GHC Newcomer Guide

Hacking GHC

31.05.2018 @HaskellerZ by Andreas Herrmann

Why Hack GHC?

- Compiler is among the most important tools
- Good to know your tools
- Better understand the language
- You can improve GHC
- It's just cool ;)

How to get started?

- GHC Newcomer guide
  https://ghc.haskell.org/trac/ghc/wiki/Newcomers
  Prepare your machine, fetch the code, build it, get started
- Building guide
  https://ghc.haskell.org/trac/ghc/wiki/Building
  In-depth about build system
- GHC Commentary
  https://ghc.haskell.org/trac/ghc/wiki/Commentary
  Details about concepts and source structure
- Community
  ghc-devs mailing list
  #ghc on FreeNode
- More references in the end

Dependencies - Prepare your machine

- Depends on your platform, follow the guide https://ghc.haskell.org/trac/ghc/wiki/Building/Preparation
- If all else fails, use Docker

```
$ docker run --rm -i -t -v "$PWD":/home/ghc \
      gregweber/ghc-haskell-dev /bin/bash
```

Get the sources

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

GitHub mirror requires extra config

```
$ git config --global \
    url."git://github.com/ghc/packages-".insteadOf \
    git://github.com/ghc/packages/
$ git clone --recursive git://github.com/ghc/ghc
```

Git submodules

- When pulling changes

```
$ git pull --recurse-submodules=yes
```

- When switching branches

```
$ git checkout <other-branchname>
$ git submodule update --init --recursive
```

- More details in GHC wiki - Git submodules

Git worktrees

- Work on different branches in different worktrees.
- Can use git worktrees to work on different branches.

`~/.gitconfig`:

```
[alias]
  # Adapted from https://gitlab.com/clacke/gists/blob/
  wta = worktree add --detach
  wtas = "!bash -ec 'if (($# != 1)); then echo >&2 \"U
```

```
$ git wtas ../ghc-my-new-feature
$ git submodule update --init
```

https://ghc.haskell.org/trac/ghc/wiki/WorkingConventions/Git

Configure the build

- Default build config
  - production settings
  - slow build
- Development build
  - less optimization
  - faster build
  - debug mode

```
$ cp mk/build.mk.sample mk/build.mk
```

`mk/build.mk`:

```
BuildFlavour = devel2  # mk/flavours/devel2.mk
```

GHC build stages

- Stage 0: The installed GHC (bootstrap compiler)
- Stage 1: Bootstrap builds dependencies, libraries, and stage 1
- Stage 2: stage 1 builds libraries, rts, and stage 2
- Optional stage 3: build again from stage 2 for testing

First build

```
$ ./boot        # generate configure scripts
$ ./configure   # configure Makefiles, etc.
$ make -j4      # builds stage 1 and stage 2
```

Go fetch coffee/tea/milk...

If it built, try it out

```
$ ./inplace/bin/ghc-stage2 --interactive
```

Faster rebuild

- Don't rebuild stage 1

  `mk/build.mk`:

  ```
  stage=2
  ```

- Run `make` where you made changes (`compiler`, `utils`, `ghc`, `libraries`)
- Use `make fast` (except after `git pull`)

Sanity check

Pick an error message

```
$ ./inplace/bin/ghc-stage2 --interactive
ghci> a + 2
<interactive>:2:1: error: Variable not in scope: a
```

Find and change it

```
$ grep -rl '"Variable not in scope:"'
compiler/typecheck/TcErrors.hs
$ $EDITOR compiler/typecheck/TcErrors.hs
$ (cd compiler && make fast -j4)
```

And try again

```
$ ./inplace/bin/ghc-stage2 --interactive
ghci> a + 2
Doesn't look like anything to me: a
```

Random hint

- `make help`: List relevant targets in each subdirectory

Hadrian

- New Shake based build system
- Coming soon (8.6?)
- See `hadrian/README.md`

Source code overview

- Top-level files: Largely build-system & communication
    - `HACKING.md`: Hacking & contributing guide
- `libraries/`: GHC's dependencies (boot packages)
- `compiler/`: `ghc` library package
  Parser, typechecker, AST, core, STG, code-generators, ...
- `ghc/`: `ghc-bin` executable package
- `rts/`: Runtime system - C implementation
  Storage manager, garbage collector, Scheduler, ...
- `docs/`: GHC documentation
- `testsuite/`: The test suite
- See commentary for more details

Compilation pipeline

- Parser → Parse tree
- Desugar → Core
- STGify → STG (Spineless Tagless G-machine)
- CodeGen → C--
- Backend
  - Native code generator → Assembly (default)
  - LLVM backend → LLVM IR (`-fllvm`)
    LLVM → Assembly
  - C backend → C (`-fvia-c` outdated)
    GCC → Assembly

Picking an issue

- Newcomers guide: "Finding a ticket"
- Tickets by milestone
- Ask on IRC or mailing-list
- You found a bug?

Contributing

- Have a ticket on Trac
- Communicate that you're working on it
- Add a test-case
- Fix the bug
- Test `make test TEST="XXX YYY"`
- Refer to ticket number in commit message
- Validate
    - Phabricator automatically validates on Harbormaster
    - Locally using `./validate`

Contributing - Phabricator

- Code review and automated build tool
- Sign-up https://phabricator.haskell.org/
- Add SSH key https://phabricator.haskell.org/settings/
- Install recent Arcanist CLI `arc`
- Install user certificate `arc install-certificate`
- Submit your changes `arc diff HEAD^` (`arc diff <rev-before-changes>`)
- Update ticket on Trac

Example

References

- GHC wiki
  - Newcomers guide
  - GHC Commentary
- Experience reports
  - by Andrew Gibiansky
  - by Annie Cherkaev
  - by Moritz Angermann
- Dive into GHC 1 2 3 by Stephen Diehl