Если вы видите что-то необычное, просто сообщите мне.

Contributing to GHC 1: Preparation

Welcome to our series on Contributing to GHC! The Glasgow Haskell Compiler is perhaps the biggest and most important open source element of the Haskell ecosystem. Without GHC and the hard work that goes into it from many volunteers, Haskell would not be the language it is today. So in this series we'll be exploring the process of building and contributing to GHC. This first part will focus on how to set up our environment for GHC development. If you've already done this, feel free to move onto part 2 of this series, where we'll start some basic hacking!

While many of you are working on Linux or Mac setups, this article uses examples from setting up on Windows. Getting GHC to build on Windows simply involves more hurdles, and we want to show that we can contribute even in the most adverse circumstances. There is a section further down that goes over the most important parts of building for Mac and Linux. You'll generally have to install some of the same packages, but use the native tools on those systems, which are much simpler than on Windows. I'll be following this guide by Simon Peyton Jones, sharing my own complications.

Now, you need to walk before you can run. If you've never used Haskell before, you have to try it out first to understand GHC! Download our Beginner's Checklist to get started! You can also read our Liftoff Series to learn more about the language basics. If you're more interested in applying Haskell than working on GHC, you should read our Haskell Web Series and download our Production Checklist!

MSYS

The main complication with Windows is that the build tools for GHC are made for Unix-like environments. These tools include programs like autoconf and make. And they don't work in the

normal Windows terminal environment. This means we need some way of emulating a Unix terminal environment in Windows. There are a couple different options for this. One is Cygwin, but the more supported option for GHC is MSYS 2. So my first step was to install this program. This terminal will apply the "Minimalist GNU for Windows" libraries, abbreviated as "MinGW".

Installing this worked fine the first time. However, there did come a couple points where I decided to nuke everything and start from scratch. Re-installing did bring about one problem I'll share. In a couple circumstances where I decided to start over, I would run the installer, only to find an error stating bash.exe: permission denied. This occurred because the old version of this program was still running on a process. You can delete the process or else just restart your machine to get around this.

Once MSys is working, you'll want to set up your terminal to use MinGW programs by default. To do this, you'll want to set the path to put the mingw directory first:

```
echo "export PATH=/mingw<bitness>/bin:\$PATH" >> ~/.bash_profile
```

Use either 32 or 64 for `<bitness>` depending on your system. Also don't forget the quotation marks around the command itself!

PACKAGE PREPARATION

Our next step will be to get all the necessary packages for GHC. MSys 2 uses an older package manager called pacman, which operates kind've like apt-get. First you'll need to update your package repository with this command:

```
pacman -Syuu
```

As per the instructions in SPJ's description, you may need to run this a couple times if a connection times out. This happened to me once. Now that pacman is working, you'll need to install a host of programs and libraries that will assist in building GHC:

```
pacman -S --needed git tar bsdtar binutils autoconf make xz \
    curl libtool automake python python2 p7zip patch ca-certificates \
    mingw-w64-$(uname -m)-gcc mingw-w64-$(uname -m)-python3-sphinx \
```

```
    mingw-w64-$(uname -m)-tools-git
```

This command typically worked fine for me. The final items we'll need are alex and happy. These are Haskell programs for lexing and parsing. We'll want to install Cabal to do this. First let's set a couple variables for our system:

```
arch=x64_64 # could also be i386
bitness=64  # could also be 32
```

Now we'll get a pre-built GHC binary that we'll use to Bootstrap our own build later:

```
curl -L https://downloads.haskell.org/~ghc/8.2.2/ghc-8.2.2-${arch}-unknown-mingw32.tar.xz | tar -xJ -C /mingw${bitness} --strip-components=1
```

Now we'll use Cabal to get those packages. We'll place them (and Cabal) in /usr/local/bin, so we'll make sure that's created first:

```
mkdir -p /usr/local/bin
curl -L https://www.haskell.org/cabal/release/cabal-install-2.2.0.0/cabal-install-2.2.0.0-${arch}-unknown-mingw32.zip | bsdtar -xzf- -C /usr/local/bin
```

Now we'll update our Cabal repository and get both alex and happy:

```
cabal update
cabal install -j --prefix=/usr/local/bin alex happy
```

Once while running this command I found that happy failed to install due to an issue with the mtl library. I got errors of this sort when running the ghc-pkg check command:

```
Cannot find any of ["Control\\Monad\\Cont.hi", "Control\\Monad\Cont.p_hi", "Control\\Monad\\Cont.dyn_hi"]
Cannot find any of ["Control\\Monad\\Cont\\Class.hi", "Control\\Monad\Cont\\Class.p_hi", "Control\\Monad\\Cont\\Class.dyn_hi"]
```

I managed to fix this by doing a manual re-install of the mtl package:

```
cabal install -j --prefix=/usr/local/ mtl --reinstall
```

After this step, there were no errors on ghc-pkg check, and I was able to install happy without any problems.

```
cabal install -j --prefix=/usr/local/ happy
Resolving dependencies...
Configuring happy-1.19.9...
Building happy-1.19.9...
Installed happy-1.19.9
```

GETTING THE SOURCE AND BUILDING

Now our dependencies are all set up, so we can actually go get the source code now! The main workflow for contributing to GHC uses some other tools, but we can start from Github.

```
git clone --recursive git://git.haskell.org/ghc.git
```

Now, you should run the ./boot command from the ghc directory. This resulted in some particularly nasty problems for me thanks to my antivirus. It decided that perl was an existential threat to my system and threw it in the Virus Chest. You might see an error like this:

```
sh: /usr/bin/autoreconf: /usr/bin/perl: bad interpreter: No such file or directory
```

Even after copying another version of perl over to the directory, I saw errors like the following:

```
Could not locate Autom4te/ChannelDefs.pm in @INC (@INC contains /usr/share/autoconf C:/msys64/usr/lib .) at C:/msys64/usr/bin/autoreconf line 39
```

In reality, the @INC path should have a lot more entries than that! It took me a while (and a couple complete do-overs) to figure out that my antivirus was the problem here. Everything worked once I dug perl out of the Virus chest. Once boot runs, you’re almost set! You now need to configure everything:

```
./configure --enable-tarballs-autodownload
```

The extra option is only necessary on Windows. Finally you’ll use to make command to build everything. Expect this to take a while (12 hours and counting for me!). Once you’re familiar with

the codebase, there are a few different ways you can make things build faster. For instance, you can customize the build.mk file in a couple different ways. You can set BuildFlavor=devel2, or you can set stage=2. The latter will skip the first stage of the compiler.

You can also run make from the sub-directories rather than the main directory. This will only build the sub-component, rather than the full compiler. Finally, there's also a make fast command that will skip building a lot of the dependencies.

MAC AND LINUX

I won't go into depth on the instructions for Mac and Linux here, since I haven't gotten the chance to try them myself. But due to the developer-friendly nature of those systems, they're likely to have fewer hitches than Windows.

On Linux for instance, you can actually do most of the setup by using a Docker container. You'll download the source first, and then you can run this docker command:

```
>> sudo docker run --rm -i -t -v `pwd`:/home/ghc gregweber/ghc-haskell-dev /bin/bash
```

On Mac, you'll need to install some similar programs to windows, but there's no need to use a terminal emulator like MSys. If you have the basic developer tools and a working version of GHC and Cabal already, it might be as simple as:

```
>> brew install autoconf automake
>> cabal install alex happy haddock
>> sudo easy_install pip
>> sudo pip install sphinx
```

For more details, check here. But once you're set up, you'll follow the same boot, configure and make instructions as for Windows.

CONCLUSION

So that wraps up our first look at GHC. There’s plenty of work to do just to get it to build! But you should now move onto part 2, where we’ll start looking at some of the simplest modifications we can make to GHC. That way, we can start getting a feel for how the code base works.

If you haven’t written Haskell, it’s hard to appreciate the brilliance of GHC! Get started by downloading our Beginners Checklist and reading our Liftoff Series!

Revision #1
Created 11 March 2022 16:30:09 by gasick
Updated 11 March 2022 17:11:16 by gasick