Gromacs - Feature #685

Wrapper binary

01/26/2011 04:08 PM - Christoph Junghans

<table>
<thead>
<tr>
<th>Status:</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>Teemu Murtola</td>
</tr>
<tr>
<td>Category:</td>
<td>build system</td>
</tr>
<tr>
<td>Target version:</td>
<td>5.0</td>
</tr>
</tbody>
</table>

**Description**

Is -1 $GMXBIN/g_* | wc -l -> 142 (for single and double)

Already at this point it gets a little unmanageable to remember the names of all binaries. I think, it would be nice to have something like a global gmx binary, which has the program trunc as the first option (e.g. "gmx dist", "gmx rdf").

One could start with something like what git does. Install one binary in /usr/bin, which calls the others from /usr/libexec/. And for those user, who prefer the old way of calling the apps, one can create symlinks, (e.g. g_rdf -> gmx, like busybox does).

This is just an idea for gromacs 5.0 or later.

**Related issues:**

- Related to Gromacs - Bug #219: Remove generic names from GROMACS binaries
  - Closed 10/02/2008
- Related to Gromacs - Task #666: Improve help printing in the command-line runner
  - Closed 01/14/2011
- Related to Gromacs - Task #672: Improve command-line trajectory analysis appl...
  - Closed 01/19/2011
- Related to Gromacs - Feature #1410: Future of shell completions
  - Closed 12/27/2013

**Associated revisions**

**Revision ab300758 - 05/07/2012 12:16 PM - Teemu Murtola**

Generic handling for multiple cmd-line modules.

Added generic methods for multiple independent modules within a single command-line tool, and modified g_ana to use this.

By itself, does not change the external behavior significantly, but makes it easier to implement parts of #666 and #672.

The wrapper binary in #685 should also be easy to implement using this approach.

Change-Id: I3058bf5db2cbb7c7a1f9e4c87ddf2dbb1f9727f8e82

**Revision 1f1c7b66 - 11/10/2012 09:16 PM - Teemu Murtola**

Option to create symlinks to g_ana.

Added an install rule that generates symlinks to g_ana using old tool names. These symlinks directly run an individual module contained in g_ana. These can make it easier to migrate users' old scripts.

For now, only added a link to g_select as a proof-of-concept. The list can be easily extended, and depends on how we want to solve #685.

Related to #685.

11/19/2015
Wrapper module to add old binaries into g_ana.

Implement a module class that allows including old binaries (basically, anything that has a main()-like function and accepts a `-h` command-line argument) into g_ana. As a proof-of-concept, move one tool (g_dist) into g_ana. Will move most of the rest in a separate commit.

To try to keep old functionality working, symbolic links are now created also into the build tree using the old binary names. Also added that on Windows, copies of the g_ana binary are created instead of symbolic links. Required some changes in the CreateLinks.cmake script.

Only man page generation should break with this change as the old gmx_add_man_page() logic does not work as there is no target with the correct name to which add the post-build rule.

Several TODO comments are left, but they can be resolved later.

Related to #685.

Replace most tools with symlinks to g_ana.

Improves link times and reduces the installation footprint with static libraries significantly. Also removes a lot of dummy files. Should not affect functionality.

To make 'g_ana help' more useful in this case, some grouping etc. for the modules needs to be implemented, but is not done in this commit. After this, programs.txt should also be auto-generated from g_ana (this is simpler if/when remaining binaries also get included).

Part of #685.
Revision 264847f5 - 12/10/2012 08:03 PM - Teemu Murtola

Rename g_ana to gmx.

Since all the modules contained in this wrapper binary are no longer analysis tools, a more general name is needed.

Part of #685.

Change-Id: lec0a0ee10daa283510127ce4c1db7b5290d50f58

---

Revision bebc698f - 06/27/2013 07:57 PM - Teemu Murtola

More binaries into the wrapper binary.

Move most remaining binaries to be built into the 'gmx' wrapper binary. Now, only mdrun and g_pme_error (which have a different initialization sequence), and ngmx (and other X11 programs) remain outside.

Renamed a few headers to match the name of their implementation file.

Part of #685.

Change-Id: l3fb7e3de905c01d7564d809893390bb9b8f8368

---

Revision e45b119f - 07/30/2013 06:10 AM - Teemu Murtola

Handle wrapper binary options also for symlinks.

Now the -quiet, -version, and -copyright options also work when the binary is invoked through a symlink (so, e.g., g_angle -quiet suppresses the startup headers). They also work for the user tools, implemented using the single-module wrapper.

Extended the command-line parser to support parsing only recognized options and removing those from the command line while leaving the rest untouched.

Related to #685 and #1209.

Change-Id: I740f70386d89694246c3e25ba0a1c1c4df17dc6b

---

Revision fc034c1d - 08/24/2013 07:50 PM - Teemu Murtola
[RFC] Really uniform startup path for all binaries.

Main changes:
- Move MPI_Init() call from init_par() to a separate initialization method that is now called from the beginning of the wrapper binary. Similarly, make it the wrapper binary’s responsibility to call MPI_Finalize(). Also make this behave uniformly for the wrappers that implement main().
- Call MPI_Abort() on exceptions.
- Move the LegacyCmdLineWrapper from legacymodules.cpp to cmdlinemodulemanager.cpp, rename it, and add methods to access it in other contexts as well.
- Merge g_pme_error into the wrapper binary. Make mdrun and other remaining binaries to use this code path for initialization, even though they still remain independent binaries.
- Remove unnecessary code from statutil.*, now that the common options are handled by CommandLineModuleManager.

All executables now start up through CommandLineModuleManager::run(), making it easy to adjust common behavior.

RFC because I don’t have easy access to an environment where I could test MPI or X11 code; thus I would appreciate it if someone could test these parts (if Jenkins verification is not sufficient).

Part of #685.

Change-Id: I2cb9b0745c084b3d77f95ca163aa6745af01c1bc

---

Revision 70421a8a - 09/13/2013 09:16 PM - Teemu Murtola
Move mdrun into the wrapper binary.

Part of #685.

Change-Id: I92d9ad5705356b2ab1e132383a5fee79ade0f4d0

---

Revision 6603baa6 - 09/13/2013 09:18 PM - Teemu Murtola
Add option to build a standalone mdrun binary only.

This makes it again possible to build a leaner version of mdrun, without the whole wrapper binary. Enabling the option also removes unnecessary
stuff from libgromacs, and suffixes it with _mdrun.

This also replaces install-mdrun: ‘make’ builds only mdrun and
‘make install’ installs only mdrun when the new option is enabled.

Part of #685.

Change-Id: l8d309fadefbdfcdabc484d632a213ae52c558fbf4

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: lca168958136a09bc5995812c4d4d5363d097c2b1

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.
  This simplifies things if it isn't needed.
  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: lbe735711f650eafaecf28fbb1ed92da97dcb81b6
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 reinstated 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 reinstated 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: ld34be9489aa3a90d94cd87f8936b030bc21f1c98

---

Revision 0b9fcb2 - 10/16/2013 08:40 PM - Teemu Murtola

Remove latex help export.

The decision in I7e55a011 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: lae5b82e288e3120ebe6ae38c0e5c3aca892194a

---

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: l37d45549214e49a6edab82fb64ec7b38b9e72094
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: I6c2efe10c53f10f7fde906b3386ddea7fbea34b89

Revision 4c15f726 - 11/19/2013 05:49 AM - Teemu Murtola

Add back mechanism for cross-tool hyperlinks.

The exported HTML pages again can contain hyperlinks to other tools. Instead of replicating the old functionality, made the links of the form [gmx-distance] instead of recognizing plain text words. This requires all the output formats to be aware of them (not a big deal in the current code layout), and makes it possible to define the appearance of the links centrally instead of relying on all links being wrapped in, e.g., [TT]...[tt]. The new link syntax is similar to Markdown implicit links (at least Doxygen accepts them without the second []).

Also added support for [THISMODULE] tag in the help text, which expands to the name of the current module, again allowing the appearance to be defined centrally.

Only changed the help text for gmx-mindist to take advantage of the new mechanism as a proof-of-concept. Bulk work for other tools is better done as a separate commit once the general approach is accepted.

Part of #685 and #1242.

Change-Id: lbd263cc2c131dc18d2a5d9046f6ecc6b1c08734f9
Uniform headers in HTML pages

The version number is now also automatically updated in the page headers for `make html`. Also, fix most links in the file format pages. If we in the future move into some proper HTML publishing system (e.g., based on Markdown), it should probably take care of this and also cross-links between these pages and the generated pages, so not much effort has been put to make the approach or the headers/footers particularly elegant.

Related to #685 and #1242.

Change-Id: I083d1f9714dd68dfc2977799378299a43a05b73

---

Generate gromacs.7 from the wrapper binary

Use the same mechanism for generating gromacs.7 as with the by-topic list of programs for the HTML pages. Now the man page content is up-to-date, no longer requires programs.txt, and uses the new gmx-something names for the referenced programs.

Part of #685

Change-Id: I070f88fd366b37d1ba287a3c2e8e26a846774002

---

Make help texts refer to gmx-something

Help text written by -h, as well as man pages and the HTML help now refer to programs as gmx-something instead of the old names that are now only symbolic links. Cross-references in the HTML pages should be there again.

Related to #685.

Change-Id: I41ce99cc224aa2c13f94d61e6265115a8c991daba

---

11/19/2015 9/28
Uniform behavior for 'make man' and 'make html'

Both man and HTML pages are now generated and installed using similar installation rules. GMX_BUILD_HELP controls automatic generation of both. They are also put into the source distribution using the same mechanism for both.

Part of #685 and #1242.

Change-Id: ld61e75623861b46f765da49c4b34047a8b327083

Revision 493a9c5e - 12/02/2013 03:56 PM - Teemu Murtola
Remove remaining uses of programs.txt

Other uses have been replaced with direct generation from the wrapper binary. Since the manual no longer has the program man pages, there is no particular need to list the programs either, in particular since this is potentially more maintenance. A reference to the online help and/or 'gmx help' should be sufficient.

Related to #685.

Change-Id: le865e955416cdd28f654c128f88994bac7550ac6

Revision 04cc4266 - 01/28/2014 02:05 PM - Teemu Murtola
Shell completion export from the wrapper binary

Bash completions now complete the arguments to the wrapper binary itself, as well as the subcommand names. After a subcommand has been specified, use the existing completion logic. Completions now work also for arbitrarily suffixed binaries (which the old system didn't work at all). No completion is generated for symlinks, but should be straightforward to do if really required; approach would be the same as for GMX_BUILD_MDRUN_ONLY.

Other completions are not working for the wrapper binary, and as far as I can tell, require a completely different approach. Removed the non-functional code.

Related to #685 and #1410.

Change-Id: i55b13a65c176dab6e2f4f41bf6e829112c99e6b3
Print common options to 'gmx -h'

The help shown for the wrapper binary now includes the command line options it accepts. Remaining issues that will be fixed separately:
- Unlike other help output, this output is actually influenced by what the user specifies on the command line.
- The information is not so easy to find, because of the long list of subcommands that comes after it.
- The information should go to some man page as well.

Related to #685.

Change-Id: lac72dd21d4a733016b331549e139d4184bd3e283

Move command listing to 'gmx help commands'

Instead of printing the list of commands on 'gmx' or 'gmx help', print it only when explicitly requested with 'gmx help commands'. This makes it easier to see the help about the common command line options that 'gmx' accepts. The list of available help topics lists the 'commands' topic, so it should be easy to find.

Related to #685.

Change-Id: I00e99ceae15ebd47d2ac129aee8e6e407f6c388a

History

#1 - 01/26/2011 05:06 PM - Justin Lemkul

Christoph Junghans wrote:

```
ls -1 $GMXBIN/* | wc -l -> 142 (for single and double)
```

Already at this point it gets a little unmanageable to remember the names of all binaries. I think, it would be nice to have something like a global gmx binary, which has the program trunc as the first option (e.g. "gmx dist", "gmx rdf").

I don't understand how this helps. Doesn't this setup still imply that you have to know the name of the tool you want to use, anyway? Instead of "g dist" the user has to remember "gmx dist." This sounds a lot like ptraj within Amber (global analysis tool requiring manual syntax), which I never liked, and is one of the reasons our lab switched to Gromacs in the first place.

I also wonder how documentation and help info would be printed. "gmx -h" would be too much to comprehend, and "gmx dist -h" still implies that you know what tool you're looking for in the first place.

Just my $0.02, but maybe I'm missing the bigger picture :)

11/19/2015
This is closely related to issue #672, although that issue only considers analysis tools that process trajectories. There, the main reason for the change would be technical; the issues that Justin points out are still valid.

Justin Lemkul wrote:

I don't understand how this helps. Doesn't this setup still imply that you have to know the name of the tool you want to use, anyway? Instead of "g_dist" the user has to remember "gmx dist."

I just think >70 binaries are too much, but this is my opinion.
I have been asked very often if there is gmx program doing a certain task, because people didn't know all 70 programs. And I would like to make it simpler to find the right program, also because not all of them are named g_* (even if fedora does that)

I also wonder how documentation and help info would be printed. "gmx -h" would be too much to comprehend, and "gmx dist -h" still implies that you know what tool you're looking for in the first place.

I think git is a good example (they had 144 binaries before switching to the wrapper).
gmx --help or gmx help should print a short description of the important programs (rdf,...). gmx help --all should print a short description of all programs (like man 7 gromacs) and gmx XXX --help should print the normal help.

Justin Lemkul wrote:

I don't understand how this helps. Doesn't this setup still imply that you have to know the name of the tool you want to use, anyway? Instead of "g_dist" the user has to remember "gmx dist."

I just think >70 binaries are too much, but this is my opinion.
I have been asked very often if there is gmx program doing a certain task, because people didn't know all 70 programs. And I would like to make it simpler to find the right program, also because not all of them are named g_* (even if fedora does that)

I just don't see how having one wrapper that hides all the tools into one umbrella "program" makes the situation any better. You still have to go through the manual to find the proper keyword (as you do know for whatever g_* you want). It seems to me that most advanced users could probably adapt, having previous knowledge of what Gromacs can do, but new users will be overwhelmed by some black box tool that can do lots of stuff, but
that they have to figure out how to use. Optionally making symlinks with the old names is all well and good, but someone new to Gromacs may not understand why they're doing that. We have lots of instances of people ignoring existing documentation...

I also wonder how documentation and help info would be printed. "gmx -h" would be too much to comprehend, and "gmx dist -h" still implies that you know what tool you're looking for in the first place.

I think git is a good example (they had 144 binaries before switching to the wrapper).

```
gmx --help or gmx help should print a short description of the important programs (rdf,...).
```

Importance is subjective. Who is to decide what is important? We don't want to try to out-think the end user, since that will certainly cause quite a lot of frustration.

```
gmx help --all should print a short description of all programs (like man 7 gromacs) and gmx XXX --help should print the normal help.
```

Sure, but if this functionality already exists (i.e., man 7 gromacs), then what is the value of the change besides compactness of the binaries?

I think all of this might not pass Berk's "single program, single function" mantra :) If any major changes are proposed, they should probably be aired over gmx-developers to get the pulse of the community, not just the few people watching this issue.

---

**#5 - 01/26/2011 09:33 PM - Erik Lindahl**

I think there are some programs we could try to merge, and long-term we will not need separate single/double precision for most things, but apart from that I too am a big proponent of the Unix idea with simple tools that only do one thing, but do it well. Still, I think they could be merged.

However, related to this we also have the issue that you frequently want to apply multiple operations during analysis, so ultimately we'd like some way of specifying "for each frame, first do a fitting, then rotate by X degrees, and finally perform analysis Y". Teemu's recent work on the analysis library will simplify this a lot, but we're not quite there yet. So, I don't want to discard this idea, but it might be better to revisit it later in the 5.0 development process when the underlying tools have converged a bit more?

---

**#6 - 01/26/2011 10:39 PM - Mark Abraham**
I don't see there's much to gain here. I also like the Unix "one tool, one function" approach. It was a big drawcard for me when choosing software. Most serious work will end up with a script anyway (even if only to document the workflow), and I'd much rather learn one scripting language (e.g. bash) that I can use with all my tools than have to learn

- dedicated FORTRAN-friendly scripting syntaxes as with CHARMM, or
- cryptic sander and ptraj horrors in AMBER, or
- even nice GROMACS python/perl/whatever extensions.

git uses the sub-tool idea - at one point in its history, things like git add called a git-add script, but now the latter symlinks the former, IIRC. I don't see that it gains anything in usability. git help add and git add --help both get the help for "add". git help gets some generic help. It took me months to learn that - after failing to find routinely-installed man pages, I used web-based ones. Even if one can find the tool names, one is still reduced to using Google to seek tutorial material to help find out "how do I do the operation I have in mind".

While statically-linked installed binary size can be smaller with a combination tool, that's not a big effect.

Reducing the linking time is useful for developers, but again not big.

We can make our list of tool functionality more accessible. I only learned of man 7 gromacs in this thread - and that only works if you've bothered to set up MANPATH properly. I no longer set that up because g_tool -h serves that need. Manual section 7.4 lists this information also, but it's hardly as prominent as it could be.

There's also a bit of overlap with issue #219.

#7 - 01/27/2011 10:49 AM - Christoph Junghans

Mark Abraham wrote:

We can make our list of tool functionality more accessible. I only learned of man 7 gromacs in this thread - and that only works if you've bothered to set up MANPATH properly. I no longer set that up because g_tool -h serves that need. Manual section 7.4 lists this information also, but it's hardly as prominent as it could be.

I found man 7 gromacs very useful, but one can also look up the list in the manual, that is true. Actually the MANPATH should be set by GMXRC, is this a bugs?

From the above discussion I would conclude that a wrapper binary is not really helpful and would break the one tool one task standard. As this was just an idea, we can nearly close this issue.

Can somebody add Jussi Lehtola (I am not powerful enough or redmine is too confusing for me ;-) ) to discuss the packaging aspect (140 binaries in /usr/bin) as well?
#8 - 01/27/2011 04:30 PM - Rossen Apostolov

Christoph Junghans wrote:

Can somebody add Jussi Lehtola (I am not powerful enough or redmine is too confusing for me ;-) ) to discuss the packaging aspect (140 binaries in /usr/bin) as well?

I added him to project “Gromacs”, should I add him to one of the others too?

#9 - 01/27/2011 05:30 PM - Teemu Murtola

- Assignee deleted (David van der Spoel)

Let me comment a bit more on the work I've been doing (what the complete analysis tool subproject is about), because it does touch these issues, and I think it would be good to clear some things out, so those can be discussed, e.g., here.

The basic idea there is to have functionality common to all trajectory analysis tools in a separate framework, and have the tools implemented only as "modules" that provide the logic for doing the actual analysis, and also provide options that the user can set to customize the actual analysis. The framework then provides common functionality for running the analysis: loading the topology and the trajectory, evaluating selections, etc. This framework can be used to implement analysis tools more or less identical to those currently out there, but it will also make it possible to, e.g., implement custom tools by instantiating an existing module and writing custom code that does something with the results. So it will be natural to have both the modules and the framework in a library.

In this context, what's the added value of writing tens of main() functions that each have perhaps three lines, and only differ in one string, compared to a single binary that accomplishes the same task and can be invoked identically? Also, how does providing extra interfaces, e.g., through Python bindings, harm those who do not wish to use them if existing functionality is preserved?

I think that the idea of "one program, one function" is a good one, and it's a big issue for maintainability as well, but I can't see how the good principle would be broken by having the same binary implement several functions as non-interacting modules (in particular, if only one function can be invoked at a time).

For the original concern, i.e., difficulty to find a tool for a certain task, I have to agree that the current situation is not very satisfactory, and it should be improved when tools are converted to the new framework (issue #665). For example, even if one knows that one wants to calculate distances, it's very difficult to guess whether g_bond, g_dist, g_mindist, or perhaps some other tool is suitable for that particular purpose. For different kinds of angles, there are probably even more tools than for distances...

#10 - 01/27/2011 05:55 PM - Jussi Lehtola

Christoph Junghans wrote:

Can somebody add Jussi Lehtola (I am not powerful enough or redmine is too confusing for me ;-) ) to discuss the packaging aspect (140

11/19/2015 15:28
binaries in /usr/bin) as well?

"One Ring to rule them all, One Ring to find them,
One Ring to bring them all and in the darkness bind them"

Oh yes, please! From a packaging point of view, the amount of binaries in GROMACS is simply staggering.
I have a pretty big installation of Fedora, but Gromacs still contributes 3.6% of the 5000+ binaries I have in /usr/bin.
This can be thought to add unnecessary overhead to e.g. anyone using autocompletion in a shell environment.

From the above discussion I would conclude that a wrapper binary is not really helpful and would break the one tool one task standard. As this was just an idea, we can nearly close this issue.

I don't agree on this. The simplicity of the code and of the use is still there, even though you replace e.g.
$ g_energy -f ener.edr
with
$ gromacs energy -f ener.edr

IMHO it's easier for the users that everything has a consistent interface.

Besides, since AFAIK all GROMACS binaries are currently essentially wrappers, e.g. here's g_rama.c:

```c
int main(int argc, char *argv[]) {
    gmx_rama(argc,argv);
    return 0;
}
```

Implementing a consistent interface is EASIER for code maintainers as well, since there's a lot less stuff to handle! E.g. in crappy C:

```c
int main(int argc, char *argv[]) {

    if(strcmp(argv[1],"rama")==0) {
        gmx_rama(argc-1,argv+1);
        return 0;
    }

    if(strcmp(argv[1],"energy")==0) {
        gmx_energy(argc-1,argv+1);
        return 0;
    }

    printf("Invalid command. See man 7 gromacs to see available commands.\n");
    return 0;
}
```

As another plus, GROMACS will consume less disk space with a single wrapper. A lot less if static libraries are used...
Summarizing:

It is suggested that we re-wrap the tools so that $g_{\text{ energy}}$ becomes something like $g_{\text{romacs energy}}$. This keeps the UNIX-style “one tool, one function” but changes the invocation to help keep some machinery cleaner and more maintainable. mdrun would stay separate.

Gains

- fewer tools to install (smaller binary footprint, particularly with static linking, reduced chance of future binary name space clashes)
- fewer tools to build (faster linking)
- fewer source files that exist only to provide a main() function
- simpler CMake build scripts (maybe?)
- packaging looks and is cleaner (encourages people to actually package GROMACS into major software distributions)
- consistent “look and feel” with modern Unix SCM tools (e.g. git, hg, svn. Any other examples?)
- reduces existing name space inconsistencies ($gm_{*}$ tools vs $g_{*}$ tools, and the latter set is larger only on Fedora)

Losses

- existing scripts break (unless we provide compatibility scripts during a changeover period)
- auto-completion support needs upgrade (else $g_{\text{romacs ener}[TAB]}$ will fail to auto-complete to $g_{\text{romacs energy}}$ in the way that the git tools do)
- people have to make changes in the way they think
- new work making $g_{\text{romacs help}}$ do something useful
- man page usage changes (people have to remember to use $g_{\text{romacs-energy}}$, or $g_{\text{romacs help energy}}$ or $g_{\text{romacs energy -h}}$)
- documentation reference to tool names need updating

Opportunities

- possible new work making $g_{\text{romacs energy}}$ lead to useful suggestions about what the user really wanted to enter (like git does; this is not hard if we’re prepared to use a regular expression library, which would be useful in simplifying other parts of the GROMACS code too)
- the changeover period also allows us to consider changing names for some tools (e.g. grompp becomes $g_{\text{romacs compile}}$, pdb2gmx and $g_{\text{x2top}}$ can be renamed something that properly reflects their function as topology builders (not file conversion utilities), genbox becomes $g_{\text{romacs solvate}}$

New commentary and suggestions on the above are welcome. I'll wait a day or two and cast it out on gmx-developers for wider comment.

Is gromacs or gmx a better name for the umbrella tool? The former is more unique (the latter is a German ISP also, I think). Auto-completion of the former will only happen with grom[TAB]. “gm” is the name of an executable in the GraphicsMagick package, which might hamper auto-completion there.

In my view, the losses are not large and are "one time" losses, whereas the gains will continue to accrue over time. So I'm in favour of such a change.
Mark Abraham wrote:

Summarizing:

It is suggested that we re-wrap the tools so that `g_energy -f ener` becomes something like `$ gromacs energy -f ener`. This keeps the UNIX-style "one tool, one function" but changes the invocation to help keep some machinery cleaner and more maintainable. `mdrun` would stay separate.

(clip)

In my view, the losses are not large and are "one time" losses, whereas the gains will continue to accrue over time. So I'm in favour of such a change.

A good job on the summary!

IMHO `mdrun` could also be included in the wrapper, just for consistency. Think about

```bash
$ gromacs compile
$ gromacs run
instead of
$ gromacs compile
$ mdrun
```

`mdrun` is such a general name that I wouldn't be at all surprised if there were N+1 other projects that used the same name.. That's why I renamed it in Fedora to `g_mdrun`.

Some problems might arise with MPI, though, since only `mdrun` is parallelized, but I don't think that this is a big issue as such. I would recommend using conditional compilation, so that the MPI enabled wrapper binary can only be run with the "run" argument, otherwise a help message is printed out "(argv 1) has not been parallelized. Please use the serial version."

If wrapping `mdrun`, too, is out of the question, I still would like it to be renamed as well. Something on the lines of `rungromacs` or `gromacs-run`, maybe?
Erik Lindahl wrote:

I think there are some programs we could try to merge, and long-term we will not need separate single/double precision for most things, but apart from that I too am a big proponent of the Unix idea with simple tools that only do one thing, but do it well. Still, I think they could be merged.

I think this also could be looked at at the same time, and it shouldn't be very hard to implement either. Since everything AFAIK already supports compile-time definitions of datatypes, what would be needed is to change the function names depending on the used precision, i.e.

```c
#ifdef DOUBLE_PRECISION
    int gmx_rdf_d(int argc,char *argv[]) {
#else
    int gmx_rdf(int argc,char *argv[]) {
#endif
    
The number of source files doesn't change. The make system can be made to compile all source code files in both single and double precision, resulting in e.g. gmx_rdf.o and gmx_rdf_d.o, and the used precision can be made a runtime option:

```c
if(usedouble)
    gmx_rdf_d(argc-1,argv+1);
else
    gmx_rdf(argc-1,argv+1);
```

Double precision is sometimes quite handy, for instance in minimizing pathologic starting configurations, even though in normal cases (runs with thermostats) it's just a waste of cpu time.

Mark Abraham wrote:

Summarizing:

It is suggested that we re-wrap the tools so that $ g\_energy -f ener$ becomes something like $ gromacs energy -f ener$. This keeps the UNIX-style "one tool, one function" but changes the invocation to help keep some machinery cleaner and more maintainable. mdrun would stay separate.
In my view, the losses are not large and are "one time" losses, whereas the gains will continue to accrue over time. So I'm in favour of such a change.

A good job on the summary!

IMHO mdrun could also be included in the wrapper, just for consistency. Think about

$ gromacs compile
$ gromacs run
instead of
$ gromacs compile
$ mdrun

mdrun is such a general name that I wouldn't be at all surprised if there were N+1 other projects that used the same name.. That's why I renamed it in Fedora to g_mdrun.

In fact there is some RAID tool that uses mdrun, so perhaps some change is in order.

Some problems might arise with MPI, though, since only mdrun is parallelized, but I don't think that this is a big issue as such. I would recommend using conditional compilation, so that the MPI enabled wrapper binary can only be run with the "run" argument, otherwise a help message is printed out "(argv^2) has not been parallelized. Please use the serial version."

There is at least one other tool parallelised (g_pme_error) and doubtless more will come with time, so this issue needs a general solution. How about $ mpirun gromacs run -mpi? There are probably a few options that should be parsed by the wrapper before invoking the tool (e.g. -mpi, -double) but I don't think $ mpirun gromacs -mpi run is worth the cost in clarity.

One thing to keep in mind is that there exist platforms that lack both dynamic linking and virtual memory (e.g. BlueGene/L; BlueGene/P lacks the latter). Now static executable size can become a serious issue. If there's 500MB available per core for program, data, I/O buffers and MPI buffers, one doesn't want useless analysis tools consuming memory during a simulation. Also, there may be MPI setups that aren't friendly to doing the little bit of processing required before MPI_Init to implement $mpirun gromacs run -mpi.

For these kind of reasons, I think we need to be conservative with mdrun, rather than elegant, and so keep our present approach (perhaps renamed) with gromacs_run and gromacs_run_mpi

If wrapping mdrun, too, is out of the question, I still would like it to be renamed as well. Something on the lines of rungromacs or gromacs-run, maybe?
Jussi Lehtola wrote:

I think this also could be looked at at the same time, and it shouldn't be very hard to implement either. Since everything AFAIK already supports compile-time definitions of datatypes, what would be needed is to change the function names depending on the used precision, i.e.

<snip>

So now gromacs compile -double could invoke double-precision grompp. As above with the MPI issues, I think (possibly renamed) mdrun should remain aloof.

Justin Lemkul wrote:

Just a thought, but it seems to me we're losing sight of one important aspect - the end user. What convenience does all of this provide? The initial feature request was that there were too many program names to remember. The solution that's emerged is to create a single "gromacs" wrapper, which then calls a whole host of newly-named things that people will have to figure out. Aside from a convenience for development and package maintenance, I fail to see how much of this benefits the end user. Anyone who's new to Gromacs will certainly learn how to use the new tool(s) just fine, but for those thousands of people out there who already do, all of this amounts to "Now I have to type 'gromacs' in front of everything, AND learn new names."

I think any transition needs to be very gradual. Merging analysis tools seems reasonable enough, but why go all the way to merge EVERY tool into one master wrapper? What's wrong with having 10 programs or so? AMBER has about 70 (and they're named ridiculously, too), but at least in Gromacs there was a pattern.

Some tools could easily be merged:

1. editconf + genbox + genion -> gmxbox
2. stripping pdb2gmx into two programs, like gmxclean to rename and prepare coordinate files, then gmxtop to merge topology functions of g_x2top and pdb2gmx (I don't know if this is "easy" or not, but it seems logical enough)

And probably a few more.

Jussi Lehtola wrote:

Mark Abraham wrote:

Some problems might arise with MPI, though, since only mdrun is parallelized, but I don't think that this is a big issue as such. I would recommend using conditional compilation, so that the MPI enabled wrapper binary can only be run with the "run" argument, otherwise a help message is printed out "(argv1) has not been parallelized. Please use the serial version."
There is at least one other tool parallelised (g_pme_error) and doubtless more will come with time, so this issue needs a general solution. How about $mpirun gromacs run -mpi? There are probably a few options that should be parsed by the wrapper before invoking the tool (e.g. -mpi, -double) but I don't think $mpirun gromacs -mpi run is worth the cost in clarity.

First of all, I just realized that redmine is very annoying compared to bugzilla, since I just lost my comments to this reply when I tried to answer a second one in the same message.

Anyway, this is not what I meant at all. I don't think that using MPI should be a runtime option, especially since
1) There are a big bunch of MPI libraries out there, and some systems might have many of them installed at the same time
2) There are a few implementations (cough mpich2 cough) that produce binaries that can't be run without $mpirun in front

In my opinion when the wrapper is compiled with MPI support enabled, it should only link against those functions that have been properly parallelized (currently mdrun and g_tune_pme). Trying to use other functions would produce the error I suggested above, prompting the user to use the serial version.

One thing to keep in mind is that there exist platforms that lack both dynamic linking and virtual memory (e.g. BlueGene/L; BlueGene/P lacks the latter). Now static executable size can become a serious issue. If there's 500MB available per core for program, data, I/O buffers and MPI buffers, one doesn't want useless analysis tools consuming memory during a simulation. Also, there may be MPI setups that aren't friendly to doing the little bit of processing required before MPI_Init to implement $mpirun gromacs run -mpi.

But even now mdrun itself doesn't run MPI_Init...?

Currently on my Fedora the g_* binaries (both in single and double precision) take all in all 3.0 MB. The dynamic libraries take 12 MB. So this would add up to something like 15 MB.

If one puts in a build time option of enabling only one precision, the size would drop down to roughly a half. And if only the MPI enabled stuff is compiled, the size shouldn't change very much from status quo.

So I really don't think this is an issue.

#18 - 01/28/2011 11:52 PM - Jussi Lehtola
Justin Lemkul wrote:

Just a thought, but it seems to me we're losing sight of one important aspect - the end user. What convenience does all of this provide? The
initial feature request was that there were too many program names to remember. The solution that's emerged is to create a single "gromacs" wrapper, which then calls a whole host of newly-named things that people will have to figure out. Aside from a convenience for development and package maintenance, I fail to see how much of this benefits the end user. Anyone who's new to Gromacs will certainly learn how to use the new tool(s) just fine, but for those thousands of people out there who already do, all of this amounts to "Now I have to type 'gromacs' in front of everything, AND learn new names."

Yes, people will complain. This is a given.

No matter how crappy a system might be, there will always be people complaining "Why did you have to change this, it used to work out so well?". That's why some people still code in Fortran 77.

The change just needs to be well reasoned, as Mark elegantly did in comment #11. Gromacs would become easier both for developers and for users, at least for new ones.

I think any transition needs to be very gradual. Merging analysis tools seems reasonable enough, but why go all the way to merge EVERY tool into one master wrapper? What's wrong with having 10 programs or so? AMBER has about 70 (and they're named ridiculously, too), but at least in Gromacs there was a pattern.

I count currently 91 binaries, every one of which needs a double precision version.
And then you can count the MPI binaries, of which there are currently two (x 2 to support both precisions).

The point is that if you change something, it's certain that you will encounter some resistance. Users will anyway have to port their scripts.

But when you make the change it's better to renormalize the convention all at once, so that new functions can easily be added later on. Taking a middle road just won't give you anything extra...

#19 - 01/29/2011 01:35 AM - Mark Abraham
Justin Lemkul wrote:

Just a thought, but it seems to me we're losing sight of one important aspect - the end user. What convenience does all of this provide?

There's not very much direct benefit. Making developers' time more fruitful has obvious end-user benefits. Making it easier to have prepackaged GROMACS available through package management systems means they have fewer installation issues. Tool names that properly reflect their functions helps users understand what they do (e.g. avoiding the recurring gmx-users posts about converting their .pdb to a .gro with pdb2gmx; more new users will understand the idea behind gromacs compile than grompp because nobody much uses a stand-alone pre-processor any more, even if
they recognize that 'pp' is a suffix).

The initial feature request was that there were too many program names to remember. The solution that's emerged is to create a single ‘gromacs’ wrapper, which then calls a whole host of newly-named things that people will have to figure out. Aside from a convenience for development and package maintenance, I fail to see how much of this benefits the end user. Anyone who's new to Gromacs will certainly learn how to use the new tool(s) just fine, but for those thousands of people out there who already do, all of this amounts to "Now I have to type 'gromacs' in front of everything, AND learn new names."

I think any transition needs to be very gradual. Merging analysis tools seems reasonable enough, but why go all the way to merge EVERY tool into one master wrapper? What's wrong with having 10 programs or so? AMBER has about 70 (and they're named ridiculously, too), but at least in Gromacs there was a pattern.

OK, someone needs to identify more stand-alone candidates.

Some tools could easily be merged:

1. editconf + genbox + genion -> gmxbox

I disagree with this one. I looked at it yesterday. There's always the need to define the box before filling it with solvent and then maybe removing solvent in favour of other stuff. A single tool hides the order of operations and I think will lead to people misunderstanding the process. The umbrella tool would also have to be engineered to cope with all possible combinations. I think these are single tools that do single functions well and should be left alone.

2. stripping pdb2gmx into two programs, like gmxclean to rename and prepare coordinate files, then gmxtop to merge topology functions of g_x2top and pdb2gmx (I don't know if this is "easy" or not, but it seems logical enough)

Once the structure is clean, then topology generation is a trivial string-matching exercise. I don't see the benefit of a pdb2gmx split, compared with just a rename.

If one were going to write pdb2gmx from scratch, then these days one would do it with perl. There must be thousands of lines of string-handling C code for pdb2gmx and grompp. However, it isn't broken so it shouldn't be fixed.
Some problems might arise with MPI, though, since only mdrun is parallelized, but I don't think that this is a big issue as such. I would recommend using conditional compilation, so that the MPI enabled wrapper binary can only be run with the "run" argument, otherwise a help message is printed out "(argv[1]) has not been parallelized. Please use the serial version."

There is at least one other tool parallelised (g_pme_error) and doubtless more will come with time, so this issue needs a general solution. How about $ mpirun gromacs run -mpi? There are probably a few options that should be parsed by the wrapper before invoking the tool (e.g. -mpi, -double) but I don't think $ mpirun gromacs -mpi run is worth the cost in clarity.

Anyway, this is not what I meant at all. I don't think that using MPI should be a runtime option, especially since

1) There are a big bunch of MPI libraries out there, and some systems might have many of them installed at the same time

MPI is always a run-time option. There is no conceptual difference between calling an MPI-linked binary on the command line with $ mdrun_mpi, and calling an MPI-linked library function from the wrapper tool with $ gromacs run -mpi. GROMACS was linked with one of the MPI libraries and that one will be used at run time, regardless of how the command line invocation looks.

2) There are a few implementations (cough mpich2 cough) that produce binaries that can't be run without $ mpirun in front

In my opinion when the wrapper is compiled with MPI support enabled, it should only link against those functions that have been properly parallelized (currently mdrun and g_tune_pme). Trying to use other functions would produce the error I suggested above, prompting the user to use the serial version.

So now we have to have gromacs and gromacs_mpi just to choose whether to run a couple of tools in serial or parallel? What's the advantage? Why can't your "use_double" example apply to MPI too?

One thing to keep in mind is that there exist platforms that lack both dynamic linking and virtual memory (e.g. BlueGene/L; BlueGene/P lacks the latter). Now static executable size can become a serious issue. If there's 500MB available per core for program, data, I/O buffers and MPI buffers, one doesn't want useless analysis tools consuming memory during a simulation. Also, there may be MPI setups that aren't friendly to doing the little bit of processing required before MPI_Init to implement $mpirun gromacs run -mpi.

But even now mdrun itself doesn't run MPI_Init...?

True, MPI_Init() is called in a function called from main(). So this is a false worry.

Currently on my Fedora the g_* binaries (both in single and double precision) take all in all 3.0 MB. The dynamic libraries take 12 MB. So this would add up to something like 15 MB.

Yes, but you're using dynamic linking. There's also system, C, math, MPI and FFT libraries, and maybe BLAS, LAPACK, GSL, XML, VMD and maybe other "one use wonders" in the future. Those add up.

If one puts in a build time option of enabling only one precision, the size would drop down to roughly a half. And if only the MPI enabled stuff is compiled, the size shouldn't change very much from status quo.

So I really don't think this is an issue.

This will vary a bit from system to system, of course. Statically-linked single-precision BlueGene/L 4.5.3 mdrun is 20MB just to start from. I'd hate to
see that grow to 50MB, but it probably wouldn't. Even on BlueGene/P with dynamic linking, the binary will be smaller, but at the very least, the code only used for analysis routines must go in a separate dynamic library.

#21 - 02/01/2011 12:52 AM - Mark Abraham

#22 - 08/29/2011 05:38 PM - Rossen Apostolov
- Target version deleted (git master)

#23 - 01/06/2012 12:45 AM - Christoph Junghans
There are another 2 new tools in the queue. Should we rethink this again?

#24 - 01/07/2012 04:34 AM - Mark Abraham
I thought this issue was on hold awaiting 5.0 so that we avoid duplicating work to make the magic work in C and then again in C++

#25 - 11/18/2012 08:29 PM - Teemu Murtola
- Target version set to 5.0

Some decision or another should be reached for 5.0, so set the target version appropriately. https://gerrit.gromacs.org/#/c/1689/, https://gerrit.gromacs.org/#/c/1709/ and https://gerrit.gromacs.org/#/c/1710/ contain one proposal for how to do this. Contains most of the src/tools/ subdirectory for now, rest should be relatively easy to add, except possibly those that require parallel initialization. Some discussion may be required on how much effort we want to put into producing "small" binaries for, e.g., mdrun (and how to best do this). But for a majority of systems, a single wrapper binary (for now, with separate versions for double and/or MPI) is probably a good enough solution.

#26 - 09/02/2013 08:28 PM - Teemu Murtola
- Status changed from New to Blocked, need info

The current situation is that everything except mdrun, ngmx and g_xrama are actually built into a single binary, gmx, and invoked as gmx angle. Symlinks to the gmx binary also work to invoke them using the old names. mdrun is also included by https://gerrit.gromacs.org/#/c/2570/. Working on making HTML help and man page generation work reasonably (some draft in https://gerrit.gromacs.org/#/c/2585/). Including also ngmx and g_xrama into gmx would simplify this as well. Shell completion requires more work, as it needs some extra logic to select the correct set of completions based on the second argument in gmx angle .... To make things simpler, I would propose:

- Only support shell completions with the new syntax (gmx angle ...), and not if the binary is invoked through a symlink.
- Drop generating the program options for the LaTeX manual. Instead, just reference the HTML manual. This is along the lines Mark proposed in #1242. This shouldn't be too hard to get working, but until #1177 is done, I don't want to start juggling with the different repositories.

Comments?

#27 - 09/02/2013 09:09 PM - Susi Lehtola
Looks pretty good, but IMHO it'd be nicer to change the name of mdrun as well to something like gmxrun. mdrun is just too general a name...

#28 - 10/01/2013 04:29 PM - David van der Spoel
In fact everything now is inside the gmx binary already! I much prefer this from the perspective of a "seasoned" user.

What about the suggestion that passed by some time ago to have single and double in one binary? It seemed appealing for a moment, but
maybe gmx and gmx_d is easier.

#29 - 10/01/2013 07:40 PM - Teemu Murtola

David van der Spoel wrote:

In fact everything now is inside the gmx binary already! I much prefer this from the perspective of a "seasoned" user.

Yes, that part is now done. But a lot of clean-up still remains:

- It should not be necessary to create the symlinks into the build directory. This is required for regression tests and the manual build, at least. Changing the manual build probably does not make sense before #1177 is done.
- Documentation would need to be updated to not refer to the deprecated symlinks. https://gerrit.gromacs.org/#/c/2590/ and changes leading up to it do a large part of it, but again #1177 would be nice to not break the manual build.
- Shell completions need to be updated. https://gerrit.gromacs.org/#/c/2590/ does so for bash, but people who know other shells need to pitch in.
- Make the wrapper binary aware of the information currently in programs.txt to make gmx help more useful. Remove the long list printed by gmx help or make it shorter.

In particular, help would be appreciated with:

- Deciding whether the LaTeX output is worth keeping (https://gerrit.gromacs.org/#/c/2612), and/or getting #1177 done.
- Shell completions for other shells than bash.
- Review for that batch of changes.

What about the suggestion that passed by some time ago to have single and double in one binary? It seemed appealing for a moment, but maybe gmx and gmx_d is easier.

That will be an enormous amount of effort (see #951; also #1165 is related). The simplest alternative would probably be separate libraries for different precisions, and selecting one dynamically, but this gets ugly as some processing is needed in the binary before the correct library can be selected, and that cannot use any functions from the library. At the same time, stuff like command-line processing and common initialization needs to be in the library to be usable from user tools.

#30 - 12/27/2013 06:19 AM - Teemu Murtola

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

#31 - 01/25/2014 06:47 AM - Gerrit Code Review Bot

Gerrit received a related patchset ‘1’ for Issue #685.
Uploader: Teemu Murtola (teemu.murtola@gmail.com)
Change-Id: Iac72dd21d4a733016b331549e139d4184bd3e283
Gerrit URL: https://gerrit.gromacs.org/3023

#32 - 01/25/2014 07:18 PM - Gerrit Code Review Bot
Gerrit received a related patchset '1' for Issue #685.
Uploader: Teemu Murtola (teemu.murtola@gmail.com)
Change-Id: I00e9ceae15ebd47d2ac129aae8e6e407f86c388a
Gerrit URL: https://gerrit.gromacs.org/3025

#33 - 02/14/2014 08:05 PM - Teemu Murtola
- Subject changed from wrapper binary to Wrapper binary
- Category changed from analysis tools to build system
- Status changed from Blocked, need info to Resolved
- Assignee set to Teemu Murtola

Done to a sufficient degree to mark Resolved. There may be some outdated documentation left somewhere, and some feedback has been given about the man pages (currently in #969; will create a separate issue about those at some point). Renaming the existing commands (e.g., mdrun) can be handled separately.

#34 - 02/24/2014 11:00 AM - Rossen Apostolov
- Status changed from Resolved to Closed