How to make your life just a bit easier with vim plugins (and .vimrc)

Rucheng Diao
Lab 10-min talk
20210723

Update: codes and slides are available at https://github.com/diaorch/swiss_army_codes

Getting into vim plugin is intimidating

- Text-only interface and process
- Strangely simplified version or lack of documentation for installing vim plugins
  - Follow specific example of plugins you have installed before
  - Search for "unofficial" guides
- It may take a surprisingly long time to make up your mind
  - For example: colorschemes

Make use of plugin managers

- Do you need a plugin manager?
  - With a small amount of plugins to handle, probably not
  - But can simplify installing processes, or some plugins only have instructions with plugin managers
- Example: vim-plug, pathogen, vundule

README.md

Nvim-R

This is the development code of Nvim-R which improves Vim's support to edit R code.

Installation

If you use a plugin manager, such as vim-plug, follow its instructions on how to install plugins from github.

To install the stable version of the plugin, if using vim-plug, put this in your `vimrc` / `init.vim` :

```
Plug 'jalvesaq/Nvim-R', {'branch': 'stable'}
```

The `stable` branch is a copy of the last released version plus minor bug fixes eventually found after the release. I plan to keep the stable branch compatible with Ubuntu LTS releases, and the master branch compatible with Ubuntu normal releases. If you need an older version, you could try either the `oldstable` branch or one of the tagged versions.

James Eapen maintains an online version of the plugin's documentation. Please, read the section Installation for details.

What are some useful plugins, or general vim tricks?

- "Turning vim into an R IDE"
  - Takes time and multiple plugins
  - Version conflicts
- Syntax highlighting
  - E.g. Markdown
- "Colorschemes"
  - Color Schemes CAN be installed without plugins
  - Why use color scheme plugins, then?
- Templates
  - For files with specific suffix, pre-define a template for whenever you create such a new file
  - My templates: Python, R, Bash
  - Underrated upside of using a template: makes you actually write documentations
- Manual highlighting

Syntax highlighting for Markdown in vim

```
### The counts of genes of significantly changed expression are not necessarily reflective of the functional sign

#### Are counts of the up- or down-regulated genes significantly different from random (0.5 up and 0.5 down)?

The plan is for each GO term of interest: consider all genes associated with the GO term, are the number of up- v
y Binomial test?

That was what I recalled that we talked about doing before. But I don't know if I am understanding Peter's idea o
 it from the point of view of number of significant genes? In that case, I would need to understand the backgroun
us distribution because I would need to figure out what the expected singificance levels are given the backgroun
n p = (up-regulated gene counts / all associated gene counts). Then I am kind of stuck here not sure how to appro
I was thinking how DESeq2 do their hypothesis testing. But LRT is specifically about testing two models with "one
one found after imposing some constrain" ([Wikipedia: Likelihood-ratio test](https://en.wikipdiea.org/wiki/Likeli
eq2 used is the Wald test.But again, Wald test "asseses constraints on statistical parameters based on the weight
alue under the null hypothesis" ([Wikipedia: Wald test](https://en.wikipedia.org/wiki/Wald_test)). So I don't thi
he rate of up-regulated genes among all genes associated with the GO term, what is the other rate I would define
he exact confidence intervals from R `rateratio.test` come directly from `binom.test` ([documentation](https://cr
nd tests like Chi-square or Fisher's exact test are for contingency tables assessing for independence between two
 is practically used for small sample sizes (esp. more than 20% of the cells have expected frequencies < 5, [PubM
)).

For now: binomial test for p = 0.5, two-sided. See table below in the "Statistical test results" section for resu

//20200504 start: test for significant genes per GO term of interest

I talked to Peter in our weekly meeting 20200504 and we decided that for the idea of testing the counts of signif
nterest, whether the number of significant genes is different from the expected counts as from the overall gene p
 not-sginificant and associated with a GO term vs not associated with the GO term. For example, for term GO:00052

|         | In-GO | Not-in-GO |   sum |
|---------|-------|-----------|-------|
|     sig |     7 |       483 |   490 |
| not-sig |    29 |     16539 | 16568 |
|     sum |    36 |     17022 | 17058 |

Some basic numbers: total gene counts in DESeq2: 23418, total gene counts with non-NA adjusted p-values in DESeq2
.

The Fisher's exact tests were done with parameters `r = 1, alternative = 'two.sided', hybrid = 1`. The p-values a
 reported in a separated table in the "Statistical test results" section below.

//20200504 end: performed Fisher's exact test for significant gene counts per GO term

#### Are the distribution of gene expression changes significantly different from random (log2FC centered at 0)?
```

Using vim templates

- Example tutorials: https://shapeshed.com/vim-templates/
- General steps
  - Make a "skeleton file"
  - Adding to .vimrc file to populate future files

```
if has("autocmd")
  augroup templates
    autocmd BufNewFile *.sh 0r ~/.vim/templates/skeleton.sh
  augroup END
endif
```

- That's all!

Template examples

```
(base) diaorch@serenity:/corexfs/diaorch$ vim test.sh
```

serenity.med.umich.edu - PuTTY

```
#!/usr/bin/env bash
```

```
(base) diaorch@serenity:/corexfs/diaorch$ vim test.py
```

```python
#!/usr/bin/env python3

"""
docstring
"""

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

def main():
    ...

if __name__ == '__main__':
    main()
```

```
(base) diaorch@serenity:/corexfs/diaorch$ vim test.R
```

```
#!/usr/bin/env Rscript
DOC = "This script illustrates how R can parse arguments if called from the command line."

### Script template from:
#   https://github.com/molgenis/molgenis-pipelines/wiki/Template-for-R-scripts
###

# Define your constants here
    #-- const1 = "first"   --#
    #-- const2 = "second"  --#

# Retrieve command line parameters
    suppressPackageStartupMessages(library("argparse"))

    # Create parser object
    parser <- ArgumentParser(description = DOC)

    parser$add_argument("-c", "--count", type="integer", default=5, help="Number of random normals to generate [default %(default)s]", metavar="number")
    parser$add_argument("--generator", default="rnorm", help = "Function to generate random deviates [default \"%(default)s\"]")
    parser$add_argument("--mean", default=0, type="double", help="Mean if generator == \"rnorm\" [default %(default)s]")
    parser$add_argument("--sd", default=1, type="double", metavar="standard deviation", help="Standard deviation if generator == \"rnorm\" [default %(default)s]")
    parser$add_argument("-v", "--verbose", action="store_true", default=TRUE, help="Print extra output [default]")
    parser$add_argument("-q", "--quietly", action="store_false", dest="verbose", help="Print little output")
    # Please note: you can add 'required = TRUE' for parameters that are required

    # Parse command line parameters
    args <- parser$parse_args()

# Load your functions here
    #-- Example: source("main.functions.R") --#

# Print inputs only if cmnd-line parameters correct and verbose 'true' (in case of errors, use 'sterr()')
if (args$verbose) {
    write("You have used the following arguments:", stdout())
    write(paste("\t--count:  ", args$count), stdout())
    write("other parameters: ...", stdout())
    write("\nStart with analysis...\n", stdout())
}
# Do some operations based on user input
if("rnorm" == args$generator) {
    cat(paste(rnorm(args$count, mean=args$mean, sd=args$sd), collapse="\n"))
} else {
    cat(paste(do.call(args$generator, list(args$count)), collapse="\n"))
}
cat("\n")
```

Examples of manual highlighting

```
so that is because there are also multiple UniProt ID mapping to the same gene

```
> print(dim(unique(normed_conf_format)))
[1] 7527620       4
> 4229 * 1780
[1] 7527620
```

==TODO==: deal with the duplication of UniProt ID more upstream. For now, use unique to get the table of right size.
```

```
According to the [DESeq2 Reference manual](https://

> ... For consistency with `results`, the column na

IMPORTANT NOTE: the z-score, coming from a differen
```