

Please note that GitHub no longer supports old versions of Firefox.
We recommend upgrading to the latest [Safari](#), [Google Chrome](#), or [Firefox](#).

[Learn more](#)

[Ignore](#)

 [krisajenkins](#) / [vim-pipe](#)


Join GitHub today


[Dismiss](#)

GitHub is home to over 20 million developers working together to host and review code, manage projects, and build software together.


[Sign up](#)

Send a vim buffer through a command and instantly see the output.

 82 commits

 2 branches

 7 releases

 Fetching contributors


 MIT

Branch: [master](#) ▾

[New pull request](#)

[Find file](#)

[Clone or download](#) ▾

 Fetching latest commit...

 [doc](#)

 [plugin](#)

 [LICENSE.txt](#)

 [README.md](#)

 [test.sql](#)

 [README.md](#)

Vim Pipe - A Productivity-Boosting Plugin

Do you do this?

```

      hack <----- alt-tab <----- react
      |               |               |
+-----+           +-----+
| code code |       | result result |
| code      |       | result        |
|           |       |               |
|           |       |               |
|           |       |               |
|           |       |               |
+-----+           +-----+
      |               |
      :w -----> alt-tab -----> invoke
    
```

This plugin lets you do this instead:

```

+-----+
| code code | |-----\
| code      | |         |
|           | |         |
|-----|   | <LocalLeader>r
| result result | |
| result      | |<-----/
|           |
    
```

+-----+

Which saves tonnes of time. It's even faster than using screen or tmux.

It's useful for developing SQL queries, fast HTML previews, markdown-checking, and anything that needs to pass through a shell command while you develop.

Detail

You associate a shell command with your file, something that will take your buffer on STDIN and show the result on STDOUT. For example, if you're editing an SQL query, that command might be `psql mydatabase`.

Having done that, `<LocalLeader>r` will run the current buffer against that command and show you the results. You no longer need to save-switch-execute-switch, which makes life faster and easier.

Installation

- Install [Pathogen](#). (You're already using Pathogen, right?)
- Clone this project into `~/.vim/bundle/vim-pipe`.
- Set a `b:vimpipes_command` variable for your buffer. The easiest way is to add a `let` command in `~/.vim/ftplugin/<filetype>.vim`. For example:

```
" In ~/.vim/ftplugin/sql.vim
let b:vimpipes_command="psql mydatabase"
" In ~/.vim/ftplugin/markdown.vim
let b:vimpipes_command="multimarkdown"
```

See below for various examples.

Usage & Tips

Once `b:vimpipes_command` is configured, type `<LocalLeader>r` to get the list results. There's no need to save the file first. It works on the current buffer, not the contents on disk.

You can set `g:vimpipes_silent=1` to disable runtime information in the output window.

PostgreSQL

```
" In ~/.vim/ftplugin/sql.vim
let b:vimpipes_command="psql mydatabase"
```

See also [vim-postgresql-syntax](#).

Oracle

If you have an OPS\$ login, it's as simple as:

```
" In ~/.vim/ftplugin/sql.vim
let b:vimpipes_command="sqlplus -s /"
```

HTML

This is only text-based, obviously, but can still speed up initial development.

```
" In ~/.vim/ftplugin/html.vim
let b:vimpipes_command="lynx -dump -stdin"
```

JavaScript

I usually run JavaScript in a browser, so I bind vim-pipe to JSLint, for code-quality checking on-the-fly.

```
" In ~/.vim/ftplugin/javascript.vim
let b:vimpipe_command='jslint <(cat)'
```

Note: JSLint doesn't accept input on STDIN, so this configuration uses bash's virtual file support. `<(cat)` takes the STDIN and re-presents it to look like a regular file.

JSON

I find attaching vim-pipe to a pretty-printer useful for development:

```
" Vim doesn't set a FileType for JSON, so we'll do it manually:
" In ~/.vimrc
autocmd BufNewFile,BufReadPost *.json setlocal filetype=javascript.json

" In ~/.vim/ftplugin/javascript.vim
" Requires that you have Python v2.6+ installed. (Most *nix systems do.)
let b:vimpipe_command="python -m json.tool"
```

Markdown

Fast-preview the HTML:

```
" In ~/.vim/ftplugin/markdown.vim
let b:vimpipe_command="multimarkdown"
let b:vimpipe_filetype="html"
```

Or combine with the HTML tip to preview the rendered result:

```
" In ~/.vim/ftplugin/markdown.vim
let b:vimpipe_command="multimarkdown | lynx -dump -stdin"
```

MongoDB

Is there an official FileType for MongoDB query files? Let's say it's `mongoql`, for all files `*.mql`:

```
" In ~/.vimrc
autocmd BufNewFile,BufReadPost *.mql setlocal filetype=mongoql

" In ~/.vim/ftplugin/mql.vim
let b:vimpipe_command="mongo"
let b:vimpipe_filetype="javascript"
```

Then try editing a file called `somequery.mql` with something like this in:

```
use books;
db.book.find(null, {author: 1, title: 1 });
db.runCommand({dbStats: 1});
```

Help

See `:help vim-pipe` for more.

FAQ

What's the difference between this and `:make` ?

The biggest difference is the way the output is presented & read. If the whole of the output is interesting, use Vim Pipe. If the only interesting part of the output is a summary of errors/warnings, use `:make`.

Does Vim Pipe fork/exec?

..or is there a long-lived background process?

It forks. So it works well with commands you'd invoke frequently from the command line, but not so well with something like `javac`, which has a (deathly) slow startup time.

Credits

Thanks to Steve Losh for his excellent guide to Vimscript, [Learn Vimscript the Hard Way](#), and Meikel Brandmeyer of [vimclojure](#) for the inspiration.

Thanks to [Markus Seeger](#) and [Dhruva Sagar](#) for their contributions.