Greater customization of GHCi prompt

Author: Nikita Sazanovich

Mentor: Nikita Kartashov

SPb AU, spring 2016

GHC[i]

The Glasgow Haskell Compiler, or simply GHC, is a state-of-the-art, open source, compiler and interactive environment for the functional language Haskell.

GHCi is GHC’s interactive environment.

GHC is heavily dependent on its users and contributors.

GHC Ticket #5850

Most shells allow arbitrary user customization of the prompt. The bash prompt has numerous escape sequences for useful information, and if those aren't enough, it allows arbitrary command calls.

GHCi should gain similar customization abilities. Ways to implement this may include:

1. addition of more escape sequences.
2. addition of a single extra escape sequence with one parameter (an external command call).
3. redesigning the :set prompt option to take a Haskell function.

Implementing the feature

1. Haskell Language.
2. Looking for inspiration: bash escape sequences.
3. Understanding the GHC codebase.
4. Refactoring the existing GHC code.
5. Writing the code: parsing the prompt, lazy evaluation, cross-platform.
6. Testing the feature locally.

Details: Parsing the prompt

```
   :set prompt "%t %w: ghci> "
set prompt "%t %w: ghci> "
prompt "%t %w: ghci> "
"%t %w: ghci> "
%t %w: ghci>
 %w: ghci>
%w: ghci>
: ghci>
 ghci>
...
```

Details: Lazy evaluation

Eager evaluation.

```
:set prompt "%t %w: ghci> "
READ AND STORE IN PROMPT_STRING
IF NEED_TO_PRINT_PROMPT
  THEN PARSE_AND_PRINT PROMPT_STRING
```

Lazy evaluation.

```
:set prompt "%t %w: ghci> "
CREATE_FUNC MAKE_PROMPT = CURRENT_TIME + " " + CURRENT_DIRECTORY +
": ghci> "
IF NEED_TO_PRINT_PROMPT
  THEN PRINT MAKE_PROMPT
```

Details: Cross-platform

```
getUserName :: IO String
getUserName = do
#ifdef mingw32_HOST_OS
  getEnv "USERNAME"
    `catchIO` \e -> do
      putStrLn $ show e
      return ""
#else
  getLoginName
#endif
```

Contributing the patch to GHC

- Communicating with GHC developers.
- Writing as clear and expressive code as possible.
- Passing code reviews.
- Updating documentation.
- Creating GHC tests.
- Working with such software as Git, Trac, Phabricator, Arcanist, Lint, etc.

Results: Escape sequences

```
Prelude> :set prompt
prompt                     prompt-cont              prompt-cont-function
prompt-function
Prelude> :set prompt "%t %w: ghci> "
13:05:59 /Users/niksaz, ghci> let filterPrime (p:xs) = p :
filterPrime [x | x <- xs, x `mod` p /= 0]
13:06:05 /Users/niksaz, ghci> let primes = filterPrime [2..]
13:06:08 /Users/niksaz, ghci> primes !! 10
31
13:06:14 /Users/niksaz, ghci> :set prompt "%l:%u %d> "
6:niksaz Mon May 30>
```

Results: %call(cmd)

```
Prelude> :set prompt "branch: %call(git rev-parse --abbrev-ref HEAD)
ghci> "
branch: present
ghci> :! git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
branch: master
ghci>
```

Results: Prompt Function

```
Prelude> let last_module :: [String] -> Int -> IO String
Prelude|     last_module ls _ | null ls   = return "empty> "
Prelude|                      | otherwise = return $ last ls ++ "> "
Prelude System.Process Data.List Control.Applicative> :set prompt-
function last_module
Control.Applicative> import Data.Complex
Data.Complex>
```

Want to play with the feature?

It is in the GHC master branch: https://github.com/ghc/ghc
The commit: https://github.com/ghc/ghc/commit/533037cc58a7c50e1c014e27e8b971d53e7b47bd
Instructions for building the GHC:
https://ghc.haskell.org/trac/ghc/wiki/Newcomers

Phabricator: https://phabricator.haskell.org/D2084

Thanks for your attention!