.g., < for <) makes things a bit harder, but those should not be that common. And majority of such problems would occur in equations, so we could possibly do something like insert CDATA tags within a $$ tag automatically before passing the string to an XML parser.
- I took a brief look at docbook, and it confirmed what I had read from somewhere: it's quite verbose, and probably not very good for our purpose. Something like XHTML would probably be ok.
- If we go with an XML-based format, it may be simplest to add a few of our own tags. We could, for example, add a <math> tag, inside which one could type LaTeX formulas and those would be processed by external tools for, e.g., HTML display. With XML, it's also easy to add an alternative textual representation for more complex cases or specify some other processing instruction.
- MathML is *very* inconvenient to write by hand, so I think that can't be used as the in-source format.
- XSLT processing would probably be most useful for the option output (produced by 4.2), and not so much for the markup itself, but could be simpler to use the same for both.

About other things:

- If we want to print out the in-source strings without any processing, that means that they should use consistent line wrapping to a predetermined line width to look reasonable. This is not very convenient for editing them.
- I don't think it is a good idea to have cmake magic that inserts stuff into the source code. Using an external text file should be clearer.
- An essential feature for getting some benefit from this exercise would be that the same output *file*, not just the same *markup*, could be used to produce all kinds of output documents (perhaps with the exception of the console output). Currently, as an example, I think that the HTML output uses tables for options, while man pages use a different layout.
- Very exotic dependencies on the external markup processor may exclude it. At least it will require discussion (again, probably post-4.6), since external dependencies seem to create that. It will be easier if it is possible to do normal work (including basic editing of the help texts and checking that they work) without that external dependency.

#7 - 08/06/2012 12:54 AM - Roland Schulz

Regarding dependencies:

1. If we take any external parser we will have a dependency for developers. And as long as binaries exists and are available for all major Linux distributions, so that installing is simply one command, I don't think it matters whether it is xsltproc (also not installed by default on Linux) or Haskell+Pandoc. Besides for developers it isn't an issue, because for users we'll have the preprocessed files and the release process is done on Jenkins.
2. I don't think it should be frustrating for developers if installing is as simple as one command. But even if developers don't have the dependency, as long as the documentation contains little formatting (we should aim for that anyhow because otherwise it will be difficult to print to console), it is not really required to have it processed to see how the documentation will look like. Also one will be able to see the output in Jenkins if one doesn't want to install the dependency.

Teemu, you have good arguments why XML would be a good option and not as verbose as I claimed. I agree that using custom tags would help. But how would XML be better than Markdown or any of the other ligthweight markup languages? Without XSLT it seems it would be a lot of code and with xslt it also has an extra dependencies. Also creating our own XML schema would require more work then simply going with an existing format. And I'm not sure we need the added flexibility.

Your right about line wrapping. We could make it nicer by automatic line wrap the text before printing. Should be very easy to do. Or we could let pandoc do it if availabe and otherwise print as is.

#8 - 08/06/2012 10:06 PM - Teemu Murtola

Some more comments in no particular order nor in particular relation to each other (sorry, this is a large topic, and it would take a long time to try to organize all my thoughts, so I think it's better to first gather stuff here):

- From the wiki (http://www.gromacs.org/Developer_Zone/Programming_Guide/Compilation), libxml2 seems to be a well-accepted external dependency (that text has been there for ages). I think that the reasons quoted there also apply to libxslt (since it is essentially only an extension on top of libxml2), but for other cases it may be more involved. But I agree that since it is only for developers, it's not that big of an issue.
- To implement line wrapping for any in-source format, we need to be able to parse at least some constructs in the markup in order to not mess things up. Essentially we would need to recognize any formatting command that produces extra linefeeds (e.g., paragraph breaks, tables) and/or indentation (e.g., bullet lists). One option would be to restrict the use of such constructs (they are not possible in the current markup either).

- With an XML-based format, it should be very easy to implement a pass-through "parser" that just strips away most tags and does line wrapping. Using libxml2, there is no string processing for the input, just walking through the XML tree. Even this simple implementation would validate the input such that it doesn't contain mismatching or unknown tags. The likelihood of the XSLT transformations breaking because of something slipping past is quite small. So the xslt dependency would only be needed for actually generating the other formats or checking something specifically in them.
- Such an implementation should be straightforward to extend to format links to other help pages (wman.cpp uses very rudimentary text replacement), and for some basic cases that need special treatment on console output (e.g., bullet lists). But this again adds more C++ code.
- Validation of an external markup syntax would require that external processor, but as long as it's installed on Jenkins, that should be fine.

- With an XML-based format, the option list (4.2 in my list) could be simply written out from the code as a list of tags for all the properties there are, and the XSLT transformations could produce any format they want from this.
- With an external format, we need to have C++ code for producing the already formatted list of options, and this needs to be the same for all output formats (probably such that the generated man pages have a standard layout). Producing markup from code is relatively straightforward, but it's still more C++ code to maintain.

- Mixing XML (for 4.2) and an external format (for 4.1) probably creates more complexity than it solves.
- I agree the it would be best if we wouldn't need to write the XSLT transformations ourselves.

#9 - 08/07/2012 12:20 AM - Roland Schulz

- Yes libxml2 is an accepted dependency but I think libxslt is a separate entity which is not necessary available.
- Probably we should just use the parser itself to do the line-wrapping. That would just mean that for developers who don't have the dependency they don't get auto line-wrapping.
- Markdown and XHMTL describe abstract layout (e.g. table, paragraph, italic, bullet list) whereas XML can describe content (e.g. for the option list: program name, default option, ...). Abstract layout only allows styling (font type/size, indention, ...) for different output formats. XML allows the usage of template/xslt and restructure the text (transpose or resort table, change order of paragraphs). If we think we might need that for something (e.g. the option list) then this would be an important advantage. I think that the option list can be a table with the same ordering in each output format. Also I don't see a need for reordering/restructering for 4.1. Thus I'm not sure we need XML.
- I agree that mixing XML and some lightweight markup would be a bad idea. But I think as long as we don't need text restructuring we don't need XML.

#10 - 08/12/2012 12:38 PM - Teemu Murtola

I think there are basically four alternatives here. Trying to summarize what they would need:

1. Keep using the old markup and write out each format explicitly in code (like in wman.cpp).
   - Can reuse the markup substitution part.
   - Need to live with the limitations (some listed in earlier comments); relatively difficult to implement new features (unless it falls under the simple search/replace model currently used).
   - Need to write code to format an Options object to each output format separately.
   - No need to change the existing code that formats documentation for t_pargs etc. (optional in the other cases as well, but it's harder to maintain two implementations that work very differently).
2. Keep using the old markup, but use only one external format.
   - Can reuse some of the markup substitution code, but need to write new code to produce the external format from the current markup.
   - Limitations with the markup still there as with above.
   - Need to define a common layout such that all output formats can be produced from it.
   - Need to write code to format an Options object to the external format.
   - Need to implement a build system that runs an external tool to convert the external format to desired output formats. The dependency is needed only for generating the documentation; it has no effect on other functionality.
3. Use an XML-based markup format and an external XML-based format.
   - Need to write new markup substitution code (for console only) using libxml2, but can focus on content, and not on parsing the input string.
   - Much easier to improve (e.g., crosslinks in the documentation, <pre>, LaTeX math with optional plain text replacement, bullet lists, ...).
   - Need to write code to format an Options object to the XML output format. C++ code does not need to deal with formatting a human-readable description of option flags.
   - Need to implement a build system that runs an external tool to convert the XML format to desired output formats. Probably also need to develop XSLTs for different output formats. The dependency (libxslt) is needed only for generating the documentation; it has no effect on other functionality.
   - Can format different output formats with different layouts (but probably no need for this).
4. Use a lightweight markup format, both in-source and as an external format.
   - Can/must write the markup directly to console output.
   - Need to define a common layout such that all output formats can be produced from it.
   - Need to write code to format an Options object to the external output format.
   - Need to implement a build system that runs an external tool to convert the external format to desired output formats.
   - Need to implement either
     - Code to run the external tool on the fly to produce the console help, or
     - Build system to run the external tool to generate console help texts and store them somewhere, and code to read and display them.
   - The external tool is needed for validating the in-source formatting, as well as for producing the console output.
   - Need to implement fallback output for the case that the external tool is not available.

Finally, some separate comments:

- Yes, libxslt is separate from libxml2, but I would assume that they have very similar build systems and dependencies. So if libxml2 is easy to build and thus acceptable as a dependency, libxslt could be as well.
- From a quick look, it seems that at least pandoc does not have a "plain text" output option, so this would mean that we must print the lightweight markup syntax also to the console output. May not be a big problem, at least if we keep its use relatively limited.
- (may have said this already) The layout for man pages is relatively standard, so if we go with only one layout for all the output formats, we should probably pick that one. So a very simple list of options and their descriptions, possibly with some descriptions in between.
- I agree that most of the transformation capabilities of XSLT are not necessary for this purpose, and it should be possible to just define one acceptable output layout.
- It's really a question of which of the alternatives is the easiest to implement, to work with, and to maintain in the long run.

#11 - 08/12/2012 07:12 PM - Roland Schulz

From looking at lightweight formats there two tools which have support for man, (x)html and Latex. These are txt2tags and pandoc.
Thus alternative 4 of Teemu has to sub-alternatives:

1. txt2tags

- it uses Python and txt2tags itself could be easily be auto-downloaded. Thus developers wouldn't need to manual do anything to get full docu support (assuming Pythong as available as we assume Perl is available for the regressiontests).
- has conversion to plain text
- no build in support for math equations or bibliography
- has support for 'tagged' mark which gets unparsed passed to backend. Can be used to write Latex equation for Latex backend. But these 'tagged' regions can unparsed passed to all backends. And txt2tags doesn't generate the tags required for MathJax to render equations. Probably requires some extension and I couldn't find anyone having already done one.

2. pandoc's markdown

- uses Haskell which developers would need to install from their packaging system to get full docu support. Without Haskell no man pages would be generated and documentation wouldn't be validated until uploaded to Jenkins.
- has Latex as input format. Would make it easy to convert existing Latex files if we ever want to integrate the manual and the man-page system more closely.
- no conversion to plain text - aim of Markdown is that it can be well understood as plain text. Thus there would be no dependency for console output or any code to convert (removes the last 3 requirements of those listed by Teeemu for 4.) But we might want to use markdown->markdown for auto-linebreak.
- support for math and bibliography

If we ever need to use xslt for any (future) output format this would be possible. Both support writing to XML (e.g. Docbook, XHTML) which can be used as input for xslt. An example is https://github.com/miekg/pandoc2rfc/

Since we need math formulas txt2tags is only an option if someone makes the effort to make it work with Latex equations rendered nicely in (X)HMTL (preferable MathJax).

#12 - 08/12/2012 08:50 PM - Teemu Murtola

Just a few quick comments.

Roland Schulz wrote:

> - it uses Python and txt2tags itself could be easily be auto-downloaded. Thus developers wouldn't need to manual do anything to get full docu support (assuming Pythong as available as we assume Perl is available for the regressiontests).

I think that Python would be useful for some other developer-only tools as well, so I don't think requiring that for developers would be a big deal.

> - no conversion to plain text - aim of Markdown is that it can be well understood as plain text. Thus there would be no dependency for

console output or any code to convert (removes the last 3 requirements of those listed by Teeemu for 4.) But we might want to use markdown->markdown for auto-linebreak.

Without the auto-linebreak (which needs a basic implementation for my 3 last requirements), we need to either

- force developers to write the in-source comments formatted to a fixed line width (it may be possible to automate reformatting the in-source comments in-place, though), or
- live with the fact that *all* users, not just developers, who don't have pandoc installed will see uglier line breaks in console output.

One additional point is that most of work that goes to implementing option 2 can be reused without much effort if we later decide to use 3/4. This holds true as long as the external tool remains the same, so it's still worthwhile to consider the alternatives at this point. But work that goes into implementing option 1 is mostly wasted if we want to switch to any of the other alternatives.

#13 - 08/12/2012 09:17 PM - Roland Schulz

I agree that auto-linebreak would be a good idea. And since we can simply put that into the build system and read the console output from the txt file if available this would be very easy to do and would fulfill the 3 requirements.

I think the most work to implement the pandoc alternative would be to make Options write out Markdown. Thus it seems to me that not that much of option 2 could be reused.

It seems to me the implementing the pandoc option would not be much work and only has the disadvantage that one wouldn't have auto-linebreak or man pages if one uses the development/git version and doesn't have Haskell. I'm not sure how much work the other alternatives are. But it seems more (1 and 2 requires also that Options write out 1 or even more formats) and for 3 we need the xslt transforms.

#14 - 08/13/2012 02:51 AM - Roland Schulz

I tried out markdown a bit. I got something descent fast by converting the html to markdown.

I few annoyances.

- It doesn't convert greek unicode characters automatically in $\letter$. It should work if xelatex is used as backend (but I don't know anything about using unicode in Latex). Otherwise one has to use $\letter$ for greek characters. This is automatically converted into the unicode for man/html.
- [link1] [link2] is misunderstood as one link with description. One has to have [link1] [link2] (double space).
- bold is

  `**some word**`

  . Single asterisk is underlined.

I left the table for the file options. It seems to me nicer even for the man page. I uploaded this test to https://github.com/rolandschulz/gromacs-pandoc-test

#15 - 08/13/2012 06:45 AM - Teemu Murtola

For option 2, one needs to choose the external format (let it be E), and then

- implement conversion from the current in-source markup to E (similar to what is now in wman.cpp),
- implement writing of option lists in E from Options,
- implement an overall mechanism for writing the help from g_ana help to a file,
- choose an external tool (let it be T) to convert E to the desired output formats, and
- implement a build system that runs g_ana_help, passes the files produced to T, and incorporates the generated documentation into appropriate places.

If E is Markdown and T is Pandoc, all of this except the first bullet is also required for option 4.

As a separate comment, the current wman.cpp system also writes out shell completion files. This we can't get easily from a Markdown-based format.

#16 - 08/13/2012 06:30 PM - Roland Schulz

Yes if Options is directly writing the external format then indeed all but the first is the same as in 4 (I thought that the current format was supposed to be still the format used to generate documentation). But then I think it would make more sense to write all new documentation also in the new format. And then option 2 and 4 are the same as long as 4 includes a function in wman.cpp to write out the new format so that the old documentation still works.

#17 - 08/13/2012 07:39 PM - Teemu Murtola

Roland Schulz wrote:

> (I thought that the current format was supposed to be still the format used to generate documentation)

That would be option 1 (where separate Gromacs-specific code is used to write out each documentation format from our own markup; it doesn't really make a difference whether that code is embedded in the binaries or as a separate binary).

> But then I think it would make more sense to write all new documentation also in the new format. And then option 2 and 4 are the same as long as 4 includes a function in wman.cpp to write out the new format so that the old documentation still works.

The essential difference between options 2 and 4 is that option 2 allows producing the console output using the existing code without any external files, any external dependencies, or extra markup present in the console output. And it avoids introducing a new markup syntax (if the new markup would not have sufficient advantages). But on the other hand, it uses the current, less flexible markup syntax. Difference between 2 and 3 (if 2 would be implemented using XML as the external format) is even less; only the part that produces the console-formatted markup is different.

#18 - 08/13/2012 08:27 PM - Teemu Murtola

BTW, I tried perhaps for nearly an hour to get pandoc working on my Mac OS X 10.5, with no luck. Binary packages don't work, and neither does installing it from source using the Haskell platform. So I unfortunately can't contribute much to the discussion about the details of how it could or couldn't be helpful in this case.

#19 - 08/13/2012 09:22 PM - Roland Schulz

If it doesn't work easily on Mac than we shouldn't use it. Did you try the binary package provided from them (http://code.google.com/p/pandoc/downloads/list) or from fink/macports? Did you try "cabal install pandoc" after installing Haskell from http://hackage.haskell.org/platform//mac.html? Do you have Gentoo Prefix installed to try that? I can try to install it on the Jenkins Mac to see how easy it is. Because easy install on all (common) install platforms is IMO very important.

#20 - 08/14/2012 05:47 AM - Teemu Murtola

I tried the binary package from pandoc website, but it gave some dyld-related errors, suggesting that it needs a more recent OS X version.

I also tried installing the Haskell platform from the hackage. The (old) version that supports OS X 10.5 did install, but using cabal install failed to compile some dependencies, and never finished to compile some others (and took up all available memory on my laptop). After that, I gave up.

All in all, it may work just fine from the binary package on a more recent OS X version, but I really don't want to put much effort into upgrading the OS on my old laptop that is already suffering from some hardware glitches as well...

If we still decide to use Markdown/Pandoc, I can maybe still do some of the basic implementation, but someone else needs to test it, and figure out and implement all the details on how to best use these tools to produce the documentation we want.

#21 - 08/14/2012 06:50 AM - Roland Schulz

Christoph showed me how to write txt2tags filters and that makes it easily possible to write math equations.

```
%!preproc(tex): '(\$[^$]*\$)' ""\1""
%!preproc(man): '\$([^$]*)\$' \1
%!preproc(html): '(\$[^$]*\$)' ""\1""
```

is sufficient to get Latex equation to work. txt2tags doesn't have that much more feature then our current syntax but we also don't really need more (besides maybe bibliography as nice to have). So given that Haskell is difficult to install at least on old Mac OS it might be better to use txt2tags if a lightweight markup language is used for either 2 or 4. Given that txt2tags doesn't have much more feature and is also mostly a simple search-replace based approach, it might be an argument to stay with the current format. On the other hand, txt2tags (either option 2 or 4) removes the need to write our own output format writers, and would allow us to write (at least new) documentation in a format easier to read/write. Thus it might be worth it.

For math an alternative to using Latex syntax would be http://www1.chapman.edu/~jipsen/mathml/asciimathsyntax.html (example: http://skypewalkerconstitution.blogspot.com/2009/07/comparison-between-asciimath-and-latex.html). It is easier to read as plain text (console, man) and can also be rendered by MathJax and by Latex: https://github.com/judah/asciimath-tex.

#22 - 08/14/2012 07:23 AM - Roland Schulz

Another option would be reStructuredText/Docutils (http://docutils.sourceforge.net). It has a similar feature set as Pandoc but is Python and thus doesn't have install problems.

#23 - 08/14/2012 03:30 PM - Roland Schulz

reStructuredText is extendable like XML. One could add a custom directive to describe a file argument and one to describe an option. That would allow us to write out the shell completion rules also from the same document. This would be a mixture of option 3 and 4. Use custom tags where useful but use an existing syntax for general formatting. Whether writing transform rules in XSLT or Python is better is probably a matter of taste.

#24 - 08/14/2012 09:01 PM - Teemu Murtola

Docutils doesn't seem to have plain text as an output option, and no passthrough mode either for line wrapping. If we want to keep the documentation in the source files as it is now (an array of string literals), it is quite inconvenient to do line wrapping to a fixed width manually. C++11 raw string literals would be useful here, but don't know how good compiler support is. I don't see the plain text output option as a must, but this point should be also considered. Most of the markup in the proposed formats doesn't disrupt the flow of the text or add too many cryptic symbols, and reStructuredText is perhaps better in this regard than some others.

#25 - 08/15/2012 01:22 AM - Roland Schulz

No it indeed doesn't have something like this build in. But it is easy to write a new writer. And one which strips all formatting but titles/headers and does line wrapping is available at: https://github.com/benoitbryon/rst2rst/blob/master/rst2rst/writer.py (I had to remove the unused import unichar2tex to test it). We could modify that easily to not strip all formatting but keep a select few.

#26 - 08/15/2012 06:32 PM - Roland Schulz

Pandoc would let us convert our user manual to restructeredText (rest). And breathe would let us convert the doxygen output to rest (we would of course keep using Doxygen input format). This would allow us to generate one developer documentation document containing all 3 documentation sources. This might be useful because it would allow to create internal link between the all 3 sources.

Of course the user documentation wouldn't contain the API documentation. But even the user documentation would benefit because we could target not only PDF but also HTML (and potentially other format) for the manual. Of course it is possible to generate HTML from Latex but all direct converters known to me cause extremely ugly output. Combining first all output to Rest would us allow to create both a nice PDF user manual and also a nice HTML version of it.

In summary:
- txt2tags according to KISS. It is not a full fledged grammar/parser and uses (mainly) search-replace. It doesn't have some of the more advanced featured (e.g. bibliography). It doesn't validate the input and doesn't use a conversion pipeline. It is small and simply thinks are extremely easy and fast to do. It already has a conversion to plaintext. The txt2tags syntax isn't supported by any pipeline/parser converter (docutils/pandoc) and thus we would be somewhat locked in. One could add bibliograph support with the custom extension framework.

- rest is similar to XML. It has clearly defined tags and can be validated. Docutils uses a pipeline and has advanced features (e.g. bibliography). Multi-page documents can be done with sphinx (http://sphinx.pocoo.org/) and we could combine all 3 types of documentation into one format. It is possible to write the shell completion from rest with custom tags. The added complexity of the much larger framework might make it more complicated if non of the advanced feature is really needed.

#27 - 08/15/2012 09:08 PM - Teemu Murtola

I would be fine with using either of these. Some things to still consider:

- If we have our own extensions to docutils, this means additional maintenance as the API may change (it is stated to be experimental, although I don't know how much it may change in practice). And we may also need to check that the version of docutils is compatible with our custom extensions. Do you have some idea on how you would like us to use docutils? Rely on it being installed on the system and try to find it using cmake, or use some auto-download system? docutils without tests is something like 2 MB of code. Most of docutils is public domain, but some individual files are under separate licenses, which may make it difficult for us to include it in our source tree, but I don't think we need to. What's the license on the rst2rst script you linked to?
- txt2tags is ~360 kB of code in a single Python file. That would be possible to include even in git if necessary, or use an auto-download (which would be simpler to set up since it's just a single file). License is GPL v2.
- Even txt2tags has a lot more features (at least for basic formatting) than the current markup. Since it uses regexps for the search/replace part, it can do a lot more than the current markup, which only replaces fixed strings, and does that independenty for the start and end tags. But I agree with your arguments why rst would be better.
- As an unrelated comment, Doxygen has added support for Markdown (but only in version 1.8) in favor of its own markup (which is still supported, though). Comments currently in the code use the old markup (because many of them were written before 1.8 came out, and because Jenkins is still using pre-1.8 version).
- I think it would be a major benefit if we didn't need to take option 1 and implement another set of formatting routines for producing all the different output formats from an Options object.

#28 - 08/16/2012 01:56 AM - Roland Schulz

- there are several rst2rst scripts. Another one is http://code.google.com/p/tobogan/source/browse/trunk/rst2rst.py?r=13. It is from 2007 and still runs fine on docutils from debian/testing. It is GPL2. For the one I linked to first we need to ask for the license because I couldn't find one either.
- I would simply rely on the developer to install it. Since it it python I hope it is easy for everyone. Is that correct? If cmake can't find it, it would simply disable docutils and one would get the Rest formated docu as is. As discussed the user wouldn't need it at all.

I didn't know about doxygen moving to markdown. There would be an obvious advantage of using the same syntax for all Gromacs documentation. Additional Doxygen moving to Markdown emphasis my feeling that Markdown is getting to be the most popular format and so people probably already know it independent of Gromacs. This advantage is somewhat reduced by the fact that different Markdown parsers have different incompatible extensions (e.g. there is no agreed upon standard for tables).
Given that it would probably be sufficient to generate man pages and html by Jenkins it might be worth looking at an alternative, in which pandoc is only required on Jenkins (for html/man) and a Python parser is used for Markdown to beautified Markdown or simple text with potential some Markdown formatting for console output. Thus the developer would not need pandoc for nice console output but would need to wait on Jenkins to see the documentation in html/man format. Two Markdown python libraries (http://freewisdom.org/projects/python-markdown/, http://code.google.com/p/python-markdown2/) and two C libraries (http://www.pell.portland.or.us/~orc/Code/markdown/, https://github.com/jgm/peg-markdown) exist. https://github.com/apenwarr/redo/blob/master/Documentation/md2man.py is an example of custom writer for the first python library. I think we could modify that for our purposes of a plain-text/light-markdown writer.
The advantage of that approach would depend on whether we even want to switch to Markdown for the Doxygen comments. Also has the disadvantage that it doesn't let one define custom tags like in rst or xml (philosophy is: be easy to read in plaintext). Thus it probably woudn't be a good idea to add custom tags for options/files and thus e.g. the shell completion would need to be generated separately.
I asked a friend to verify the problem that the official pandoc binary doesn't run on Mac OS 10.5. I filed in issue https://github.com/jgm/pandoc/issues/587. Maybe if they resolve that quickly that might also be still a viable option. I'm happy to upgrade Doxygen on Jenkins to 1.8 if you want me to.

#29 - 08/16/2012 07:47 PM - Teemu Murtola

Again, some more comments:

- I think that installation of Python packages shouldn't be a problem for developers, and docutils should be available in many distributions as well.
- Is it a good idea to use multiple different Markdown parsers? As you say, they may have different extensions, and setting up support for multiple tools is always more work (and is also more difficult to understand).
- For Doxygen, I don't think that there is any hurry or particular reason to switch to Markdown. Haven't tried whether it would make some things easier, but there isn't that much formatting used in the current documentation; mostly just bullet lists, and occasional example code blocks. The implementation in Doxygen is also a bit confusing; writing, e.g., check*() in the documentation produces a warning about a mismatched tag, which took me some time to figure out... We should at least wait until Doxygen 1.8 has been readily available in major distributions for some time, so that things will simply work even for people who are not on the bleeding edge with their software.
- If we want to do some customization, docutils (and txt2tags) have the advantage that they are written in Python, which is readily understood by a lot more people than Haskell.
- If we need the external tool (e.g., docutils) for console output, then actually the user *does* need it. Namely, we ship the template under share/template/, and when compiling their own analysis tools, the users are also exposed to the mechanism that we use to generate the console output. Judging from occasional posts on the mailing list, even relatively beginning users may want to write their custom analysis, so I think this part should also "just work". Ideally such that if you copy the source for one of the analysis tools and modify it, the command-line help still works identically. I don't think we need to support generating man pages or html help (easily) for user tools, but the console output everyone will see. I think this aspect hasn't been yet included in the discussion, and it does complicate the implementation. For example, if we store the preformatted console help in text files under share/, we need to be 100% sure that they are not used to replace the help text in user-written tools, even if those tools have the same names as our internal ones. And we need to support a similar mechanism for the user-built tools.

#30 - 08/17/2012 12:58 AM - Roland Schulz

- You are right, multiple Markdown parsers will be a problem if we use extensions. And since tables is already an extension we will need extensions. So this probably wasn't a good idea.
- (simple non-multi level) Lists are anyhow the same between Markdown/RST/Doxygen thus it is not an issue. If the only other thing regularly used is code blocks we probably don't gain much by having the same syntax and it isn't an important advantage for Markdown
- You are right that we need to make sure that it also works for user-build tools/modules. But I'm not so sure about your example and whether we need to support that it works correctly, if the user makes a copy without changing the name. It would be nice but if it is difficult I don't think it is a high priority. I think it is reasonable to say that the documentation requires that program names are unique. And other than that user tools hopefully work the same way. The console output would come from our version of rst2rst if docutils is available and otherwise it would be directly the original rst.

#31 - 08/17/2012 06:25 AM - Teemu Murtola

It could be worth developing the idea of console output using rst2rst (or any other similar approach) a bit further to have a clearer picture on the possible problems. Will just use rst2rst as an example of the external formatter below, any other formatter should work more or less the same.

There are two different types of strings that may contain markup in the source:

- Program description and other long text blocks (e.g., the interactive help for selections).
- Descriptions for individual options. I imagine it would be useful to allow markup also here, but if it is too difficult, it could also be an option to make them not-so-well supported. Some notes below where this may complicate things.

I think there are basically three options:

1. Just print the in-source strings as they are.
   - This option needs to be implemented in any case as a fall-back.
   - I think we still need to implement line wrapping for the option descriptions. This will be simpler, as I don't think it is necessary to allow complex paragraph-forming markup here.
2. Run rst2rst every time the tool is run to format the output from the in-source strings.
   - For option descriptions, we need to either run the tool separately for each of them, or combine all of them together with the description and have some special tags in the input/output such that they can be parsed back.
   - rst2rst needs to be available at runtime, where the environment may be different than at build time. In particular for mdrun on HPC systems, this may be a problem. The tool needs to detect at runtime whether rst2rst is available, which puts in additional complexity and likely OS-specific code.
3. Cache the output of rst2rst to an external file, e.g., under share/gromacs/help/.
   - Again, to support option descriptions, they need to be specially tagged, since I don't think we want to put each into a separate file. Probably one file per one long text block is a reasonable first approach to avoid reading all of the text each time one tool is run.
   - A mechanism needs to be developed that finds the correct text file for a program. The binary name by itself isn't sufficient, as I think that we really want to have a single wrapper binary at least for the tools. The current g_ana could also be easily extended to cover other programs than just trajectory analysis tools. There are also help texts not directly associated with the tool (currently, the mentioned selection online help texts).
   - The cached file would be generated either
     - by the build system, preferably from the same source as is used for the other output formats, or
     - by the tool using a special flag, or on first use.
   - It's not clear how this would work with user-written tools. Where would the cached text file be stored and looked up from? The look-up code will be shared with the Gromacs-provided tools, and depending on how the cache generation is implemented, it may be as well (if it is done by the build system, then we have more control).
4. Implement a simple C code that does the console output directly from the in-source strings.
   - This option does not need any fallback.
   - This "simple" code potentially becomes more and more complex over time, as we take more features of the selected markup syntax into use. C(++) isn't the best suited language for this kind of task. Still, some basic features should be possible to implement quite easily.

- In all cases (except the first and the last), we need to install the rst2rst if it is written by us.
- I don't think we want to store the console-printed option tables in the cached files, or even process them through rst2rst as a whole. We probably want different output formatting for the option tables in console output than in other documentation formats. For console output, it is more important that they are concise, while other formats can be more verbose to describe more details. And it's easier to generate the tables in, e.g., rst format, if one doesn't need to also worry about them fitting to the console width. Currently, these tables are also dynamic in that they show the values provided by the user, and not the default values.

#32 - 08/18/2012 05:41 AM - Roland Schulz

- If we put in the effort to do line wrapping (and you are probably right it isn't that difficult), it might be worth considering using only very simple search-replace for the console output. I think most markup emphasis (*), lists, tables is fine. Removing extra tags (e.g. :math:), escaping (e.g. \: to :) would be very easy. If that is enough this approach might be the easiest. Otherwise I wouldn't try to make the fallback nice given that Python is so widely available and one really doesn't need the console help on a compute node of a HPC machine.
- rst has build in support for options list: http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#option-lists . It doesn't have support for a type. But it would be possible to include that in the description (if we want it to be parse-able using a role). It seems to me that often the type isn't needed anyhow. If we don't like the format we could add our own directive. Either way we could create one document which can be converted but still parsed.
- In theory we could run rst2rst using the C-Python API, but that is overkill (unless we anyhow have it anyhow in the future, to support e.g. Python trajectory-analysis modules). It seems we could just simply try to run rst2rst and if fails (e.g. because Python isn't avaiable at run-time) just use the fallback. Running is an external OS dependent command but relative simple.
- If we can get easily implement it to only have option 1, we need option 3 for source distributions so that normal users don't require docutils. Option 2 would be optional and should only be don't if it is trivial and/or Option 3 anyhow uses Option 2 to generate the Cache file.
- I think if we have Option 2, we shouldn't support Option 3 for user generated tools. If we don't have Option 2 we could put the cache file always at ../share/{binary-name}-{module-name} relative to the binary location and add the code to generate the cache file to the template cmake file.

#33 - 08/18/2012 12:23 PM - Teemu Murtola

- I updated my comment 31 to add numbers to the options, and also listed a fourth option (didn't want to change the order of the existing options, although the new could most naturally be the second). If we think that a simple search-replace would suffice for the console output, then I would suggest we do it using that option to avoid most of the potential issues. To do proper line wrapping, the line wrapping code needs to recognize "hard" linebreaks in the input from "soft" linebreaks, and this needs special handling for each markup construct that introduces line breaks. Paragraph breaks are easy, but lists, tables, headings, literal blocks etc. are slightly more complex. Could still be doable, though.
- Running a system command is easy, but is there a platform-independent way of capturing the output and processing it further? I don't think it would be nice if some operating system or Python error messages were printed every time one tries to run a Gromacs program without rst2rst available. We also need to do further processing if we want to process markup in option descriptions. Unless we first format the option list as a rst table, then try to pass it to rst2rst, and if that fails, reformat it separately, but that is starting to become quite complex.
- I agree that at least for non-console output, we could easily go with a simple option list that does not have the type explicitly. The type could be indicated otherwise, e.g., like

```
-rcut REAL
  Specifies the cutoff.
```

- I think that the Gromacs-specific cached files for option 3 could easily go under ../share/gromacs/help/. I think that there is value in keeping the user tool build system as simple as possible (such that the binaries can simply be generated in the same directory as the source, and that there is no requirement on directory structure otherwise). But of course we could make also the template CMakeLists.txt and Makefile build the binary under a bin/ subdirectory, and create that share/gromacs/help/ as a separate subdirectory. I just feel that some beginning users could be confused by this complexity.

#34 - 08/19/2012 12:31 PM - Teemu Murtola

I was originally thinking to treat each string in the in-source description array as a single line, but it may be actually simpler to force the user to add explicit \n in those places where the markup needs those. If it is done this way, then the line wrapping code should be quite easy to implement without any external dependencies, with just some special treatment for bullet lists and possibly some other constructs. To make things a bit easier to write, an empty string in the array could be easily treated as "\n\n" for a paragraph break. Some other exceptions can also be added if they are needed often. But besides the paragraph break, I don't think there would be that many cases where explicit line breaks would then be needed, so it would not be a big inconvenience to have to write those. And since the issues caused by missing \n would be immediately obvious in the console output, it should be easy to detect even without Jenkins.

#35 - 08/31/2012 01:03 PM - Teemu Murtola

I did a draft of using reStructuredText for the in-source markup format and for exporting it. It's very preliminary, but it could support the discussion. And it should be relatively easy to adjust that to try out also other output formats. I pushed it to refs/private/tmurtola/help-export on gerrit; for some reason, I could not push it as a draft to Gerrit directly (perhaps because there are multiple commits in the branch). Gerrit says (the last commit in the log is already in gerrit):

```
$ git log HEAD~5.. --oneline
c1abd09 WIP: Export for help texts from g_ana.
9008dae Improve markup substitution in HelpWriterContext.
e9ef25a Add subsection depth to HelpWriterContext.
cd37c62 More features for TextLineWrapper.
62b3db7 Reorganize CommandLineHelpWriter implementation.
$ git push gerrit HEAD:refs/drafts/master
Total 0 (delta 0), reused 0 (delta 0)
remote: Processing changes: refs: 1, done
To ssh://tmurtola@gerrit.gromacs.org:29418/gromacs.git
 ! [remote rejected] HEAD -> refs/drafts/master (no new changes)
error: failed to push some refs to 'ssh://tmurtola@gerrit.gromacs.org:29418/gromacs.git'
```

Some comments from the process:

- The rst option list format is quite inflexible, and doesn't support the way command-line options are written in Gromacs. It simply refuses to recognize a list as an option list if it does not conform exactly to the parser's expectations, and parses it as a definition list instead (whose syntax it does not exactly conform to either...).
- To produce nice-looking output, it may be necessary to use at least .. compound::, and that is a bit more difficult to parse correctly in the console output code.
- Haven't tried math (looking at the source code, at least rst2man seems to produce warnings if it encounters a :math: role), nor haven't put much effort on the build system.

#36 - 03/23/2013 01:08 PM - Teemu Murtola

I now added a separate draft that implements the alternative of keeping everything like they were earlier. So now gerrit has a few different RFC commits related to this issue:

1. https://gerrit.gromacs.org/#/c/1335/: reStructuredText as both in-source markup and as the exported format.
   - To continue on this path, the RST output should be cleaned up such that it produces nice-looking documentation, and the build system should be adjusted to incorporate the output into, e.g., the PDF manual build. Mostly CMake and docutils-related work. This approach is also relatively easy to adapt to other output formats (e.g., Markdown) if we want to try those. The C++ code needs to be changed if/when
     - we need to produce different kinds of headers in the RST output for different purposes, e.g., if we want to make HTML pages appear different from those produced from the current output format (which is tailored for man pages).
     - we need to add RST constructs that we do not want to print out verbatim to console output (e.g., .. compound:: or :math:). Possibly also if we want to format itemized lists better in the console output.
2. https://gerrit.gromacs.org/#/c/2249/: the "old" way, i.e., Gromacs-specific in-source markup and C++ code to write out each different format.
   - This requires more work than the RST to clean up things. Build system support is missing from here as well. The C++ code needs to be changed for nearly everything:
     - Each output format needs to be thought out separately and the formatting fixed.
     - If we want to add support for more formatting constructs (e.g., itemized lists), we need to add quite a bit of code, as the search&replace approach in wman.cpp doesn't work very well for all of these.
3. https://gerrit.gromacs.org/#/c/2248/ and its dependencies: some prerequisite refactoring that is common to both of the above. Split into separate commits to make the above two easier to compare.

It is possible to reduce the work a bit if we for now leave the selection help (produced by gmx help selections etc.) out of the scope for now.

#37 - 03/24/2013 07:44 PM - Teemu Murtola

- Assignee set to Teemu Murtola

#38 - 03/27/2013 05:50 AM - Roland Schulz

Erik suggested before to get rid of man pages. I don't really care either way. But if that would simplify this issue than it might be something to consider.

#39 - 03/27/2013 08:09 PM - Teemu Murtola

I don't think that getting rid of man pages would simplify the situation much, unless we also get rid of html and/or latex output. Man pages are the only output format that currently actually more or less works in both of the drafts. ;) It would remove some code from both alternatives, but I don't think it would really solve any of the harder problems. What we would need now is a decision: whether to

1. Dig into docutils to find out how to best use it to produce nice documentation in all the formats we want, integrated the way we want to our existing infrastructure (the PDF manual, the online manual etc.). The main issues are to format the option list and to produce custom headers/footers for the produced HTML/latex documents so that we can integrate them. Other stuff should be relatively straightforward, even though it may require some work.
2. Investigate some other markup processor.
3. Go with the pure C++-based approach. This is just grunt work, and everything extra that we want to support simply needs to be implemented. Can be done in parts, but there can be a lot of code in the end...

From this point, very little of the work between the alternatives can be shared, so I don't want to start doing a full-fledged implementation for any of these until I can be reasonably certain that it won't cause a storm of complaints directed at me several months after I've implemented it... I really don't have any strong opinion on this, so I'm a bit reluctant to do the decision myself.

#40 - 04/26/2013 05:44 AM - Teemu Murtola

- Status changed from New to Accepted

#41 - 04/26/2013 05:44 AM - Teemu Murtola

- Status changed from Accepted to In Progress

#42 - 04/26/2013 05:44 AM - Teemu Murtola

- Status changed from In Progress to Blocked, need info

#43 - 12/25/2013 06:59 AM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #969.
Uploader: Teemu Murtola (teemu.murtola@gmail.com)
Change-Id: I25143ddfa4226abd5ae5b95172f69c8fc615b720
Gerrit URL: https://gerrit.gromacs.org/2932

#44 - 12/31/2013 07:09 AM - Teemu Murtola

- Project changed from Next-generation analysis tools to Gromacs

- Category set to core library

- Status changed from Blocked, need info to Fix uploaded

https://gerrit.gromacs.org/#/c/2249/ contains the basic implementation. The implementation could still be cleaned up etc., but marking as Fix uploaded, since most of the work and functionality is done. Shell completion discussion continues in #1410.

#45 - 12/31/2013 07:09 AM - Teemu Murtola

- Related to Feature #1410: Future of shell completions added

#46 - 01/20/2014 06:30 PM - Szilárd Páll

Had a look at change 2249 and I have a few questions/comments regarding the man pages:

- We should install the man pages *by default*.
- It would be useful to have a *brief* "-h" help output with usage and brief option description; e.g. see git commit -h vs git commit --help: 45 vs 450 lines.
- SYNOPSYS: file arguments are normally not listed with the default values, but in a generic way, right? Also, the current formatting is somewhat off of what the synopsis typically looks like = lacks the [] and | hints; additionally, the different forms of invocation would be nice to separate (MD, EM, etc.). Moving toward a more standard SYNOPSYS section would be useful not only for the sake of standardization, but also because IMO it'd make it more readable and suggestive.
- OPTIONS: AFAIK all "-foo/--foo-long" switches and options typically go to this section which means everything currently in "FILES" belongs to "OPTIONS".
- FILES: seems to be for the file-arguments, not the options with file arguments. We don't have many (any?) plain file arguments, I think.
- There are a few stray spaces at the beginning or end of lines (e.g. man gmx-mdrun DESCRIPTION 1st par 1st line, 2nd par 2nd line both beginning and end). Would it be easy to strip these away?
- How/where should we document relevant env. vars - often there is a separate section for this (see e.g. man git)?
  - Could be nice to add in the future a few more sections like "FURTHER DOCUMENTATION", "REPORTING BUGS", etc. (got inspired by man git). This will require code to format new sections, right?

#47 - 01/20/2014 06:46 PM - Teemu Murtola

Szilárd Páll wrote:

- We should install the man pages *by default*.

We already do, if you compile from a source distribution. If you want to have this on for dev builds from git, please suggest how we should handle cross-compilation etc. such that failure to execute the gmx help -export man would never cause the whole build to fail by default.

- It would be useful to have a *brief* "-h" help output with usage and brief option description; e.g. see git commit -h vs git commit --help: 45 vs 450 lines.

This is on my nice-to-have lists, once we first get rid of the duplicated help writing code (wman.cpp vs. cmdlinehelpwriter.cpp) and get the basic functionality working nicely with the wrapper binary.

#48 - 01/20/2014 06:57 PM - Szilárd Páll

Szilárd Páll wrote:

> Had a look at change 2249 and I have a few questions/comments regarding the man pages:
>
> - We should install the man pages *by default*.

- these only need the binaries, so there can be no problem with extra dependencies. If I understand correctly, GMX_BUILD_HELP triggers building man + HTML + pdf. Could we set GMX_BUILD_HELP=ON by default? The only drawback I could see while testing is that the user may get a "Could not find Doxygen" message which is not particularly harmful.

#49 - 01/20/2014 07:05 PM - Szilárd Páll

Teemu Murtola wrote:

> Szilárd Páll wrote:
>
> - We should install the man pages *by default*.

> We already do, if you compile from a source distribution. If you want to have this on for dev builds from git, please suggest how we should handle cross-compilation etc. such that failure to execute the gmx help -export man would never cause the whole build to fail by default.

Good point. Still, I'd prefer to not have to manually set GMX_BUILD_HELP=ON just to get the man pages installed. I think we could come up with a simple solution to test for cross-compile; would it not be enough to try to either build and run a test program or run gmx -h and if it fails we can assume that the build is either broken or it is cross-compiling. A note could reflect this assumption and this could trigger GMX_BUILD_HELP=OFF. Does this sound too complicated?

One more thing: with GMX_BUILD_MDRUN_ONLY=ON we should install a share/man/man1/mdrun.1 (or perhaps symlink to gmx-mdrun.1?).

#50 - 01/20/2014 08:19 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #969.
Uploader: Teemu Murtola (teemu.murtola@gmail.com)
Change-Id: I5b38d7ac6d7c58c3decce64ddfe60fff0a5797ed
Gerrit URL: https://gerrit.gromacs.org/3007

#51 - 01/20/2014 10:05 PM - Szilárd Páll

Szilárd Páll wrote:

> Good point. Still, I'd prefer to not have to manually set GMX_BUILD_HELP=ON just to get the man pages installed. I think we could come up with a simple solution to test for cross-compile; would it not be enough to try to either build and run a test program or run gmx -h and if it fails we can assume that the build is either broken or it is cross-compiling. A note could reflect this assumption and this could trigger GMX_BUILD_HELP=OFF. Does this sound too complicated?

Actually, I came to think that no matter what the requirement for generating the help is to be able to execute the compiled binaries. Hence, a simple cmake "cross compile test" that always checks whether the binaries can be executed should be useful regardless of whether GMX_BUILD_HELP=ON is default or not.

#52 - 01/20/2014 10:14 PM - Szilárd Páll

Just realized that the main binary is called gmx, but we install share/man/man7/gromacs.7. As one would typically expect to have a man page corresponding to the binary's name, perhaps we should add a gmx.7 -> gromacs.7 symlink.

#53 - 02/04/2014 07:53 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #969.
Uploader: Teemu Murtola (teemu.murtola@gmail.com)
Change-Id: I900883eb39853ed67d5f6177ae06847283348d9e

Gerrit URL: https://gerrit.gromacs.org/3078

#54 - 02/04/2014 07:54 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #969.
Uploader: Teemu Murtola (teemu.murtola@gmail.com)
Change-Id: Ic09ac91b3d5b5e2d42d41b83935d54c894eb8e97
Gerrit URL: https://gerrit.gromacs.org/3079

#55 - 02/07/2014 07:55 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #969.
Uploader: Teemu Murtola (teemu.murtola@gmail.com)
Change-Id: Ib77543367c38803ef186f6024a1af14feb806d80
Gerrit URL: https://gerrit.gromacs.org/3119

#56 - 02/14/2014 08:01 PM - Teemu Murtola

- Status changed from Fix uploaded to Resolved

I think this part is now done, including shell completions. Also old code (t_pargs/t_filenm-based) gets redirected to the new help formatter. I'll create a new issue with Szilard's comments about the man pages, since they aren't really related to this functionality. The new formatter just generates something that more or less resembles the old; changing that is a separate issue. May take some time, though, to gather all the content for the new issue and write something sensible there.

#57 - 02/15/2014 06:09 AM - Teemu Murtola

- Related to Feature #1437: Online help formatting improvements added

#58 - 02/24/2014 11:00 AM - Rossen Apostolov

- Status changed from Resolved to Closed