vim-noweb – extend Vim to support Noweb

Edward McGuire

March 23, 2024

Abstract

`vim-noweb.nw` is the source file for a Vim plugin that adds support for Noweb `.nw` source file editing, such as syntax highlighting. Within `.nw` files, documentation chunks get TEX syntax highlighting. Code chunks get the syntax highlighting of the code language, if it can be identified. Otherwise they get a generic "String" highlighting.

New languages can be added easily by passing a language name, filename pattern, and syntax file name to a registration function.

Contents

1 Overview

The *vim-noweb* package is four files: a file type detection Vim script, a syntax highlighting Vim script, a `README`, and a `LICENSE`, saved to a tarball. The package is compiled from the source file `vim-noweb.nw` using *Noweb* and *make*. The finished product can be found at

4a ⟨ *url* 4a⟩≡ (16b 18c)

```
https://metaed.com/papers/vim-noweb/
```

The package development environment is Slackware64 Linux, release 15.0.

4b ⟨*makefile* 4b⟩≡ 5b▷

```
# ⟨boilerplate 1 21a⟩
# ⟨boilerplate 2 21b⟩
# ⟨boilerplate 3 21c⟩
# ⟨boilerplate 4 21d⟩
usage ::
        @echo 'makefile: usage:'
        @echo '  make all'
        @echo '  make install (implies all)'
        @echo '  make package (implies install)'
        @echo '  make website (implies package)'
        @echo '  make commit'
        @echo '  make clean'
all ::
install :: all
package :: install
website :: package
commit ::
        git commit -av -uno
clean ::
        rm -f *~ .*~
```

This code is written to file `makefile`.

2 Filetype detection

The filetype detection file `ftdetect-noweb.vim` identifies *Noweb* source files by the filename suffix `.nw`. Its name in the bundle is `ftdetect/noweb.vim`.

5a ⟨*ftdetect-noweb.vim* 5a⟩≡ 5c▷

```
" ⟨boilerplate 1 21a⟩
" ⟨boilerplate 2 21b⟩
" ⟨boilerplate 3 21c⟩
" ⟨boilerplate 4 21d⟩
```

This code is written to file `ftdetect-noweb.vim`.

5b ⟨*makefile* 4b⟩+≡ ◁4b 6b▷

```
all :: ftdetect-noweb.vim
install :: ~/.vim/ftdetect/noweb.vim
~/.vim/ftdetect/noweb.vim : ftdetect-noweb.vim
        mkdir -p ~/.vim/ftdetect && cp $< $@
ftdetect-noweb.vim : vim-noweb.nw.sentinel ;
clean ::
        rm -f ftdetect-noweb.vim
```

Detection is enabled by a single-line declaration that the filename pattern `*.nw` identifies a file of type `noweb`.

5c ⟨*ftdetect-noweb.vim* 5a⟩+≡ ◁5a

```
autocmd  BufNewFile,BufRead  *.nw  set filetype=noweb
```

3 Syntax highlighting

`syntax-noweb.vim` contains syntax highlighting directives for *Noweb* source files. Its name in the bundle is `syntax/noweb.vim`.

6a ⟨*syntax-noweb.vim* 6a⟩≡ 6c▷

```
" ⟨boilerplate 1 21a⟩
" ⟨boilerplate 2 21b⟩
" ⟨boilerplate 3 21c⟩
" ⟨boilerplate 4 21d⟩
```

This code is written to file `syntax-noweb.vim`.

6b ⟨*makefile* 4b⟩+≡ ◁5b 15a▷

```
all :: syntax-noweb.vim
install :: ~/.vim/syntax/noweb.vim
~/.vim/syntax/noweb.vim : syntax-noweb.vim
        mkdir -p ~/.vim/syntax && cp $< $@
syntax-noweb.vim : vim-noweb.nw.sentinel ;
clean ::
        rm -f syntax-noweb.vim
```

The stock TEX syntax file `tex.vim` is used to syntax-highlight everything outside code chunks.

> **NOTE:** All regular expressions below use the *Vim* "very magic" syntax. This is done by prefixing each expression with `\v`.

> **NOTE:** According to `syn-pattern` in the manual, syntax patterns are always interpreted like the `magic` option is set, no matter what the actual value of `magic` is. Hence the "very magic" prefix is always an override of "magic". I prefer "very magic" for its readability, and for its similarity to Extended Regular Expressions. Using "very magic", there is no need to escape parentheses (grouping), vertical line (alternation), or braces (repetition). But maybe "magic" would be easier for other *Vim* programmers to read.

6c ⟨*syntax-noweb.vim* 6a⟩+≡ ◁6a 7a▷

```
if exists("b:current_syntax") | finish | endif
syntax include @SyntaxTeX syntax/tex.vim
unlet b:current_syntax
syntax region Normal start=/\v%^/ end=/\v%$/ contains=@SyntaxTeX
```

Defines:
`SyntaxTex`, never used.

Noweb syntax highlighting is declared as extensions to the TEX syntax.

7a ⟨*syntax-noweb.vim* 6a⟩+≡ ◁6c 7b▷
⟨*syntax-noweb.vim recognition* 11a⟩
⟨*syntax-noweb.vim doc chunk* 8⟩
⟨*syntax-noweb.vim quote* 9⟩
⟨*syntax-noweb.vim code chunk* 10a⟩
⟨*syntax-noweb.vim code use* 14a⟩

The syntax file ends by setting the `current_syntax` variable in buffer scope.

7b ⟨*syntax-noweb.vim* 6a⟩+≡ ◁7a
```
let b:current_syntax = "noweb"
```

The *Noweb* syntax is described partly in the manual, partly in the source. Some notable quotations that were helpful in writing this syntax file:

- "A module name is any text enclosed in double angle brackets."
- "Double angle brackets may be escaped in source by preceding them with the at sign."
- "No other character, not even the at sign, needs to be escaped [in source]."
- "A module definition is a module name, followed by one equals sign, possibly followed by white space, on a line by itself."

"Test" subsections below are used to check highlighting. Install the plugin and then open the file `vim-noweb.nw` in *Vim*.

3.1 Documentation chunk directive

A documentation chunk is introduced by a single at-sign. Indexing or plain documentation can follow.

8 ⟨*syntax-noweb.vim doc chunk* 8⟩≡ (7a)

```
syntax match PreProc "\v^[@]($| [%]def .*| )" containedin=@SyntaxTeX
```

3.1.1 Test

plain documentation

3.2 Quoted code in documentation

Double square brackets bracket a code quotation within a documentation chunk. It can span lines. It can contain code uses. It can contain nested quotes. A preceding at-sign (@) escapes quoting. The language cannot be determined, so plain String highlighting is used.

9 ⟨*syntax-noweb.vim quote* 9⟩≡ (7a)

```
syntax region nowebCodeQuotation matchgroup=Operator
        \ start="\v[@]@<!\[\["
        \ end="\v\]\]"
        \ containedin=@SyntaxTeX
        \ contains=nowebCodeUse,nowebCodeQuotation
highlight link nowebCodeQuotation String
```

Defines:
`nowebCodeQuotation`, used in chunks 10a and 14a.
Uses `nowebCodeUse` 14a.

3.2.1 Test

```
CODE QUOTATION
CODE USE WITHIN CODE QUOTATION
⟨code use (never defined)⟩
[[NESTED CODE QUOTATION ]]
```

3.3 Generic code-chunk declaration and body

A code chunk declaration is introduced by name in double angle brackets followed by equals-sign and optional trailing whitespace. The name can contain code quotations. The code body begins on the next line.

A code-chunk body is terminated by a new doc or code chunk introducer. It can span lines. It can contain code uses.

Generic syntax is defined first so that specific syntaxes override it. Like code quotations, it is given plain String highlighting.

10a ⟨*syntax-noweb.vim code chunk* 10a⟩≡ (7a) 11b▷

```
syntax match nowebCodeChunkDecl "\v^[<][<].*[>][>][=][ \t]*$"
        \ skipnl
        \ containedin=@SyntaxTeX
        \ contains=nowebCodeQuotation
        \ nextgroup=nowebCodeChunkBody
highlight link nowebCodeChunkDecl PreProc
syntax region nowebCodeChunkBody
        \ start="\v.*"
        \ end="\v^([@]($| ))|([<][<].*[>][>][=][ \t]*$)"me=s-1
        \ contained
        \ contains=nowebCodeUse
highlight link nowebCodeChunkBody String
```

Defines:
`nowebCodeChunkBody`, never used.
`nowebCodeChunkDecl`, never used.
Uses `nowebCodeQuotation` 9 and `nowebCodeUse` 14a.

3.3.1 Test

10b ⟨*test generic syntax* 10b⟩≡

```
This is a sample generic syntax code chunk.
```

10c ⟨*test* `with code quotation` *generic syntax* 10c⟩≡

```
This is a sample generic syntax code chunk.
```

3.4 Language-specific code chunk declaration and body

Credit for this technique goes to the developers of the *Ant* syntax file `ant.vim` distributed with *Vim*.

11a ⟨*syntax-noweb.vim recognition* 11a⟩≡ (7a)

```
function NowebRecognize( language, pattern, file )
        execute 'syntax match nowebCodeChunkDecl' . a:language
                \ . ' "\v^[<][<]' . a:pattern . '[>][>][=]\s*$"'
                \ . ' skipnl'
                \ . ' containedin=@SyntaxTeX'
                \ . ' contains=nowebCodeQuotation'
                \ . ' nextgroup=nowebCodeChunkBody' . a:language
        execute 'highlight link nowebCodeChunkDecl' . a:language . ' PreProc'
        execute 'syntax include @Syntax' . a:language . ' syntax/' . a:file
        execute 'unlet b:current_syntax'
        execute 'syntax region nowebCodeChunkBody' . a:language
                \ . ' keepend'
                \ . ' start="\v.*"'
                \ . ' end="\v^([@]($| ))|([<][<].*[>][>][=][ \t]*$)"me=s-1'
                \ . ' contained'
                \ . ' contains=nowebCodeUse,@Syntax' . a:language
endfunction
```

Defines:
`nowebRecognize`, never used.
Uses `nowebCodeUse` 14a.

This is the list of languages currently set up to be recognized and syntax-highlighted using a stock syntax file.

11b ⟨*syntax-noweb.vim code chunk* 10a⟩+≡ (7a) ◁10a 12a▷

```
call NowebRecognize( 'Awk'     ,     '.*\.awk(|\s.*)' ,     'awk.vim' )
call NowebRecognize( 'Bash'    ,    '.*\.bash(|\s.*)' ,    'bash.vim' )
call NowebRecognize( 'Crontab' , '.*\.crontab(|\s.*)' , 'crontab.vim' )
call NowebRecognize( 'Gnuplot' ,      '.*\.gp(|\s.*)' , 'gnuplot.vim' )
call NowebRecognize( 'Make'    , '[mM]akefile(|\s.*)' ,    'make.vim' )
call NowebRecognize( 'Man'     ,    '.*\.[18](|\s.*)' ,     'man.vim' )
call NowebRecognize( 'Python'  ,      '.*\.py(|\s.*)' ,  'python.vim' )
call NowebRecognize( 'Sed'     ,     '.*\.sed(|\s.*)' ,     'sed.vim' )
call NowebRecognize( 'Sh'      ,      '.*\.sh(|\s.*)' ,      'sh.vim' )
```

Some stock syntaxes use the `extend` keyword when they define a region. It can cause their syntax highlighting to leak out past the end of a code block, because it overrides the `keepend` keyword. I have tested stock syntaxs that use `extend` to see if they leak. Those that do are not enabled by default. The local system operator will have to enable them manually. The following syntaxes are disabled for that reason:

12a ⟨*syntax-noweb.vim code chunk* 10a⟩+≡ (7a) ◁11b

```
" call NowebRecognize( 'C'     ,       '.*\.c(|\s.*)' ,       'c.vim' )
" call NowebRecognize( 'Perl'  ,      '.*\.pl(|\s.*)' ,    'perl.vim' )
" call NowebRecognize( 'Vim'   ,     '.*\.vim(|\s.*)' ,     'vim.vim' )
```

3.4.1 Test

12b ⟨*makefile example* 12b⟩≡

```
target :: dependency ; action
```

12c ⟨*makefile* `with quoting` *example* 12c⟩≡

```
target :: dependency ; action
```

12d ⟨*not makefile counterexample* 12d⟩≡

```
target :: dependency ; action
```

12e ⟨*makefile.c counterexample* 12e⟩≡

```
target :: dependency ; action
```

12f ⟨*example .vim* 12f⟩≡

```
unlet recognition
```

12g ⟨*example .vim* `with quoting` 12g⟩≡

```
unlet recognition
```

12h ⟨*counterexample .vi* 12h⟩≡

```
unlet recognition
```

12i ⟨*example try.pl* 12i⟩≡

```
use strict ;
```

12j ⟨*example try.pl* `with quoting` 12j⟩≡

```
use strict ;
```

12k ⟨*counterexample try.pli* 12k⟩≡

```
use strict ;
```

13a ⟨*example try.c* 13a⟩≡
```
main() { return ; }
```

13b ⟨*example try.c* `with quoting` 13b⟩≡
```
main() { return ; }
```

13c ⟨*counterexample try.cpp* 13c⟩≡
```
main() { return ; }
```

13d ⟨*example try.sh* 13d⟩≡
```
echo hello world
```

13e ⟨*example try.sh* `with quoting` 13e⟩≡
```
echo hello world
```

13f ⟨*counterexample try.shm* 13f⟩≡
```
echo hello world
```

13g ⟨*example try.pl that demonstrates a leak out of the code block* 13g⟩≡
```
/*
```

13h ⟨*example try.c that demonstrates a leak out of the code block* 13h⟩≡
```
/*
```

13i ⟨*example try.py* 13i⟩≡
```
# This is a comment
import sys # This is a comment
hello_text  =  "hello, world"
def hello_function():
        print( hello_text )
hello_function()
sys.exit(0)

"""
multiline string used as a comment
"""

"""
unterminated multiline string to test for leak out of the code block
```

3.5 Code use

Double angle brackets bracket a code use within a code chunk. It cannot span lines. When used more than once on a line, the closest close-brackets end a code use. It can contain quotes. A preceding at-sign (@) escapes a code use.

14a ⟨*syntax-noweb.vim code use* 14a⟩≡ (7a)

```
syntax match nowebCodeUse "\v[@]@<![<][<].{-}[>][>]"
        \ contained
        \ contains=nowebCodeQuotation
highlight link nowebCodeUse Identifier
```

Defines:
`nowebCodeUse`, used in chunks 9–11.
Uses `nowebCodeQuotation` 9.

3.5.1 Test

14b ⟨*code use test inner block* 14b⟩≡ (14c)

```
CODE TEST BODY
```

14c ⟨*code use test outer block* 14c⟩≡

```
123
        ⟨code use test inner block 14b⟩
456
```

14d ⟨*code use test inner block 2* 14d⟩≡ (14e)

```
CODE TEST BODY
```

14e ⟨*code use test outer block 2* 14e⟩≡

```
123⟨code use test inner block 2 14d⟩456
```

4 Compiling the developer manual, makefile, and sentinel file

15a ⟨*makefile* 4b⟩+≡ ◁6b 15b▷

```
all :: vim-noweb.pdf
vim-noweb.pdf : vim-noweb.tex
        latexmk -pdf vim-noweb
clean ::
        latexmk -C vim-noweb
vim-noweb.tex : vim-noweb.nw.sentinel ;
clean ::
        rm -f vim-noweb.tex
```

15b ⟨*makefile* 4b⟩+≡ ◁15a 15c▷

```
all :: makefile
makefile : vim-noweb.nw.sentinel ;
clean ::
        rm -f makefile
```

15c ⟨*makefile* 4b⟩+≡ ◁15b 16a▷

```
vim-noweb.nw.sentinel : vim-noweb.nw
        noweb $<
```

5 Compiling the plugin package

16a ⟨*makefile* 4b⟩+≡ ◁15c 16c▷

```
TMPDIR = /tmp
PKGDIR = noweb
PKG = $(TMPDIR)/$(PKGDIR)
package :: vim-noweb.tgz
vim-noweb.tgz : README LICENSE ftdetect-noweb.vim syntax-noweb.vim
        rm -rf  $(PKG)
        mkdir -p  $(PKG)/{ftdetect,syntax}
        cp  README LICENSE $(PKG)/
        cp  ftdetect-noweb.vim  $(PKG)/ftdetect/noweb.vim
        cp  syntax-noweb.vim  $(PKG)/syntax/noweb.vim
        (  cd $(TMPDIR)  &&  tar cf - $(PKGDIR) | gzip -9  ) > $@
```

16b ⟨*README* 16b⟩≡

```
Vim plugin for Noweb (.nw) files

vim-noweb is a Vim plugin that adds support for Noweb source file editing, such
as syntax highlighting.

TeX syntax highlighting is applied to documentation chunks.

Syntax highlighting applied to code chunks is that of the code language, if it
can be identified.

More information and support:
⟨ url 4a⟩
https://www.reddit.com/r/LitProg/
```

This code is written to file `README`.

16c ⟨*makefile* 4b⟩+≡ ◁16a 17b▷

```
README : vim-noweb.nw.sentinel ;
clean ::
        rm -f README
```

17a ⟨*LICENSE* 17a⟩≡

```
Copyright © 2023, 2024 Edward K. McGuire, Fort Worth, Texas. All rights reserved.
Redistribution and use of this software, with or without modification, is
permitted, provided that the following conditions are met:

1. Redistribution of this software must retain the copyright notice above, this
list of conditions, and the disclaimer below.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR \AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCY OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
```

This code is written to file `LICENSE`.

17b ⟨*makefile* 4b⟩+≡ ◁16c 18b▷

```
LICENSE : vim-noweb.nw.sentinel ;
clean ::
        rm -f LICENSE
```

6 Compiling the plugin website

18a ⟨ *html* 18a⟩≡ (18b)

```
/var/www/metaed.com/root/papers/vim-noweb
```

18b ⟨*makefile* 4b⟩+≡ ◁17b

```
all :: header.html footer.html htaccess
header.html footer.html htaccess : vim-noweb.nw.sentinel ;
clean ::
        rm -f header.html footer.html htaccess
website :: ⟨ html 18a⟩/header.html
website :: ⟨ html 18a⟩/footer.html
website :: ⟨ html 18a⟩/.htaccess
website :: ⟨ html 18a⟩/vim-noweb.pdf
website :: ⟨ html 18a⟩/vim-noweb.nw
website :: ⟨ html 18a⟩/vim-noweb.html
website :: ⟨ html 18a⟩/vim-noweb.tgz
⟨ html 18a⟩ : ; mkdir -p $@
⟨ html 18a⟩/header.html    : ⟨ html 18a⟩ header.html       ; cp header.html       $@
⟨ html 18a⟩/footer.html    : ⟨ html 18a⟩ footer.html       ; cp footer.html       $@
⟨ html 18a⟩/.htaccess      : ⟨ html 18a⟩ htaccess          ; cp htaccess          $@
⟨ html 18a⟩/vim-noweb.pdf  : ⟨ html 18a⟩ vim-noweb.pdf     ; cp vim-noweb.pdf     $@
⟨ html 18a⟩/vim-noweb.nw   : ⟨ html 18a⟩ vim-noweb.nw      ; cp vim-noweb.nw      $@
⟨ html 18a⟩/vim-noweb.html : ⟨ html 18a⟩ vim-noweb.nw.html ; cp vim-noweb.nw.html $@
⟨ html 18a⟩/vim-noweb.tgz  : ⟨ html 18a⟩ vim-noweb.tgz     ; cp vim-noweb.tgz     $@
vim-noweb.nw.html : vim-noweb.nw
        vim -c 'set noundofile' -c TOhtml -c wqa $<
clean ::
        rm -f vim-noweb.nw.html
```

18c ⟨*header.html* 18c⟩≡

```
<!-- ⟨boilerplate 1 21a⟩ -->
<!-- ⟨boilerplate 2 21b⟩ -->
<!-- ⟨boilerplate 3 21c⟩ -->
<!-- ⟨boilerplate 4 21d⟩ -->
<h1> ⟨ url 4a⟩ </h1>
<p>
This is the home of <code>vim-noweb</code>, a Vim syntax highlighting plugin for
Noweb source files.
```

This code is written to file `header.html`.

19 ⟨*footer.html* 19⟩≡

```
<!-- ⟨boilerplate 1 21a⟩ -->
<!-- ⟨boilerplate 2 21b⟩ -->
<!-- ⟨boilerplate 3 21c⟩ -->
<!-- ⟨boilerplate 4 21d⟩ -->
<p>
Copyright © 2023, 2024 Edward K. McGuire, Fort Worth, Texas. All rights reserved.
Redistribution and use of this software, with or without modification, is
permitted, provided that the following conditions are met:
<p>
1. Redistribution of this software must retain the copyright notice above, this
list of conditions, and the disclaimer below.
<p>
THIS SOFTWARE IS PROVIDED BY THE AUTHOR \AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCY OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
```

This code is written to file `footer.html`.

20 ⟨*htaccess* 20⟩≡

```
# This file is part of the vim-noweb package.
# Copyright (c) 2023, 2024 Edward K. McGuire.
# It was compiled from vim-noweb.nw using Norman Ramsey's Noweb.
<IfModule autoindex_module>
Options         +Indexes
HeaderName      header
ReadmeName      footer
IndexOptions    FancyIndexing
IndexOptions    +IgnoreClient
IndexOptions    +VersionSort
IndexOptions    +Charset=UTF-8
IndexOptions    +NameWidth=* DescriptionWidth=*
# Ignore hidden files. The syntax of IndexOptions filename patterns makes it
# impossible to ignore two-character hidden files without also ignoring ..
# (parent directory), so we ignore three-character hidden files and longer.
IndexIgnore     .??*
IndexIgnore     *~
IndexIgnore     header.html footer.html
AddDescription  "Current technical paper (PDF)"         vim-noweb.pdf
AddDescription  "Current source (Noweb)"                vim-noweb.nw
AddDescription  "How source looks highlighted"          vim-noweb.html
AddDescription  "Current plugin (compressed tar)"       vim-noweb.tgz
</IfModule>
```

This code is written to file `htaccess`.

7 Boilerplate

The comment lines below appear at the top of each text file in the distribution.

21a ⟨*boilerplate 1* 21a⟩≡ (4–6 18c 19)

```
This file is part of the vim-noweb package.
```

21b ⟨*boilerplate 2* 21b⟩≡ (4–6 18c 19)

```
Copyright (c) 2023, 2024 Edward K. McGuire.
```

21c ⟨*boilerplate 3* 21c⟩≡ (4–6 18c 19)

```
It was compiled from vim-noweb.nw using Norman Ramsey's Noweb.
```

21d ⟨*boilerplate 4* 21d⟩≡ (4–6 18c 19)

```
Last commit $Date: Fri Oct 13 20:23:21 2023 +0000 $
```

8 Roadmap

It was suggested in a general way here https://news.ycombinator.com/item?id=35960743 that it would be useful for a LitProg syntax highlighter to easily gray out code chunks, or documentation chunks. Consider for a future release.

While I'm at it, folding should be really easy and would make a great addition.

9 Notes on packagemaking

These are notes on how to distribute a Vim plugin as a package.

9.1 Source 1: Vim documentation

23 ⟨*to clean up* 23⟩≡

```
Vim Reference Manual, Chapter 26, "Repeating commands, Vim scripts and
debugging", section 6, "Creating Vim packages" and section 6, "Using Vim
packages".

Distribute as an archive, or distribute from a repository.
"An archive can be used by more users, but is harder to update to a new
version."
"A repository can usually be kept up-to-date easiy, but it requires a program
like 'git' to be available."
"You can do both, github can automatically create an archive for a release."

Directory layout example

   start/foobar/plugin/foo.vim          " always loaded, defines commands
   start/foobar/plugin/bar.vim          " always loaded, defines commands
   start/foobar/autoload/foo.vim        " loaded when foo command used
   start/foobar/doc/foo.txt             " help for foo.vim
   start/foobar/doc/tags                " help tags
   opt/fooextra/plugin/extra.vim        " optional plugin, defines commands
   opt/fooextra/autoload/extra.vim      " loaded when extra command used
   opt/fooextra/doc/extra.txt           " help for extra.vim
   opt/fooextra/doc/tags                " help tags

This allows for the user to do: >
        mkdir ~/.vim/pack/myfoobar
        cd ~/.vim/pack/myfoobar
        git clone https://github.com/you/foobar.git

Here "myfoobar" is a name that the user can choose, the only condition is that
it differs from other packages.


In your documentation you explain what the plugins do, and tell the user how
to load the optional plugin: >
        :packadd! fooextra
```

```
You could add this packadd command in one of your plugins, to be executed when
the optional plugin is needed.

Run the ‘:helptags‘ command to generate the doc/tags file.  Including this
generated file in the package means that the user can drop the package in his
pack directory and the help command works right away.  Don’t forget to re-run
the command after changing the plugin help: >
        :helptags path/start/foobar/doc
        :helptags path/opt/fooextra/doc

Vim startup supports packages, which are collections of plugins.
It looks for the "pack" directory in all the places in "packpath". It
scans "pack/*/start" for plugins. And, Vim also supports optional plugins found
in "pack/*/opt".
pack/  where to find packages
pack/a
pack/b
pack/c  named by local operator, these are installed packages
pack/a/start  Vim finds this at startup and scans it for plugins
pack/b/start  Vim finds this at startup and scans it for plugins
pack/c/start  Vim finds this at startup and scans it for plugins

pack/x/opt is for manual loading -- :packadd pkgname will load the package from
pack/x/opt/pkgname

<https://dev.to/iggredible/how-to-use-vim-packages-3gil>
The user-part of the name is for the end-user to organize packages. Organization
can be anything from a monolith such as ".vim/pack/my", to an organization by
type of package
~/.vim/pack/colors
~/.vim/pack/syntax
~/.vim/pack/objects
~/.vim/pack/plugins


There are various package manager add-ons for Vim
pathogen
vundle
dein
vim-plug
vim-update-bundles
minpac

Matvey at vim\_use at Google Groups points out the "user-part" can be thought
```

of as "manager-name" and be used to avoid conflicts between multiple plugin managers. So
~/.vim/pack/pathogen/*
~/.vim/pack/vundle/*
~/.vim/pack/dein/*
~/.vim/pack/vim-plug/*
~/.vim/pack/manual/*

Pathogen is not recommended even by the developer for new users. The startup search for "start" folders, and the :packadd search for "opt" folders, replaces Pathogen.

Vundle can install and update plugins. Its Plugin command takes a URI and automatically integrates with GitHub and with vim-scripts.org. Examples from the Vundle manual:
Plugins with a slash in the name are grabbed from GitHub.
"VundleVim/Vundle.vim" => https://github.com/VundleVim/Vundle.vim
Plugins without a slash are grabbed from vim-scripts
"ctrlp.vim" => https://github.com/vim-scripts/ctrlp.vim
This is (was) a mirror on Github of the vim-scripts site, called "vim-scraper". Vundle is described by the mirror author as an "early package manager", along with Vim Update Bundles, and obsolete because Vim scripts are now developed on Github and installable straight from source.
"Mostly it was created because vimballs are super duper unfriendly to package managers."
--- https://vim-scraper.github.io/
This does not however address auto-updating.

dein -- active development has stopped. Only bug fixes will be made.

vim-plug - works a lot like Vundle. Has a "Plug" command that can grab a plugin from Github or really any URL. Last updated very recently.

vim-update-bundles - end of life.

minpac - updates, but only supports Github URLs (and short form account/repo). From its readme:
Similar projects
There are some other plugin managers built on top of the Vim 8's packages feature.
vim-packager: written in Vim script
pack: written in Rust
infect: written in Ruby
vim-pck: written in Python
vim8-pack: written in Bash

```
volt: written in Go
autopac: modified version of minpac
plugpac.vim: thin wrapper of minpac, provides vim-plug like experience
minPlug: written in Vim script

Vim also distributes with a plugin called the "Vimball Archiver", "vimball.vim".
See :help vimball
Vimball supports installing and removing plugins that are packaged as
"vimballs", analogous to "tarballs".
It also has a MkVimball command which builds a Vimball .vba file.
There is external documentation on automating this process using make.
[[http://vim.wikia.com/wiki/Using_VimBall_with_%27Make%27]]

vim.org has a place to upload scripts and packages.
This collection does not seem to be supported by package managers.

Vim documentation includes a Getscript plugin that seems to get latest versions
of installed scripts.
```