BASHSUPPORT

bashsupport.txt Bash Support March 25 2014

Bash Support **bash-support bashsupport**

Plugin version 4.2.1
for Vim version 7.0 and above
Fritz Mehner <mehner@fh-swf.de>

Bash Support implements a bash-IDE for Vim/gVim. It is written to considerably speed up writing code in a consistent style. This is done by inserting complete statements, comments, idioms, and code snippets. Syntax checking, running a script, starting a debugger can be done with a keystroke. There are many additional hints and options which can improve speed and comfort when writing shell scripts. This plug-in can be used for Bash version 4.0.

++
++
++ The plug-in version 4.0+ is a rewriting of version 3.12.1. ++
++ The versions 4.0+ are based on a new and more powerful template system ++
++ (please see |template-support|for more information). ++
++ The template syntax has changed! ++
++
++

---

1. USAGE with GUI (gVim) **bashsupport-usage-gvim**

If the menus are not visible call them with the entry "Load Bash Support"
in the standard Tools-menu.

1.1 MENU 'Comments' **bashsupport-comments**

1.1.1 APPEND ALIGNED COMMENTS TO CONSECUTIVE LINES **bashsupport-aligned-comm**

In NORMAL MODE the menu item 'end-of-line comment' will append a comment to
the current line.
In VISUAL MODE, this item will append aligned comments to all marked lines.
Marking the 4 lines

```
x11=11;
x1111=1111;

x11111111=11111111;
```

and choosing 'end-of-line comment' will yield

```
x11=11;                                     # |
x1111=1111;                                 #

x11111111=11111111;                         #
```

The cursor position above is marked by '|' . Empty lines will be ignored.

The default starting column is 49 (= (multiple of 2,4, or 8) + 1). This can
be changed by setting a global variable in the file '.vimrc', e.g. :

```
  let g:BASH_LineEndCommColDefault    = 45
```

The starting column can also be set by the menu item 'Comments->adjust
end-of-line com.' . Just position the cursor in an arbitrary column (normal
mode; column number is shown in the Vim status line) and choose this menu
item. This setting is buffer related.
If the cursor was at the end of a line you will be asked for a column number
because this position is most likely not the desired starting column. Your
choice will be confirmed.

1.1.2 ADJUST END-OF-LINE COMMENTS **bashsupport-comm-realign**

After some copy/paste/change actions comments may be misaligned:

```
pathname=$(pwd)                    # the complete path
basename=${pathname##*/}                 # the basename
dirname=${pathname%/*}                         # the pathname
```

Realignment can be achieved with the menu item 'adjust end-of-line com.' In
normal mode the comment (if any) in the current line will be aligned to the

end-of-line comment column (see above) if possible. In visual mode the
comments in the marked block will be aligned:

```
 pathname=$(pwd)                                 # the complete path
 basename=${pathname##*/}                        # the basename
 dirname=${pathname%/*}                          # the pathname
```

Another way is to use the hotkey \cj. In normal and insert mode the current
line will be adjusted. To adjust n lines starting with the current one use
n\cj.

1.1.3 COMMENT/UNCOMMENT **bashsupport-comm-toggle**

The comment sign # can be set or removed at the beginning of the current line
or for a marked block using the menu items 'comment' (or \cc) and 'uncomment'
(or \cu). A single line needs not to be marked (normal mode, insert mode).

A marked block

```
 pathname=$(pwd)                                 # the complete path
 basename=${pathname##*/}                        # the basename
 dirname=${pathname%/*}                          # the pathname
```

will be changed into (and vice versa)

```
 #pathname=$(pwd)                                # the complete path
 #basename=${pathname##*/}                       # the basename
 #dirname=${pathname%/*}                         # the pathname
```

1.1.4 FRAME COMMENTS, FILE HEADER, ... **bashsupport-comm-templates**

Frame comments, function descriptions and file header comments are read as
templates from the appropriate files (see |bashsupport-templates|).

1.1.5 COMMENT/UNCOMMENT WITH echo **bashsupport-comm-echo**

The echo-entry in the Comments-menu can be used to put one or more statements
in a sort of comment. This is usually done for testing:

```
 echo "rm -f $allfiles"
```

This can only be done in normal mode: The complete line (from the first
non-blank to the end) is framed with echo "..." ; the cursor will then be
moved to the next line. Now this line can be framed and so forth.

The remove-echo-entry removes an echo. The keyword echo, the following double
quotation mark, and the last double quotation mark in the current line will be
removed; the cursor will then be moved to the next line.

1.1.6 KEYWORD+comment **bashsupport-comm-keywords**

Preliminary end-of-line comments to document (and find again!) places where
work will be resumed shortly, like

```
 # :BUG:23.05.2013 16:43:fgm:
```

Here 'fgm' is the authors reference (see section 4. for templates and tags).
Usually these comments are not meant for the final documentation.

1.2 MENU 'Statements' **bashsupport-statements**

1.2.1 NORMAL MODE, INSERT MODE. **bashsupport-stat-norm-ins**

An empty statement will be inserted and properly indented. The item 'if'
will insert an if-statement:

```
 if | ; then
 fi
```

The statement will be indented.

1.2.2 VISUAL MODE **bashsupport-stat-visual**

The highlighted area

```
 xxxxx
 xxxxx
```

can be surrounded by one of the following statements ('|'
marks the cursor position after insertion):

```
  +--------------------------+----------------------------------------+
  |                          |                                        |
  |  for | in ; do           |  for (( CNTR=0; CNTR<|; CNTR++ )); do |
  |    xxxxx                 |    xxxxx                               |
  |    xxxxx                 |    xxxxx                               |
  |  done                    |  done                                  |
  |                          |                                        |
  +--------------------------+----------------------------------------+
  |                          |                                        |
  |  if | ; then             |  if | ; then                           |
  |    xxxxx                 |    xxxxx                               |
  |    xxxxx                 |    xxxxx                               |
  |  fi                      |  else                                  |
  |                          |                                        |
  |                          |  fi                                    |
  |                          |                                        |
  +--------------------------+----------------------------------------+
  |                          |                                        |
  |  select | in ; do        |  until | ; do                          |
  |    xxxxx                 |    xxxxx                               |
  |    xxxxx                 |    xxxxx                               |
  |  done                    |  done                                  |
  |                          |                                        |
  +--------------------------+----------------------------------------+
  |                          |                                        |
  |  while | ; do            |                                        |
  |    xxxxx                 |                                        |
  |    xxxxx                 |                                        |
  |  done                    |                                        |
  |                          |                                        |
  +--------------------------+----------------------------------------+
  |                                                                   |
  |  fname ()                                                         |
  |  {                                                                |
  |    xxxxx                                                          |
  |    xxxxx                                                          |
  |  }  # ----------  end of function fname  ----------               |
  |                                                                   |
  +-------------------------------------------------------------------+
```

The whole statement will be indented after insertion.

1.3 MENUS 'Tests' ... 'Shopts' **bashsupport-tests**

Insert tests, parameter substitutions, ... I/O-redirections.

1.4 MENU 'PatternMatching' **bashsupport-pattern**

This menu can be used to compose globs and regular expressions with a few
keystrokes. In normal and insert mode the items insert the shown constructs
after the cursor. The first group ("zero or more" ... "anything expect") can
enclose a marked text. The marked text

 xxxxx

will be changed into

 *(xxxxx|)

by the first choice.

1.5 MENU 'I/O-Redir' **bashsupport-io-redir**

This menu can be used to insert I/O redirectors. The cursor will be positioned
at the obvious or most likely continuation point.
The item 'here-document' has a visual mode. A few marked lines will be
surrounded by the necessary statements, e.g.

```
 << EOF
 xxxxxxxxxxx
 xxxxxxxxxxx
 EOF
 # ===== end of here-document =====
```

1.6 MENU 'SNIPPETS' **bashsupport-snippets**

1.6.1 Code Snippets **bashsupport-codesnippets**

Code snippets are pieces of code which are kept in separate files in a special
directory. File names are used to identify the snippets. The default snippet
directory is '$HOME/.vim/bash-support/codesnippets'. Snippets are managed
with the 4 items

```
 'read  code snippet'
 'view  code snippet'
 'write code snippet'
 'edit  code snippet'
```

from the Snipptes submenu.

Creating a new snippet:

When nothing is marked, 'write code snippet' will write the whole buffer
to a snippet file, otherwise the marked area will be written to a file.

Insert a snippet:

Select the appropriate file from the snippet directory ('read code snippet').
The inserted lines will be indented.

Indentation / no indentation

Code snippets are normally indented after insertion. To suppress indentation
add the file extension "ni" or "noindent" to the snippet file name, e.g.

```
  parameter_handling.noindent
```

Snippet browser

Under a GUI a file requester will be put up. Without GUI the filename will be
read from the command line. You can change this behavior by setting a global
variable in your $HOME/.vimrc :

```
 let g:BASH_GuiSnippetBrowser = 'commandline'
```

The default value is 'gui'.

1.6.2 Edit Templates **bashsupport-templates-edit**

Nearly all menu items insert code snippets or comments. All of these are
contained within template files and can be changed by the user to meet their
requirements (see|bashsupport-templates|on how to use the template system).

The menu item 'edit templates' opens the main template file in a local plug-in
installation. This is usually the file
'$HOME/.vim/bash-support/templates/Templates'. There may be dependent files
loaded from the main file. Now change whatever file you want, save it, and
click on the menu item 'reread templates' to read in the file(s) and to
rebuild the internal representation of the templates. If menu items, hotkeys,
or shortcuts have been added or changed these changes will be applied with the
next start of Vim.

1.7 MENU 'Run' **bashsupport-run**

1.7.1 Save and run **bashsupport-run-script**

Save the current buffer to his file and run this file. The command line, the
applied parameters and the shell return code are shown in the bottom line.

 * In NORMAL MODE the whole buffer will be executed.
 * In VISUAL MODE only the selected lines will be executed.

There are three output destinations (see|bashsupport-output|). If the output
destination is VIM runtime errors are caught in an error window allowing
quickfix commands to be used (see|quickfix|).

The shell used can be set in the file '.vimrc', e.g. :

```
  let g:BASH_Executable                = '/bin/zsh'
```

The default is the value of $SHELL (Linux/U**X) or 'bash.exe' (Windows).

1.7.2 Script command line arguments **bashsupport-cmdline-args**

The item 'script cmd. line arg.' calls an input dialog which asks for command
line arguments. These arguments are forwarded to the script which is run by
the 'run' item. The arguments are kept until you change them. For file names
tab-expansion will work. The ex command

```
  :BashScriptArguments ...
```

can also be used to set the command line arguments for the current script.

The arguments belong to the current buffer (that is, each buffer can have its
own arguments). The input dialog has a history.

If the buffer gets a new name with "save as" the arguments will now belong to
the buffer with the new name.

1.7.3 Bash command line arguments **bashsupport-bash-cmdline-args**

The menu item 'Bash com. line arg.' calls an input dialog which asks for
command line options for the the Bash shell running the script in the actual
buffer (like -x or -v; see the Bash manual for possible options). These
arguments are forwarded to the invocation of Bash which is run by the 'run'
entry. The arguments are kept until you change them.

The arguments belong to the current buffer (that is, each buffer can have its
own arguments).
If the buffer gets a new name with "save as" the arguments will now belong to
the buffer with the new name.

1.7.4 Save and check syntax **bashsupport-syntax-check**

Save the current buffer to this file and run this file with the -n flag
(Bash: read commands but do not execute them; same for dash, ksh, zsh).
Errors are listed in an error window; now the quickfix commands can be used.

1.7.4.1 ERROR FORMAT **bashsupport-errorformat**

The format used to parse runtime errors handle bash, dash, ksh, and zsh
errors. This format is appropriate for most locales including 'C' and
'POSIX'.

1.7.4.2 SYNTAX CHECK OPTIONS **bashsupport-syntax-check-options**

The syntax check can be influenced by shopt options. Options which shall
always take effect can be declared in '.vimrc' , e.g.

```
  let g:BASH_SyntaxCheckOptionsGlob  = "-O extglob"
```

These options can be augmented or canceled using the menu entry 'syntax check
options' typing for instance

```
  +O extglob -O nocaseglob
```

after the prompt. The options entered this way are buffer related. The global
options set in '.vimrc' and the buffer related options are checked.

1.7.5 START DEBUGGER **bashsupport-debugger**

Start the debugger 'bashdb' or the frontend 'ddd' from the menu entry
Run->debug (GUI), with hotkey \rd or F9. Your version of the bash must be
prepared for debugging and the debugger must be installed (see
http://bashdb.sourceforge.net/).

(1) Using bashdb
When using gvim or vim running under a GUI the debugger will be started in an
independent xterm. This is done because the shell integration in gvim has
deficiencies (see also :h shell-window).

When using vim from a console terminal the debugger will be started as

```
  :!xterm <xterm defaults>  -e bashdb -- <script> <arguments> &
```

The debugger now will be run inside vim.

(2) Using ddd
The frontend ddd can be started via the menu or the hotkeys instead of bashdb
as described above. The preference can be set with the global variable
g:BASH_Debugger (possible values: 'term', 'ddd') in '.vimrc' :

```
  let g:BASH_Debugger     = 'ddd'
```

The default is 'term'.

In all cases the command line arguments from the argument setting
(|bashsupport-cmdline-args|) are passed to the debugger.

1.7.6 Hardcopy **bashsupport-hardcopy**

Generates a PostScript file from the whole buffer or from a marked region.
The hardcopy goes to the current working directory. If the buffer contains
documentation or other material from non-writable directories the hardcopy
goes to the HOME directory. The output destination will be shown in a message.

The print header contains date and time for the current locale. The definition
used is

```
  let s:BASH_Printheader = "%<%f%h%m%<  %=%{strftime('%x %X')}     Page %N"
```

The current locale can be overwritten by changing the language, e.g.

```
  :language C
```

or by setting a global variable in the file '.vimrc', e.g. :

```
  let g:BASH_Printheader = "%<%f%h%m%<  %=%{strftime('%x %X')}     SEITE %N"
```

See :h printheader and :h strftime() for more details.

1.7.7 Xterm size **bashsupport-xterm**

The size of the xterm (see below) can be changes for the current session. The
size has to be specified as COLUMNS LINES (e.g. 96 32).

1.7.8 Output redirection **bashsupport-output**

The last menu entry 'output: ... ' has 3 states:
  'output: VIM->buffer->xterm'
  'output: BUFFER->xterm->vim'
  'output: XTERM->vim->buffer'
The first state (upper-case) is the current one.

Target VIM

The script is run from the command line like ":!${SHELL} % arguments".
This is suitable for scripts with dialog elements and few lines of output.

When a script is started this way errors and warnings (if any) are caught in
an error window allowing quickfix commands to be used (see|quickfix|).

Target BUFFER

The shell output will be displayed in a window with name "Bash-Output". This
buffer and its content will disappear when the window is closed. It is not
writable and it has no file. The content could be saved with "save as".
If the output fits completely into this window the cursor will stay in the
script window otherwise the cursor will be set into the output window (you may
want to scroll).
When the script is run again and the output buffer is still open it will be
reused.

The buffer will not be opened if the script does not produce any output.
This is for convenience; you do not have to close an empty buffer.

* This is suitable for scripts without dialog elements and many lines of output
* (e.g. from options like xtrace). Use Vim as pager (scroll, jump, search with
* regular expressions, ..)

Target XTERM

The script will be run in a xterm-window. A wrapper script will ask you to
close this window with Return or <C-D> (bash).
The wrapper shows the complete command line and the return code of the script.

* This is suitable for scripts with dialog elements and many lines of output.
* The xterm is scrollable and independent from the editor window.

Appearance of the xterm

The appearance of the xterm can be controlled by the global variable
g:BASH_XtermDefaults. The assignment

```
  let g:BASH_XtermDefaults = "-fa courier -fs 10 -geometry 96x32"
```

placed in '.vimrc' would override the defaults. The defaults are

```
  "-fa courier -fs 12 -geometry 80x24"
```

FreeType font 'courier', FreeType font size 12 point, window width 80
characters, window height 24 lines.

The default output method is VIM. The preferred output method can be selected
in '.vimrc' by the global variable g:BASH_OutputGvim, e.g.

```
  let g:BASH_OutputGvim            = "xterm"
```

The methods are "vim", "buffer" and "xterm".

1.8 MENU 'Help' **bashsupport-help**

Item 'Bash manual'

Open the Bash manual.

Item 'help (Bash builtins)'

Look up Bash help for the word under the cursor. If there is no word under the
cursor you will be asked for the name of a builtin. The tab expansion can be
used.

Item 'manual (utilities)'

Display the manual for the word under the cursor. If there is more than one
hit a list of possibilities will be displayed to choose from.

If there is no word under the cursor you will be asked for the name of a
command line utility. In this case the command completion is on while entering
a name.
An interface to the on-line reference manuals must be installed (usually
man(1) for Linux/Unix, see|bashsupport-custom-variables|).

Item 'bash-support'

Display this help text if it was properly added with ':helptags'.

---

2. USAGE without GUI (Vim) **bashsupport-usage-vim**

The frequently used constructs can be inserted with key mappings. The
mappings are also described in the document 'bash-hotkeys.pdf' (reference
card, part of this package).

 * All mappings are filetype specific: they are only
 * defined for buffers with filetype 'sh' to minimize conflicts with mappings
 * from other plug-ins.

Hint: Typing speed matters. The combination of a leader ('\') and the
following character(s) will only be recognized for a short time.

Some mappings can be used with line range. In normal mode

 \cl

appends a end-of-line comment to the current line, whereas

 4\cl

appends end-of-line comments to the 4 lines starting with the current line.

Legend: (i) insert mode, (n) normal mode, (v) visual mode
         [n] range

  -- Submenus --

     \bps   parameter substitution (list)   (n, i)
     \bsp   special parameters (list)   (n, i)
     \ben   environment (list)   (n, i)
     \bbu   builtin (list)   (n, i)
     \bse   set options (list)   (n, i)
     \bso   shopts (list)   (n, i)

  -- Comments --

  [n]\cl   line end comment   (n, i, v)
  [n]\cj   adjust end-of-line comments   (n, i, v)
     \cs   set end-of-line comment column   (n)
  [n]\cc   code -> comment   (n, i, v)
  [n]\cu   uncomment code   (n, i, v)
     \cfr   frame comment   (n, i)
     \cfu   function description   (n, i)
     \ch   file header   (n, i)
     \cd   date   (n, i, v)
     \ct   date & time   (n, i, v)
     \css   script sections   (n, i)
     \ckc   keyword comments   (n, i)
     \cma   plug-in macros   (n, i)
     \ce   echo "..."   (n, i)
     \cr   remove echo "..."   (n, i)

  -- Statements --

     \sc   case in ... esac   (n, i)
     \sei   elif then   (n, i)
     \sf   for in do done   (n, i, v)
     \sfo   for ((...)) do done   (n, i, v)
     \si   if then fi   (n, i, v)
     \sie   if then else fi   (n, i, v)
     \ss   select in do done   (n, i, v)
     \su   until do done   (n, i, v)
     \sw   while do done   (n, i, v)
     \sfu   function   (n, i, v)
     \se   echo -e "..."   (n, i, v)
     \sp   printf "..."   (n, i, v)
     \sa   array element, ${.[.]}   (n, i, v)
     \saa   array elements (all), ${.[@]}   (n, i, v)
     \sas   array elements (string), ${.[*]}   (n, i, v)
     \ssa   subarray, ${.[@]::}   (n, i, v)
     \san   no. of array elements, ${#.[@]}   (n, i, v)

```
   \sai    array indices, ${!.[*]}            (n, i, v)

  -- Test ------------------------------------------------

   \ta     arithmetic tests                   (n, i)
   \tfp    file permission                    (n, i)
   \tft    file types                         (n, i)
   \tfc    file characteristics               (n, i)
   \ts     string comparison                  (n, i)
   \toe    option is enabled                  (n, i)
   \tvs    variables has been set             (n, i)
   \tfd    file descriptor is open            (n, i)
   \tm     string matches regexp              (n, i)

  -- I/O-Redirection -------------------------------------
   \ior    IO-redirections (list)             (n, i)
   \ioh    here-document                      (n, i)

  -- Pattern Matching ------------------------------------

   \pzo    zero or more, ?( | )               (n, i)
   \pzm    zero or more, *( | )               (n, i)
   \pom    one or more, +( | )                (n, i)
   \peo    exactly one, @( | )                (n, i)
   \pae    anything except, !( | )            (n, i)
   \ppc    POSIX classes                      (n, i)
   \pbr    BASH_REMATCH                       (n, i)

  -- Snippets --------------------------------------------

   \nr     read code snippet                  (n, i)
   \nv     view code snippet  (readonly)      (n, i)
   \nw     write code snippet                 (n, v, i)
   \ne     edit code snippet                  (n, i)

   \ntl    edit templates                     (n, i)
   \ntr    reread the templates               (n, i)
   \nts    switch template style              (n, i)

  -- Run -------------------------------------------------

  [n]\rr     update file, run script            (n, i)
   \ra     set script cmd. line arguments     (n, i)
   \rba    set Bash cmd. line arguments       (n, i)
   \rc     update file, check syntax          (n, i)
   \rco    syntax check options               (n, i)
   \rd     start debugger                     (n, i)
   \re     make script executable/not exec.(*) (in )
   \rh     hardcopy buffer to FILENAME.ps     (n, i)
   \rs     plug-in settings                   (n, i)
   \rt     set xterm size (*)                 (n, i)
   \ro     change output destination          (n, i)

  -- Bash help -------------------------------------------

   \hb     Displays the Bash manual           (n, i)
   \hh     Displays help for the builtin      (n, i)
           under the cursor (Bash help).
           The tab expansion is active.
   \hm     displays the manual for the        (n, i)
           Bash command under the cursor
           The tab expansion is active.
   \hbs    Displays the Vim help page for     (n, i)
           this plug-in.

(*) Linux/UNIX only

File 'bash-hotkeys.pdf' contains a reference card for these key mappings.
Multiline inserts and code snippets will be indented after insertion.
```

Changing the default map leader '\'

The map leader can be changed by the user by setting a global variable in the
file .vimrc

```
 let g:BASH_MapLeader  = ','
```

The map leader is now a comma. The 'line end comment' command is now defined
as ',cl'. This setting will be used as a so called local leader and influences
only files with filetype 'sh'.

The configured mapleader can also be used in the ftplugin, by calling the
functions **Bash_SetMapLeader()** and **Bash_ResetMapLeader()** . The maps created
between the two calls will use |g:BASH_MapLeader| as the |<LocalLeader>|:

```
  call Bash_SetMapLeader ()

  map  <buffer>  <LocalLeader>eg  :echo "Example Map :)"<CR>

  call Bash_ResetMapLeader ()
```

---

3. HOT KEYS **bashsupport-hotkeys**

The following hot keys are defined in NORMAL, VISUAL and INSERT MODE:

```
    Ctrl-F9   run script
     Alt-F9   run syntax check
   Shift-F9   command line arguments (for the current buffer)
         F9   start debugger (bashdb)
```

See |bashsupport-usage-vim| for more hotkeys.

4.0 CUSTOMIZATION **bashsupport-customization**

4.1 FILES **bashsupport-custom-files**

README.bashsupport	Release notes, installation description.
plugin/bash-support.vim	The Bash plug-in for Vim/gVim.
bash-support/scripts/wrapper.sh	A wrapper script for the use of an xterm.
doc/bashsupport.txt	The help file for the local online help.
bash-support/codesnippets/*	Some code snippets as a starting point.
bash-support/templates/*	Bash template files (see |bashsupport-comm-templates|).
bash-support/wordlists/*	Additional word lists (dictionaries).

```
  ------------------------------------------------------------
  The following files and extensions are for convenience only.
  bash-support.vim will work without them.
  ------------------------------------------------------------
```

bash-support/rc/costumization.bashrc	Additional settings I use in .bashrc: set the prompt P2, P3, P4 (for debugging).
bash-support/rc/costumization.vimrc	Additional settings I use in '.vimrc': incremental search, tabstop, hot keys, font, use of dictionaries, ... The file is commented. Append it to your '.vimrc'

```
                                    if you like.

bash-support/rc/costumization.gvimrc  Additional settings I use in '.gvimrc': hot keys,
                                      mouse settings, ...  The file is commented.
                                      Append it to your '.gvimrc' if you like.
bash-support/rc/sh.vim                Suggestions for additional maps.

bash-support/doc/*                    Hotkey reference card (PDF), changelog.
```

4.2 GLOBAL VARIABLES **bashsupport-custom-variables**

Several global variables are checked by the script to customize it:

---

global variable	default value
g:BASH_GlobalTemplateFile	root_dir.'bash-support/templates/Templates'
g:BASH_LocalTemplateFile	$HOME.'/.vim/bash-support/templates/Templates'
g:BASH_CodeSnippets	$HOME.'/.vim/bash-support/codesnippets' (Linux/U**X) $VIM.'\vimfiles\bash-support/codesnippets/' (Windows)
g:BASH_LoadMenus	'yes'
g:BASH_CreateMenusDelayed	'no'
g:BASH_Dictionary_File	$HOME."/.vim/bash-support/wordlists/bash.list"
g:BASH_RootMenu	'&Bash.'
g:BASH_GuiSnippetBrowser	'gui'
g:BASH_OutputGvim	'vim' (Linux/U**X) 'xterm' (Windows)
g:BASH_XtermDefaults	'-fa courier -fs 12 -geometry 80x24'
g:BASH_Debugger	'term'
g:BASH_LineEndCommColDefault	49
g:BASH_SyntaxCheckOptionsGlob	''
g:BASH_Printheader	'%<%f%h%m%< %=%{strftime('%x %X')} Page %N'
g:BASH_InsertFileHeader	'yes'
g:BASH_Executable	$SHELL (Linux/U**X) 'bash.exe' (Windows)
g:BASH_ManualReader	'man' (Linux/U**X) 'man.exe' (Windows)
g:BASH_MapLeader	'\'
g:BASH_Errorformat	'%f:\ line\ %l:\ %m'
g:BASH_AlsoBash	''

---

1. group: Defines the text which will be inserted for the tags when a
   template is read in (see |bashsupport-templates|).

   - g:BASH_GlobalTemplateFile : sets the global template file (see|bashsupport-templates|)
   - g:BASH_LocalTemplateFile : sets the local template file (see|bashsupport-templates|)

2. group:
   - g:BASH_CodeSnippets : The name of the code snippet directory (see |bashsupport-snippets|).
   - g:BASH_LoadMenus : Load menus and mappings (yes/no) at start up.
   - g:BASH_CreateMenusDelayed : Load menus only with filetype 'sh'
   - g:BASH_Dictionary_File : Path and file name of the Bash word list used for dictionary completion (see |bashsupport-dictionary|).
   - g:BASH_RootMenu : Name of the root menu item of this plug-in (see |bashsupport-custom-root|).
   - g:BASH_GuiSnippetBrowser : code snippet browser: 'gui', 'commandline'

3. group:
   - g:BASH_OutputGvim : Target for a script output

```
                                    (see |bashsupport-output|).
        g:BASH_XtermDefaults        : The xterm defaults (see |bashsupport-xterm|).
        g:BASH_Debugger             : the debugger called by F9 (term, ddd).
        g:BASH_LineEndCommColDefault  : default starting column for line end comments
        g:BASH_SyntaxCheckOptionsGlob : shopt options used with syntax checking
        g:BASH_Printheader          : hardcopy header format
        g:BASH_InsertFileHeader     : suppress file header comment for new files
        g:BASH_Executable           : the shell used
        g:BASH_ManualReader         : the interface to the on-line manuals
        g:BASH_MapLeader            : the map leader for hotkeys
                                    (see|bashsupport-usage-vim|)
        g:BASH_Errorformat          : errorforamat used to parse runtime errors
        g:BASH_AlsoBash             : filename patterns considered as Bash files
                                    (see |bashsupport-also-bash|)

To override the defaults add appropriate assignments in '.vimrc'.
```

4.3 THE ROOT MENU **bashsupport-custom-root**

```
The variable g:BASH_RootMenu, if set (in '.vimrc' or in '.gvimrc'), gives the
name of the single gVim root menu entry in which the Bash submenus will be
put.  The default is

 '&Bash'

If you want to set the plug-in root menu into another menu, e.g. 'Plugin',
this is done by the following line in '.vimrc'

  let g:BASH_RootMenu = "&Plugin.&Bash"
```

4.4 System-wide installation **bashsupport-system-wide**

```
A system-wide installation (one installation for all users) is done as
follows.

As *** SUPERUSER *** :

(1) Find the Vim installation directory.
The Vim ex command ':echo $VIM' gives '/usr/local/share/vim' or something like
that. Beyond this directory you will find the Vim installation,  e.g. in
'/usr/local/share/vim/vim73' if Vim version 7.3 has been installed.

(2) Create a new subdirectory 'vimfiles', e.g. '/usr/local/share/vim/vimfiles'.

(3) Install Bash Support
Copy the archive 'bash-support.zip' to this new directory and unpack it:

  unzip bash-support.zip

(4) Generate the help tags:

  :helptags $VIM/vimfiles/doc

SPECIAL CASES. Some Linux distributions use non-standard names for Vim
directories. SUSE has a directory '/usr/share/vim/site' to put plug-ins in.
These directories will not be found automatically.  After installing the
plug-in below '/usr/share/vim/site' the use of the templates will be enabled by
the following line in '~/.vimrc':

  let g:BASH_GlobalTemplateFile = '/usr/share/vim/site/bash-support/templates/Templates'


As ****** USER ****** :
```

The plug-in tries to create a minimal template file 'Templates' (and the
necessary directory '$HOME/.vim/bash-support/templates') when the first buffer
with filetype 'vim' will be opened. You should edit this file to personalize
some macros.

Create your private snippet directory:

```
  mkdir --parents  $HOME/.vim/bash-support/codesnippets
```

You may want to copy the snippets coming with this plug-in (in
$VIM/vimfiles/bash-support/codesnippets) into the new directory or to set a
link to the global directory.

---

5. TEMPLATE FILES AND TAGS **bashsupport-templates**

5.1 TEMPLATE FILES **bashsupport-templates-files**

Nearly all menu items insert code snippets or comments. All of these are
contained within template files and can be changed by the user to meet their
requirements. The menu shortcuts (e.g. 'c' for the Comments menu) and the
menu item hotkeys (e.g. '\ct' insert date and time) are also defined in the
templates.
The template engine comes as a separate plug-in contributed by Wolfgang Mehner.
This section is a short introduction to this template system. Please see
|templatesupport| for more information.

The master template file is '$HOME/.vim/bash-support/templates/Templates' for
a user installation and '$VIM/vimfiles/bash-support/templates/Templates' for
a system-wide installation (see |bashsupport-system-wide|).

The master template file starts with a macro section followed by templates for
single menu items or better by including other template files grouping the
templates according to the menu structure of this plug-in. The master file
usually looks like this (my settings as an example):

```
 § ================================================================
 §  User Macros
 § ================================================================

 SetMacro( 'AUTHOR',      'Dr. Fritz Mehner' )
 SetMacro( 'AUTHORREF',   'fgm' )
 SetMacro( 'COMPANY',     '' )
 SetMacro( 'COPYRIGHT',   'Copyright (c) |YEAR|, |AUTHOR|' )
 SetMacro( 'EMAIL',       'mehner.fritz@fh-swf.de' )
 SetMacro( 'LICENSE',     'GNU General Public License' )
 SetMacro( 'ORGANIZATION','FH SÃ¼dwestfalen, Iserlohn, Germany' )

 SetFormat( 'DATE', '%x' )
 SetFormat( 'TIME', '%H:%M' )
 SetFormat( 'YEAR', '%Y' )

 SetStyle( 'default' )

 § ================================================================
 §  File Includes and Shortcuts
 § ================================================================

 MenuShortcut( 'Comments',        'c' )
 MenuShortcut( 'Statements',      's' )
 MenuShortcut( 'Tests',           't' )
 MenuShortcut( 'ParamSub',        'p' )
 MenuShortcut( 'PatternMatching', 'p' )
 MenuShortcut( 'IO-Redir',        'i' )

 IncludeFile( 'comments.templates'        )
```

```
 IncludeFile( 'statements.templates'      )
 IncludeFile( 'tests.templates'           )
 IncludeFile( 'paramsub.templates'        )
 IncludeFile( 'specialparams.templates'   )
 IncludeFile( 'environment.templates'     )
 IncludeFile( 'builtins.templates'        )
 IncludeFile( 'set.templates'             )
 IncludeFile( 'shelloptions.templates'    )
 IncludeFile( 'io-redirection.templates'  )
 IncludeFile( 'patternmatching.templates' )
```

Lines starting with a paragraph sign are comments. The section starting with

```
  SetMacro( 'AUTHOR',      'Dr. Fritz Mehner' )
```

assigns values to predefined tags (macros). Arbitrary user-defined macros are possible. The macro name must follows the rules for a C language identifier: first character letter or underscore; case matters; digits are allowed beginning with the second character.

The statement

```
  IncludeFile( 'comments.templates' )
```

includes the templates from the file 'comments.templates' (in the same directory). An absolute path would also be possible. The statement

```
  MenuShortcut( 'Comments',      'c' )
```

sets 'c' as the shortcut for the Comments menu.

5.2 Macros **bashsupport-templates-macros**

The following macro names are predefined. The first group of macros is used to personalize templates.

PREDEFINED MACROS FOR PERSONALIZATION, DEFAULT VALUE is ''
|AUTHOR|
|AUTHORREF|
|COMPANY|
|COPYRIGHT|
|EMAIL|
|LICENSE|
|ORGANIZATION|

PREDEFINED MACROS	DEFAULT VALUE
|BASENAME|	filename without path and suffix
|DATE|	the preferred date representation for the current locale without the time
|FILENAME|	filename without path
|PATH|	path without filename
|SUFFIX|	filename suffix
|TIME|	the preferred time representation for the current locale without the date and the time zone or name or abbreviation
|YEAR|	the year as a decimal number including the century

PREDEFINED TAGS USED IN TEMPLATES	
<CURSOR>	The cursor position after insertion of a template.
<+text+>,<-text-> {+text+},{-text-}	See |bashsupport-templates-jumptags|.

```
 <SPLIT>           The split point when inserting in visual mode
                   (see|bashsupport-templates|)

A dependent template file can start with its own command section. There is no
need to have all user defined macros in the master file.
```

5.2.1 User defined formats for date and time **bashsupport-templates-date**

```
The format for |DATE| ,|TIME| , and|YEAR| can be set by the user. The
defaults are
    |DATE|        '%x'
    |TIME|        '%X'
    |YEAR|        '%Y'
See the manual page of the C function strftime() for the format.  The accepted
format depends on your system, thus this is not portable!  The maximum length
of the result is 80 characters.

User defined formats can be set using the following function calls in the
master template file is '$HOME/.vim/bash-support/templates/Templates',  e.g.

  SetFormat( 'DATE', '%D'      )
  SetFormat( 'TIME', '%H:%M'   )
  SetFormat( 'YEAR', 'year %Y' )
```

5.3 Templates **bashsupport-templates-names**

5.3.1 Template Definition **bashsupport-templates-definition**

```
The template behind a menu item is identified its name. The first part
of the name identifies the menu name, the second part identifies the item.
A template definition starts with a template header with the following syntax:

  == menu_name.template_name == options ==

The options are described here: |template-support-options|.
```

5.3.2 The jump tags <+text+> etc. **bashsupport-templates-jumptags**

```
There are four jump tag types which can be used as jump targets in templates:

 <+text+>   Can be jumped to by hitting Ctrl-j.
 {+text+}   Same as <+text+>. Used in cases where indentation gives unwanted
            results with the first one.

 <-text->   Same as the two above. Will be removed if the template is used
 {-text-}   in visual mode.

The text inside the brackets is userdefined and can be empty. The text can be
composed from letters (uppercase and lowercase), digits, and underscores.
After the insertion of an template these jump targets will be highlighted.
```

5.3.3 Command Ctrl-j **bashsupport-Ctrl-j**

```
Use the command Ctrl-j to jump to the next target. The target will be removed
and the mode will switched to insertion. Ctrl-j works in normal and in insert
mode.  The template for an if-else-statement can be written as follows:

 == Statements.if-else == map:sie, shortcut:i  ==
 if <CURSOR> ; then
 <SPLIT><-IF_PART->
 else
  <-ELSE_PART->
 fi

The cursor will be set as shown. When the condition is specified a Ctrl-j let
you jump to the target <-IF_PART-> and deletes it.  When the block is written
```

a Ctrl-j leads you to the else-part. The target <-ELSE_PART-> disappears and
you can type on.

5.4 ADDITIONAL FILENAMES WITH FILETYPE 'sh' **bashsupport-also-bash**

The standard extension for shell script files used by this plug-in is ".sh".
If you want to have other filenames recognized and treated as shell files
set the global variable g:BASH_AlsoBash in your '.vimrc' :

```
 let g:BASH_AlsoBash = [ '.SlackBuild' , 'rc.' ]
```

This is a Vim List. Please quote the entries and separate them with commas.
Do not include the default extension '*.sh'.

---

6. BASH DICTIONARY **bashsupport-dictionary**

The file 'bash.list' contains words used as dictionary for automatic word
completion. This feature is enabled by default. The default word list is

```
  $HOME/.vim/bash-support/wordlists/bash.list
```

If you want to use an additional list 'MyBash.List' put the following line
into $HOME/.vimrc :

```
  let g:BASH_Dictionary_File   =  "$HOME/.vim/bash-support/wordlists/bash.list,".
                               \   "$HOME/.vim/bash-support/wordlists/MyBash.List"
```

The right side is a comma separated list of files. Note the point at the end
of the first line (string concatenation) and the backslash in front of the
second line (continuation line).
You can use Vim's dictionary feature CTRL-X, CTRL-K (and CTRL-P, CTRL-N).

---

7. ADDITIONAL MAPPINGS **bashsupport-ad-mappings**

Some suggestions are in 'bash-support/rc/sh.vim'. This is a filetype plug-in
and would go to '$HOME/.vim/ftplugin/'.

---

8. WINDOWS PARTICULARITIES **bashsupport-windows**

For a user installation the plug-in should go into the directory structure
below

```
  $HOME/vimfiles/
```

for a system installation below

```
 $VIM/vimfiles/
```

The values of the two variables can be found from inside Vim:

```
   :echo $VIM
```

or

```
   :echo $HOME
```

The configuration files for a user are

  $HOME/_vimrc   and $HOME/_gvimrc

for the system

  $VIM/_vimrc   and $VIM/_gvimrc

A Bash shell is not a part of a Windows system and has to be installed in
addition. This plug-in assumes that the shell port bash.exe
(http://www.cygwin.com) is present and the that command line utilities you
want to use can be reached via your path variable. This shell can be changed
to xyz-shell.exe by setting the following line into the file _vimrc:

  let g:BASH_Executable = 'xyz-shell.exe'

The run-menu and the corresponding hotkeys are restricted. Please see
the document 'bash-hotkeys.pdf' for this restrictions.

The file format is switches to 'unix' on entering a buffer of type 'sh'.
The filetype can be changed by setting a global variable in file _vimrc:

  let g:BASH_FileFormat = 'something_else'

CYGWIN

Executing scripts with DOS style pathnames causes the error "MS-DOS style path
detected". To turn these messages off add the CYGWIN environment variable
CYGWIN with the value "nodosfilewarning".

---

9. TROUBLESHOOTING **bashsupport-troubleshooting**

---

* I do not see any new main menu item.
  - Was the archive extracted into the right directory?

* How can I see what was loaded?
  - Use ':scriptnames' from the Vim command line.

* No main menu item.
  - Loading of plug-in files must be enabled. If not use

     :filetype plugin on

    This is the minimal content of the file '$HOME/.vimrc'. Create one if
    there is none, or better use customization.vimrc.

* Most key mappings do not work.
  - They are defined in a filetype plugin in '$HOME/.vim/ftplugin/'. Use
    ':filetype' to check if filetype plug-ins are enabled. If not, add the line

      filetype plugin on

    to the file '~/.vimrc'.

* Some hotkeys do not work.
  - The hotkeys might be in use by your graphical desktop environment. Under
    KDE Ctrl-F9 is the hotkey which let you switch to the 9. desktop. The key
    settings can usually be redefined.

---

10. Release Notes **bashsupport-release-notes**

See file 'README.bashsupport'.

---

vim:tw=78:noet:ts=2:ft=help:norl:

Generated by vim2html on Mo 21. Apr 18:16:15 CEST 2014