```
Standard plugins ~
|pi_getscript.txt| Downloading latest version of Vim scripts
|pi_gzip.txt|      Reading and writing compressed files
|pi_logipat.txt|   Logical operators on patterns
|pi_netrw.txt|     Reading and writing files over a network
|pi_paren.txt|     Highlight matching parens
|pi_tar.txt|       Tar file explorer
|pi_vimball.txt|   Create a self-installing Vim script
|pi_zip.txt|       Zip archive explorer

==============================================================================
*pi_getscript.txt*  For Vim version 7.0.  Last change: 2017 Aug 01
>
                GETSCRIPT REFERENCE MANUAL  by Charles E. Campbell
<
Authors:  Charles E. Campbell  <NdrOchip@ScampbellPfamilyA.Mbiz>
          (remove NOSPAM from the email address)
                                                *GetLatestVimScripts-copyright*
Copyright: (c) 2004-2012 by Charles E. Campbell *glvs-copyright*
        The VIM LICENSE (see |copyright|) applies to the files in this
        package, including getscriptPlugin.vim, getscript.vim,
        GetLatestVimScripts.dist, and pi_getscript.txt, except use "getscript"
        instead of "Vim".  Like anything else that's free, getscript and its
        associated files are provided *as is* and comes with no warranty of
        any kind, either expressed or implied.  No guarantees of
        merchantability.  No guarantees of suitability for any purpose.  By
        using this plugin, you agree that in no event will the copyright
        holder be liable for any damages resulting from the use of this
        software. Use at your own risk!

Getscript is a plugin that simplifies retrieval of the latest versions of the
scripts that you yourself use!  Typing |:GLVS| will invoke getscript; it will
then use the <GetLatestVimScripts.dat> (see |GetLatestVimScripts_dat|) file to
get the latest versions of scripts listed therein from http://vim.sf.net/.

==============================================================================
1. Contents                             *glvs-contents* *glvs* *getscript*
                                        *GetLatestVimScripts*

        1. Contents.....................................: |glvs-contents|
        2. GetLatestVimScripts -- Getting Started.....: |glvs-install|
        3. GetLatestVimScripts Usage....................: |glvs-usage|
        4. GetLatestVimScripts Data File................: |glvs-data|
        5. GetLatestVimScripts Friendly Plugins.........: |glvs-plugins|
        6. GetLatestVimScripts AutoInstall..............: |glvs-autoinstall|
        7. GetLatestViMScripts Options..................: |glvs-options|
        8. GetLatestVimScripts Algorithm................: |glvs-alg|
        9. GetLatestVimScripts History..................: |glvs-hist|


==============================================================================
2. GetLatestVimScripts -- Getting Started               *getscript-start*
                                                *getlatestvimscripts-install*

        VERSION FROM VIM DISTRIBUTION                   *glvs-dist-install*

Vim 7.0 does not include the GetLatestVimScripts.dist file which
serves as an example and a template.  So, you'll need to create
your own!  See |GetLatestVimScripts_dat|.

        VERSION FROM VIM SF NET                         *glvs-install*
```

```
NOTE: The last step, that of renaming/moving the GetLatestVimScripts.dist
file, is for those who have just downloaded GetLatestVimScripts.tar.bz2 for
the first time.

The GetLatestVimScripts.dist file serves as an example and a template for your
own personal list.  Feel free to remove all the scripts mentioned within it;
the "important" part of it is the first two lines.

Your computer needs to have wget or curl for GetLatestVimScripts to do its work.

        1. if compressed:  gunzip getscript.vba.gz
        2. Unix:
                vim getscript.vba
                :so %
                :q
                cd ~/.vim/GetLatest
                mv GetLatestVimScripts.dist GetLatestVimScripts.dat
                (edit GetLatestVimScripts.dat to install your own personal
                list of desired plugins -- see |GetLatestVimScripts_dat|)

        3. Windows:
                vim getscript.vba
                :so %
                :q
                cd **path-to-vimfiles**/GetLatest
                mv GetLatestVimScripts.dist GetLatestVimScripts.dat
                (edit GetLatestVimScripts.dat to install your own personal
                list of desired plugins -- see |GetLatestVimScripts_dat|)


==============================================================================
3. GetLatestVimScripts Usage                            *glvs-usage* *:GLVS*

Unless it has been defined elsewhere, >

        :GLVS

will invoke GetLatestVimScripts().  If some other plugin has defined that
command, then you may type
>
        :GetLatestVimScripts
<
The script will attempt to update and, if permitted, will automatically
install scripts from http://vim.sourceforge.net/.  To do so it will peruse a
file,
>
        .vim/GetLatest/GetLatestVimScripts.dat                  (unix)
<
or >
        ..wherever..\vimfiles\GetLatest\GetLatestVimScripts.dat   (windows)
(see |glvs-data|), and examine plugins in your [.vim|vimfiles]/plugin
directory (see |glvs-plugins|).

Scripts which have been downloaded will appear in the
~/.vim/GetLatest (unix) or ..wherever..\vimfiles\GetLatest (windows)
subdirectory.  GetLatestVimScripts will attempt to automatically
install them if you have the following line in your <.vimrc>: >

        let g:GetLatestVimScripts_allowautoinstall=1

The <GetLatestVimScripts.dat> file will be automatically be updated to
reflect the latest version of script(s) so downloaded.
```

(also see |glvs-options|)

==
4. GetLatestVimScripts Data File *getscript-data* *glvs-data*
:GetLatestVimScripts_dat

The data file <GetLatestVimScripts.dat> must have for its first two lines
the following text:
>
        ScriptID SourceID Filename

<
Following those two lines are three columns; the first two are numeric
followed by a text column. The GetLatest/GetLatestVimScripts.dist file
contains an example of such a data file. Anything following a #... is
ignored, so you may embed comments in the file.

The first number on each line gives the script's ScriptID. When you're about
to use a web browser to look at scripts on http://vim.sf.net/, just before you
click on the script's link, you'll see a line resembling

        http://vim.sourceforge.net/scripts/script.php?script_id=40

The "40" happens to be a ScriptID that GetLatestVimScripts needs to
download the associated page, and is assigned by vim.sf.net itself
during initial uploading of the plugin.

The second number on each line gives the script's SourceID. The SourceID
records the count of uploaded scripts as determined by vim.sf.net; hence it
serves to indicate "when" a script was uploaded. Setting the SourceID to 1
insures that GetLatestVimScripts will assume that the script it has is
out-of-date.

The SourceID is extracted by GetLatestVimScripts from the script's page on
vim.sf.net; whenever it is greater than the one stored in the
GetLatestVimScripts.dat file, the script will be downloaded
(see |GetLatestVimScripts_dat|).

If your script's author has included a special comment line in his/her plugin,
the plugin itself will be used by GetLatestVimScripts to build your
<GetLatestVimScripts.dat> file, including any dependencies on other scripts it
may have. As an example, consider: >

        " GetLatestVimScripts: 884 1 :AutoInstall: AutoAlign.vim

This comment line tells getscript.vim to check vimscript #884 and that the
script is automatically installable. Getscript will also use this line to
help build the GetLatestVimScripts.dat file, by including a line such as: >

        884 1 :AutoInstall: AutoAlign.vim
<
assuming that such a line isn't already in GetLatestVimScripts.dat file.
See |glvs-plugins| for more. Thus, GetLatestVimScripts thus provides a
comprehensive ability to keep your plugins up-to-date!

In summary:

  * Optionally tell getscript that it is allowed to build/append a
    GetLatestVimScripts.dat file based upon already installed plugins: >
        let g:GetLatestVimScripts_allowautoinstall=1
<
  * A line such as >

```
        " GetLatestVimScripts: 884  1 :AutoInstall: AutoAlign.vim
<   in an already-downloaded plugin constitutes the concurrence of the
    plugin author that getscript may do AutoInstall.  Not all plugins
    may be AutoInstall-able, and the plugin's author is best situated
    to know whether or not his/her plugin will AutoInstall properly.

  * A line such as >
        884 1 :AutoInstall: AutoAlign.vim
<   in your GetLatestVimScripts.dat file constitutes your permission
    to getscript to do AutoInstall.  AutoInstall requires both your
    and the plugin author's permission.  See |GetLatestVimScripts_dat|.


                                                    *GetLatestVimScripts_dat*
As an example of a <GetLatestVimScripts.dat> file:
>
    ScriptID SourceID Filename
    --------------------------
    294 1 :AutoInstall: Align.vim
    120 2 Decho.vim
     40 3 DrawIt.tar.gz
    451 4 EasyAccents.vim
    195 5 engspchk.vim
    642 6 GetLatestVimScripts.vim
    489 7 Manpageview.vim
<
Note: the first two lines are required, but essentially act as comments.


==============================================================================
5. GetLatestVimScripts Friendly Plugins *getscript-plugins* *glvs-plugins*

                (this section is for plugin authors)~

If a plugin author includes the following comment anywhere in their plugin,
GetLatestVimScripts will find it and use it to automatically build the user's
GetLatestVimScripts.dat files:
>
                                  src_id
                                    v
        " GetLatestVimScripts: ### ### yourscriptname
                                    ^
                               scriptid
<
As an author, you should include such a line in to refer to your own script
plus any additional lines describing any plugin dependencies it may have.
Same format, of course!

If your command is auto-installable (see |glvs-autoinstall|), and most scripts
are, then you may include :AutoInstall: just before "yourscriptname":
>
                                  src_id
                                    v
        " GetLatestVimScripts: ### ### :AutoInstall: yourscriptname
                                    ^
                               scriptid
<
NOTE: The :AutoInstall: feature requires both the plugin author's and~
      the user's permission to operate!~

GetLatestVimScripts commands for those scripts are then appended, if not
already present, to the user's GetLatest/GetLatestVimScripts.dat file.  It is
```

a relatively painless way to automate the acquisition of any scripts your
plugins depend upon.

Now, as an author, you probably don't want GetLatestVimScripts to download
your own scripts atop your own copy, thereby overwriting your not-yet-released
hard work. GetLatestVimScripts provides a solution for this: put
>
        0 0 yourscriptname
<
into your <GetLatestVimScripts.dat> file and GetLatestVimScripts will skip
examining the "yourscriptname" scripts for those GetLatestVimScripts comment
lines. As a result, those lines won't be inadvertently installed into your
<GetLatestVimScripts.dat> file and subsequently used to download your own
scripts. This is especially important to do if you've included the
:AutoInstall: option.

Be certain to use the same "yourscriptname" in the "0 0 yourscriptname" line
as you've used in your GetLatestVimScripts comment!

==
6. GetLatestVimScripts AutoInstall   *getscript-autoinstall*
                                                      glvs-autoinstall

GetLatestVimScripts now supports "AutoInstall". Not all scripts are
supportive of auto-install, as they may have special things you need to do to
install them (please refer to the script's "install" directions). On the
other hand, most scripts will be auto-installable.

To let GetLatestVimScripts do an autoinstall, the data file's comment field
should begin with (surrounding blanks are ignored): >

        :AutoInstall:
<
Both colons are needed, and it should begin the comment (yourscriptname)
field.

One may prevent any autoinstalling by putting the following line in your
<.vimrc>: >

        let g:GetLatestVimScripts_allowautoinstall= 0
<
With :AutoInstall: enabled, as it is by default, files which end with

        ---.tar.bz2 : decompressed & untarred in .vim/ directory
        ---.vba.bz2 : decompressed in .vim/ directory, then vimball handles it
        ---.vim.bz2 : decompressed & moved into .vim/plugin directory
        ---.tar.gz   : decompressed & untarred in .vim/ directory
        ---.vba.gz   : decompressed in .vim/ directory, then vimball handles it
        ---.vim.gz   : decompressed & moved into .vim/plugin directory
        ---.vba   : unzipped in .vim/ directory
        ---.vim   : moved to .vim/plugin directory
        ---.zip   : unzipped in .vim/ directory

and which merely need to have their components placed by the untar/gunzip or
move-to-plugin-directory process should be auto-installable. Vimballs, of
course, should always be auto-installable.

When is a script not auto-installable? Let me give an example:

        .vim/after/syntax/blockhl.vim

The <blockhl.vim> script provides block highlighting for C/C++ programs; it is
available at:

        http://vim.sourceforge.net/scripts/script.php?script_id=104

Currently, vim's after/syntax only supports by-filetype scripts (in
blockhl.vim's case, that's after/syntax/c.vim). Hence, auto-install would
possibly overwrite the current user's after/syntax/c.vim file.

In my own case, I use <aftersyntax.vim> (renamed to after/syntax/c.vim) to
allow a after/syntax/c/ directory:

        http://vim.sourceforge.net/scripts/script.php?script_id=1023

The script allows multiple syntax files to exist separately in the
after/syntax/c subdirectory. I can't bundle aftersyntax.vim in and build an
appropriate tarball for auto-install because of the potential for the
after/syntax/c.vim contained in it to overwrite a user's c.vim.

==
7. GetLatestVimScripts Options   *glvs-options*
>
        g:GetLatestVimScripts_wget
<   default= "wget"
                This variable holds the name of the command for obtaining
                scripts.
>
        g:GetLatestVimScripts_options
<   default= "-q -O"
                This variable holds the options to be used with the
                g:GetLatestVimScripts_wget command.
>
        g:GetLatestVimScripts_allowautoinstall
<   default= 1
                This variable indicates whether GetLatestVimScripts is allowed
                to attempt to automatically install scripts. Furthermore, the
                plugin author has to have explicitly indicated that his/her
                plugin is automatically installable (via the :AutoInstall:
                keyword in the GetLatestVimScripts comment line).
>
        g:GetLatestVimScripts_autoinstalldir
<   default= $HOME/.vim   (linux)
        default= $HOME/vimfiles (windows)
                Override where :AutoInstall: scripts will be installed.
                Doesn't override vimball installation.
>
        g:GetLatestVimScripts_scriptaddr
<   default='http://vim.sourceforge.net/script.php?script_id='
                Override this if your system needs
          ... ='http://vim.sourceforge.net/script/script.php?script_id='

==
8. GetLatestVimScripts Algorithm   *glvs-algorithm* *glvs-alg*

The Vim sourceforge page dynamically creates a page by keying off of the
so-called script-id. Within the webpage of

        http://vim.sourceforge.net/scripts/script.php?script_id=40

is a line specifying the latest source-id (src_id). The source identifier
numbers are always increasing, hence if the src_id is greater than the one

```
recorded for the script in GetLatestVimScripts then it's time to download a
newer copy of that script.

GetLatestVimScripts will then download the script and update its internal
database of script ids, source ids, and scriptnames.

The AutoInstall process will:

        Move the file from GetLatest/ to the following directory
                Unix   : $HOME/.vim
                Windows: $HOME\vimfiles
        if the downloaded file ends with ".bz2"
                bunzip2 it
        else if the downloaded file ends with ".gz"
                gunzip it
        if the resulting file ends with ".zip"
                unzip it
        else if the resulting file ends with ".tar"
                tar -oxvf it
        else if the resulting file ends with ".vim"
                move it to the plugin subdirectory


==============================================================================
9. GetLatestVimScripts History          *getscript-history* *glvs-hist* {{{1

v36 Apr 22, 2013 : * (glts) suggested use of plugin/**/*.vim instead of
                     plugin/*.vim in globpath() call.
                   * (Andy Wokula) got warning message when setting
                     g:loaded_getscriptPlugin
v35 Apr 07, 2012 : * (MengHuan Yu) pointed out that the script URL has
                     changed (somewhat).  However, it doesn't work, and
                     the original one does (under Linux). I'll make it
                     yet-another-option.
v34 Jun 23, 2011 : * handles additional decompression options for tarballs
                     (tgz taz tbz txz)
v33 May 31, 2011 : * using fnameescape() instead of escape()
                   * *.xz support
v32 Jun 19, 2010 : * (Jan Steffens) added support for xz compression
v31 Jun 29, 2008 : * (Bill McCarthy) fixed having hls enabled with getscript
                   * (David Schaefer) the acd option interferes with vimballs
                     Solution: bypass the acd option
v30 Jun 13, 2008 : * GLVS now checks for existence of fnameescape() and will
                     issue an error message if it is not supported
v29 Jan 07, 2008 : * Bram M pointed out that cpo is a global option and that
                     getscriptPlugin.vim was setting it but not restoring it.
v28 Jan 02, 2008 : * improved shell quoting character handling, cygwin
                     interface, register-a bypass
    Oct 29, 2007   * Bill McCarthy suggested a change to getscript that avoids
                     creating pop-up windows
v24 Apr 16, 2007 : * removed save&restore of the fo option during script
                     loading
v23 Nov 03, 2006 : * ignores comments (#...)
                   * handles vimballs
v22 Oct 13, 2006 : * supports automatic use of curl if wget is not
                     available
v21 May 01, 2006 : * now takes advantage of autoloading.
v20 Dec 23, 2005 : * Eric Haarbauer found&fixed a bug with unzip use;
                     unzip needs the -o flag to overwrite.
v19 Nov 28, 2005 : * v18's GetLatestVimScript line accessed the wrong
                     script! Fixed.
v18 Mar 21, 2005 : * bugfix to automatic database construction
```

```
                    * bugfix - nowrapscan caused an error
                      (tnx to David Green for the fix)
    Apr 01, 2005    * if shell is bash, "mv" instead of "ren" used in
                      :AutoInstall:s, even though its o/s is windows
    Apr 01, 2005    * when downloading errors occurred, GLVS was
                      terminating early.  It now just goes on to trying
                      the next script (after trying three times to
                      download a script description page)
    Apr 20, 2005    * bugfix - when a failure to download occurred,
                      GetLatestVimScripts would stop early and claim that
                      everything was current.  Fixed.
v17 Aug 25, 2004 : * g:GetLatestVimScripts_allowautoinstall, which
                      defaults to 1, can be used to prevent all
                      :AutoInstall:
v16 Aug 25, 2004 : * made execution of bunzip2/gunzip/tar/zip silent
                    * fixed bug with :AutoInstall: use of helptags
v15 Aug 24, 2004 : * bugfix: the "0 0 comment" download prevention wasn't
                      always preventing downloads (just usually).  Fixed.
v14 Aug 24, 2004 : * bugfix -- helptags was using dotvim, rather than
                      s:dotvim.  Fixed.
v13 Aug 23, 2004 : * will skip downloading a file if its scriptid or srcid
                      is zero.  Useful for script authors; that way their
                      own GetLatestVimScripts activity won't overwrite
                      their scripts.
v12 Aug 23, 2004 : * bugfix - a "return" got left in the distribution that
                      was intended only for testing.  Removed, now works.
                    * :AutoInstall: implemented
v11 Aug 20, 2004 : * GetLatestVimScripts is now a plugin:
                    * :GetLatestVimScripts command
                    * (runtimepath)/GetLatest/GetLatestVimScripts.dat
                      now holds scripts that need updating
v10 Apr 19, 2004 : * moved history from script to doc
v9  Jan 23, 2004 :   windows (win32/win16/win95) will use
                     double quotes ("") whereas other systems will use
                     single quotes ('') around the urls in calls via wget
v8  Dec 01, 2003 :   makes three tries at downloading
v7  Sep 02, 2003 :   added error messages if "Click on..." or "src_id="
                     not found in downloaded webpage
                     Uses t_ti, t_te, and rs to make progress visible
v6  Aug 06, 2003 :   final status messages now display summary of work
                     ( "Downloaded someqty scripts" or
                       "Everything was current")
                     Now GetLatestVimScripts is careful about downloading
                     GetLatestVimScripts.vim itself!
                     (goes to <NEW_GetLatestVimScripts.vim>)
v5  Aug 04, 2003 :   missing an endif near bottom
v4  Jun 17, 2003 :   redraw! just before each "considering" message
v3  May 27, 2003 :   Protects downloaded files from errant shell
                     expansions with single quotes: '...'
v2  May 14, 2003 :   extracts name of item to be obtained from the
                     script file.  Uses it instead of comment field
                     for output filename; comment is used in the
                     "considering..." line and is now just a comment!
                    * Fixed a bug: a string-of-numbers is not the
                      same as a number, so I added zero to them
                      and they became numbers.  Fixes comparison.

==============================================================================
vim:tw=78:ts=8:ft=help:fdm=marker
*pi_gzip.txt*   For Vim version 8.0.  Last change: 2016 Nov 06
```

VIM REFERENCE MANUAL by Bram Moolenaar

Editing compressed files with Vim *gzip* *bzip2* *compress*

1. Autocommands |gzip-autocmd|

The functionality mentioned here is a |standard-plugin|.
This plugin is only available if 'compatible' is not set.
You can avoid loading this plugin by setting the "loaded_gzip" variable: >
:let loaded_gzip = 1

{Vi does not have any of this}

==
1. Autocommands *gzip-autocmd*

The plugin installs autocommands to intercept reading and writing of files
with these extensions:

extension	compression ~
*.Z	compress (Lempel-Ziv)
*.gz	gzip
*.bz2	bzip2
*.lzma	lzma
*.xz	xz
*.lz	lzip
*.zst	zstd

That's actually the only thing you need to know. There are no options.

After decompressing a file, the filetype will be detected again. This will
make a file like "foo.c.gz" get the "c" filetype.

If you have 'patchmode' set, it will be appended after the extension for
compression. Thus editing the patchmode file will not give you the automatic
decompression. You have to rename the file if you want this.

==
vim:tw=78:ts=8:ft=help:norl:
pi_logipat.txt Logical Patterns Jun 22, 2015

Author: Charles E. Campbell <NdrOchip@ScampbellPfamily.AbizM>
Copyright: (c) 2004-2015 by Charles E. Campbell *logiPat-copyright*
The VIM LICENSE applies to LogiPat.vim and LogiPat.txt
(see |copyright|) except use "LogiPat" instead of "Vim"
No warranty, express or implied. Use At-Your-Own-Risk.

==
1. Contents *logiPat* *logiPat-contents*

1. Contents.....: |logiPat-contents|
2. LogiPat Manual.....: |logiPat-manual|
3. LogiPat Examples.....: |logiPat-examples|
4. Caveat.....: |logiPat-caveat|
5. LogiPat History.....: |logiPat-history|

==
2. LogiPat Manual *logiPat-manual* *logiPat-man*

logiPat-arg *logiPat-input* *logiPat-pattern* *logiPat-operators*

```
        Boolean logic patterns are composed of

                        operators  ! = not
                                   | = logical-or
                                   & = logical-and
                        grouping   ( ... )
                        patterns   "pattern"

        :LogiPat {boolean-logic pattern}                *:LogiPat*
                :LogiPat is a command which takes a boolean-logic
                argument (|logiPat-arg|).

        :LP {boolean-logic pattern}                     *:LP*
                :LP is a shorthand command version of :LogiPat

        :LPE {boolean-logic pattern}                    *:LPE*
                No search is done, but the conversion from the
                boolean logic pattern to the regular expression
                is performed and echoed onto the display.

        :LogiPatFlags {search flags}                    *LogiPat-flags*
                :LogiPatFlags {search flags}
                LogiPat uses the |search()| command.  The flags
                passed to that call to search() may be specified
                by the :LogiPatFlags command.

        :LPF {search flags}                             *:LPF*
                :LPF is a shorthand version of :LogiPatFlags.

        :let pat=LogiPat({boolean-logic pattern})       *LogiPat()*
                If one calls LogiPat() directly, no search
                is done, but the transformation from the boolean
                logic pattern into a regular expression pattern
                is performed and returned.

        To get a " inside a pattern, as opposed to having it delimit
        the pattern, double it.


==============================================================================
3. LogiPat Examples                                     *logiPat-examples*

        LogiPat takes Boolean logic arguments and produces a regular
        expression which implements the choices.  A series of examples
        follows:
>
        :LogiPat "abc"
<               will search for lines containing the string  :abc:
>
        :LogiPat "ab""cd"
<               will search for lines containing the string  :ab"c:
>
        :LogiPat !"abc"
<               will search for lines which don't contain the string  :abc:
>
        :LogiPat "abc"|"def"
<               will search for lines which contain either the string
                :abc:  or the string  :def:
>
        :LogiPat !("abc"|"def")
<               will search for lines which don't contain either
                of the strings  :abc:  or  :def:
```

```
>
        :LogiPat "abc"&"def"
<               will search for lines which contain both of the strings
                :abc:  and  :def:
>
        :let pat= LogiPat('!"abc"')
<               will return the regular expression which will match
                all lines not containing  :abc: .  The double quotes
                are needed to pass normal patterns to LogiPat, and
                differentiate such patterns from boolean logic
                operators.


==============================================================================
4. Caveat                                                *logiPat-caveat*

        The "not" operator may be fragile; ie. it may not always play well
        with the & (logical-and) and | (logical-or) operators.  Please try out
        your patterns, possibly with :set hls, to insure that what is matching
        is what you want.


==============================================================================
3. LogiPat History                                       *logiPat-history*

        v4 Jun 22, 2015 * LogiPat has been picked up by Bram M for standard
                          plugin distribution; hence the name change
        v3 Sep 25, 2006 * LP_Or() fixed; it now encapsulates its output
                          in \%(...\) parentheses
           Dec 12, 2011 * |:LPE| added
                        * "" is mapped to a single " and left inside patterns
        v2 May 31, 2005 * LPF and LogiPatFlags commands weren't working
        v1 May 23, 2005 * initial release


==============================================================================
vim:tw=78:ts=8:ft=help
*pi_netrw.txt*  For Vim version 8.0.  Last change: 2016 Apr 20

            ------------------------------------------------
            NETRW REFERENCE MANUAL    by Charles E. Campbell
            ------------------------------------------------
Author:  Charles E. Campbell  <NdrOchip@ScampbellPfamily.AbizM>
          (remove NOSPAM from Campbell's email first)

Copyright: Copyright (C) 2016 Charles E Campbell    *netrw-copyright*
        The VIM LICENSE applies to the files in this package, including
        netrw.vim, pi_netrw.txt, netrwFileHandlers.vim, netrwSettings.vim, and
        syntax/netrw.vim.  Like anything else that's free, netrw.vim and its
        associated files are provided *as is* and comes with no warranty of
        any kind, either expressed or implied.  No guarantees of
        merchantability.  No guarantees of suitability for any purpose.  By
        using this plugin, you agree that in no event will the copyright
        holder be liable for any damages resulting from the use of this
        software. Use at your own risk!


                *netrw*
                *dav*    *ftp*    *netrw-file*  *rcp*    *scp*
                *davs*   *http*   *netrw.vim*   *rsync*  *sftp*
                *fetch*  *network*
```

```
==============================================================================
1. Contents                                          *netrw-contents* {{{1

1.  Contents.....................................|netrw-contents|
2.  Starting With Netrw..........................|netrw-start|
3.  Netrw Reference..............................|netrw-ref|
      EXTERNAL APPLICATIONS AND PROTOCOLS........|netrw-externapp|
      READING....................................|netrw-read|
      WRITING....................................|netrw-write|
      SOURCING...................................|netrw-source|
      DIRECTORY LISTING..........................|netrw-dirlist|
      CHANGING THE USERID AND PASSWORD...........|netrw-chgup|
      VARIABLES AND SETTINGS.....................|netrw-variables|
      PATHS......................................|netrw-path|
4.  Network-Oriented File Transfer...............|netrw-xfer|
      NETRC......................................|netrw-netrc|
      PASSWORD...................................|netrw-passwd|
5.  Activation...................................|netrw-activate|
6.  Transparent Remote File Editing..............|netrw-transparent|
7.  Ex Commands..................................|netrw-ex|
8.  Variables and Options........................|netrw-variables|
9.  Browsing.....................................|netrw-browse|
      Introduction To Browsing...................|netrw-intro-browse|
      Quick Reference: Maps......................|netrw-browse-maps|
      Quick Reference: Commands..................|netrw-browse-cmds|
      Banner Display.............................|netrw-I|
      Bookmarking A Directory....................|netrw-mb|
      Browsing...................................|netrw-cr|
      Squeezing the Current Tree-Listing Directory........|netrw-s-cr|
      Browsing With A Horizontally Split Window..........|netrw-o|
      Browsing With A New Tab....................|netrw-t|
      Browsing With A Vertically Split Window....|netrw-v|
      Change Listing Style.(thin wide long tree).........|netrw-i|
      Changing To A Bookmarked Directory.........|netrw-gb|
      Changing To A Predecessor Directory........|netrw-u|
      Changing To A Successor Directory..........|netrw-U|
      Customizing Browsing With A Special Handler.........|netrw-x|
      Deleting Bookmarks.........................|netrw-mB|
      Deleting Files Or Directories..............|netrw-D|
      Directory Exploring Commands...............|netrw-explore|
      Exploring With Stars and Patterns..........|netrw-star|
      Displaying Information About File..........|netrw-qf|
      Edit File Or Directory Hiding List.........|netrw-ctrl-h|
      Editing The Sorting Sequence...............|netrw-S|
      Forcing treatment as a file or directory...........|netrw-gd| |netrw-gf|
      Going Up...................................|netrw--|
      Hiding Files Or Directories................|netrw-a|
      Improving Browsing.........................|netrw-ssh-hack|
      Listing Bookmarks And History..............|netrw-qb|
      Making A New Directory.....................|netrw-d|
      Making The Browsing Directory The Current Directory.|netrw-c|
      Marking Files..............................|netrw-mf|
      Unmarking Files............................|netrw-mF|
      Marking Files By Location List.............|netrw-qL|
      Marking Files By QuickFix List.............|netrw-qF|
      Marking Files By Regular Expression........|netrw-mr|
      Marked Files: Arbitrary Shell Command......|netrw-mx|
      Marked Files: Arbitrary Shell Command, En Bloc......|netrw-mX|
      Marked Files: Arbitrary Vim Command........|netrw-mv|
      Marked Files: Argument List................|netrw-ma| |netrw-mA|
      Marked Files: Compression And Decompression.........|netrw-mz|
      Marked Files: Copying......................|netrw-mc|
```

      Marked Files: Diff.....|netrw-md|
      Marked Files: Editing.....|netrw-me|
      Marked Files: Grep.....|netrw-mg|
      Marked Files: Hiding and Unhiding by Suffix.....|netrw-mh|
      Marked Files: Moving.....|netrw-mm|
      Marked Files: Printing.....|netrw-mp|
      Marked Files: Sourcing.....|netrw-ms|
      Marked Files: Setting the Target Directory.....|netrw-mt|
      Marked Files: Tagging.....|netrw-mT|
      Marked Files: Target Directory Using Bookmarks.....|netrw-Tb|
      Marked Files: Target Directory Using History.....|netrw-Th|
      Marked Files: Unmarking.....|netrw-mu|
      Netrw Browser Variables.....|netrw-browser-var|
      Netrw Browsing And Option Incompatibilities.....|netrw-incompatible|
      Netrw Settings Window.....|netrw-settings-window|
      Obtaining A File.....|netrw-O|
      Preview Window.....|netrw-p|
      Previous Window.....|netrw-P|
      Refreshing The Listing.....|netrw-ctrl-l|
      Reversing Sorting Order.....|netrw-r|
      Renaming Files Or Directories.....|netrw-R|
      Selecting Sorting Style.....|netrw-s|
      Setting Editing Window.....|netrw-C|
10. Problems and Fixes.....|netrw-problems|
11. Debugging Netrw Itself.....|netrw-debug|
12. History.....|netrw-history|
13. Todo.....|netrw-todo|
14. Credits.....|netrw-credits|

{Vi does not have any of this}

==
2. Starting With Netrw *netrw-start* {{{1

Netrw makes reading files, writing files, browsing over a network, and
local browsing easy! First, make sure that you have plugins enabled, so
you'll need to have at least the following in your <.vimrc>:
(or see |netrw-activate|) >

```
        set nocp                    " 'compatible' is not set
        filetype plugin on          " plugins are enabled
<
```

(see |'cp'| and |:filetype-plugin-on|)

Netrw supports "transparent" editing of files on other machines using urls
(see |netrw-transparent|). As an example of this, let's assume you have an
account on some other machine; if you can use scp, try: >

```
        vim scp://hostname/path/to/file
<
```

Want to make ssh/scp easier to use? Check out |netrw-ssh-hack|!

So, what if you have ftp, not ssh/scp? That's easy, too; try >

```
        vim ftp://hostname/path/to/file
<
```

Want to make ftp simpler to use? See if your ftp supports a file called
<.netrc> -- typically it goes in your home directory, has read/write
permissions for only the user to read (ie. not group, world, other, etc),
and has lines resembling >

```
        machine HOSTNAME login USERID password "PASSWORD"
```

```
        machine HOSTNAME login USERID password "PASSWORD"
        ...
        default          login USERID password "PASSWORD"
<
Windows' ftp doesn't support .netrc; however, one may have in one's .vimrc:  >

   let g:netrw_ftp_cmd= 'c:\Windows\System32\ftp -s:C:\Users\MyUserName\MACHINE'
<
Netrw will substitute the host's machine name for "MACHINE" from the url it is
attempting to open, and so one may specify >
        userid
        password
for each site in a separate file: c:\Users\MyUserName\MachineName.

Now about browsing -- when you just want to look around before editing a
file.  For browsing on your current host, just "edit" a directory: >

        vim .
        vim /home/userid/path
<
For browsing on a remote host, "edit" a directory (but make sure that
the directory name is followed by a "/"): >

        vim scp://hostname/
        vim ftp://hostname/path/to/dir/
<
See |netrw-browse| for more!

There are more protocols supported by netrw than just scp and ftp, too: see the
next section, |netrw-externapp|, on how to use these external applications with
netrw and vim.

PREVENTING LOADING                                      *netrw-noload*

If you want to use plugins, but for some reason don't wish to use netrw, then
you need to avoid loading both the plugin and the autoload portions of netrw.
You may do so by placing the following two lines in your <.vimrc>: >

        :let g:loaded_netrw       = 1
        :let g:loaded_netrwPlugin = 1
<

==============================================================================
3. Netrw Reference                                      *netrw-ref* {{{1

   Netrw supports several protocols in addition to scp and ftp as mentioned
   in |netrw-start|.  These include dav, fetch, http,... well, just look
   at the list in |netrw-externapp|.  Each protocol is associated with a
   variable which holds the default command supporting that protocol.

EXTERNAL APPLICATIONS AND PROTOCOLS                     *netrw-externapp* {{{2

        Protocol  Variable             Default Value
        --------  ---------------      -------------
           dav:   *g:netrw_dav_cmd*      = "cadaver"    if cadaver is executable
           dav:   g:netrw_dav_cmd      = "curl -o"    elseif curl is available
         fetch:   *g:netrw_fetch_cmd*    = "fetch -o"   if fetch is available
           ftp:   *g:netrw_ftp_cmd*      = "ftp"
          http:   *g:netrw_http_cmd*     = "elinks"     if   elinks  is available
          http:   g:netrw_http_cmd     = "links"      elseif links is available
          http:   g:netrw_http_cmd     = "curl"       elseif curl  is available
          http:   g:netrw_http_cmd     = "wget"       elseif wget  is available
```

```
          http:   g:netrw_http_cmd     = "fetch"      elseif fetch is available
          http:   *g:netrw_http_put_cmd* = "curl -T"
           rcp:   *g:netrw_rcp_cmd*      = "rcp"
         rsync:   *g:netrw_rsync_cmd*    = "rsync -a"
           scp:   *g:netrw_scp_cmd*      = "scp -q"
          sftp:   *g:netrw_sftp_cmd*     = "sftp"
          file:   *g:netrw_file_cmd*     = "elinks" or "links"

        *g:netrw_http_xcmd* : the option string for http://... protocols are
        specified via this variable and may be independently overridden.  By
        default, the option arguments for the http-handling commands are: >

                    elinks : "-source >"
                    links  : "-dump >"
                    curl   : "-o"
                    wget   : "-q -O"
                    fetch  : "-o"
<
        For example, if your system has elinks, and you'd rather see the
        page using an attempt at rendering the text, you may wish to have >
                let g:netrw_http_xcmd= "-dump >"
<       in your .vimrc.

        g:netrw_http_put_cmd: this option specifies both the executable and
        any needed options.  This command does a PUT operation to the url.


READING                                          *netrw-read* *netrw-nread* {{{2

        Generally, one may just use the url notation with a normal editing
        command, such as >

                :e ftp://[user@]machine/path
<
        Netrw also provides the Nread command:

        :Nread ?                                        give help
        :Nread "machine:path"                           uses rcp
        :Nread "machine path"                           uses ftp w/ <.netrc>
        :Nread "machine id password path"               uses ftp
        :Nread "dav://machine[:port]/path"              uses cadaver
        :Nread "fetch://[user@]machine/path"            uses fetch
        :Nread "ftp://[user@]machine[[:#]port]/path"    uses ftp w/ <.netrc>
        :Nread "http://[user@]machine/path"             uses http  uses wget
        :Nread "rcp://[user@]machine/path"              uses rcp
        :Nread "rsync://[user@]machine[:port]/path"     uses rsync
        :Nread "scp://[user@]machine[[:#]port]/path"    uses scp
        :Nread "sftp://[user@]machine/path"             uses sftp

WRITING                                    *netrw-write* *netrw-nwrite* {{{2

        One may just use the url notation with a normal file writing
        command, such as >

                :w ftp://[user@]machine/path
<
        Netrw also provides the Nwrite command:

        :Nwrite ?                                       give help
        :Nwrite "machine:path"                          uses rcp
        :Nwrite "machine path"                          uses ftp w/ <.netrc>
        :Nwrite "machine id password path"              uses ftp
```

```
        :Nwrite "dav://machine[:port]/path"             uses cadaver
        :Nwrite "ftp://[user@]machine[[:#]port]/path"   uses ftp w/ <.netrc>
        :Nwrite "rcp://[user@]machine/path"             uses rcp
        :Nwrite "rsync://[user@]machine[:port]/path"    uses rsync
        :Nwrite "scp://[user@]machine[[:#]port]/path"   uses scp
        :Nwrite "sftp://[user@]machine/path"            uses sftp
        http: not supported!

SOURCING                                        *netrw-source* {{{2

        One may just use the url notation with the normal file sourcing
        command, such as >

                :so ftp://[user@]machine/path
<
        Netrw also provides the Nsource command:

        :Nsource ?                                      give help
        :Nsource "dav://machine[:port]/path"            uses cadaver
        :Nsource "fetch://[user@]machine/path"          uses fetch
        :Nsource "ftp://[user@]machine[[:#]port]/path"  uses ftp w/ <.netrc>
        :Nsource "http://[user@]machine/path"           uses http  uses wget
        :Nsource "rcp://[user@]machine/path"            uses rcp
        :Nsource "rsync://[user@]machine[:port]/path"   uses rsync
        :Nsource "scp://[user@]machine[[:#]port]/path"  uses scp
        :Nsource "sftp://[user@]machine/path"           uses sftp

DIRECTORY LISTING               *netrw-trailingslash* *netrw-dirlist* {{{2

        One may browse a directory to get a listing by simply attempting to
        edit the directory: >

                :e scp://[user]@hostname/path/
                :e ftp://[user]@hostname/path/
<
        For remote directory listings (ie. those using scp or ftp), that
        trailing "/" is necessary (the slash tells netrw to treat the argument
        as a directory to browse instead of as a file to download).

        The Nread command may also be used to accomplish this (again, that
        trailing slash is necessary): >

                :Nread [protocol]://[user]@hostname/path/
<
                                        *netrw-login* *netrw-password*
CHANGING USERID AND PASSWORD            *netrw-chgup* *netrw-userpass* {{{2

        Attempts to use ftp will prompt you for a user-id and a password.
        These will be saved in global variables |g:netrw_uid| and
        |s:netrw_passwd|; subsequent use of ftp will re-use those two strings,
        thereby simplifying use of ftp.  However, if you need to use a
        different user id and/or password, you'll want to call |NetUserPass()|
        first.  To work around the need to enter passwords, check if your ftp
        supports a <.netrc> file in your home directory.  Also see
        |netrw-passwd| (and if you're using ssh/scp hoping to figure out how
        to not need to use passwords for scp, look at |netrw-ssh-hack|).

        :NetUserPass [uid [password]]           -- prompts as needed
        :call NetUserPass()                     -- prompts for uid and password
        :call NetUserPass("uid")                -- prompts for password
        :call NetUserPass("uid","password")     -- sets global uid and password
```

```
(Related topics: |ftp| |netrw-userpass| |netrw-start|)

NETRW VARIABLES AND SETTINGS                              *netrw-variables* {{{2
    (Also see:
    |netrw-browser-var|     : netrw browser option variables
    |netrw-protocol|        : file transfer protocol option variables
    |netrw-settings|        : additional file transfer options
    |netrw-browser-options| : these options affect browsing directories
    )

Netrw provides a lot of variables which allow you to customize netrw to your
preferences.  One way to look at them is via the command :NetrwSettings (see
|netrw-settings|) which will display your current netrw settings.  Most such
settings are described below, in |netrw-browser-options|, and in
|netrw-externapp|:

 *b:netrw_lastfile*     last file Network-read/written retained on a
                        per-buffer basis (supports plain :Nw )

 *g:netrw_bufsettings*  the settings that netrw buffers have
                        (default) noma nomod nonu nowrap ro nobl

 *g:netrw_chgwin*       specifies a window number where subsequent file edits
                        will take place.  (also see |netrw-C|)
                        (default) -1

 *g:Netrw_funcref*      specifies a function (or functions) to be called when
                        netrw edits a file.  The file is first edited, and
                        then the function reference (|Funcref|) is called.
                        This variable may also hold a |List| of Funcrefs.
                        (default) not defined.  (the capital in g:Netrw...
                        is required by its holding a function reference)
>
                            Example: place in .vimrc; affects all file opening
                            fun! MyFuncRef()
                            endfun
                            let g:Netrw_funcref= function("MyFuncRef")

<
 *g:Netrw_UserMaps*     specifies a function or |List| of functions which can
                        be used to set up user-specified maps and functionality.
                        See |netrw-usermaps|

 *g:netrw_ftp*             if it doesn't exist, use default ftp
                        =0 use default ftp                    (uid password)
                        =1 use alternate ftp method       (user uid password)
                           If you're having trouble with ftp, try changing the
                           value of this variable to see if the alternate ftp
                           method works for your setup.

 *g:netrw_ftp_options*     Chosen by default, these options are supposed to
                         turn interactive prompting off and to restrain ftp
                         from attempting auto-login upon initial connection.
                         However, it appears that not all ftp implementations
                         support this (ex. ncftp).
                        ="-i -n"

 *g:netrw_ftpextracmd*  default: doesn't exist
                        If this variable exists, then any string it contains
                        will be placed into the commands set to your ftp
                        client.  As an example:
                           ="passive"
```

```
*g:netrw_ftpmode*      ="binary"                                  (default)
                       ="ascii"

*g:netrw_ignorenetrc*  =0 (default for linux, cygwin)
                       =1 If you have a <.netrc> file but it doesn't work and
                          you want it ignored, then set this variable as
                          shown. (default for Windows + cmd.exe)

*g:netrw_menu*         =0 disable netrw's menu
                       =1 (default) netrw's menu enabled

*g:netrw_nogx*         if this variable exists, then the "gx" map will not
                       be available (see |netrw-gx|)

*g:netrw_uid*          (ftp) user-id,      retained on a per-vim-session basis
*s:netrw_passwd*       (ftp) password,     retained on a per-vim-session basis

*g:netrw_preview*      =0 (default) preview window shown in a horizontally
                          split window
                       =1 preview window shown in a vertically split window.
                          Also affects the "previous window" (see |netrw-P|)
                          in the same way.
                       The |g:netrw_alto| variable may be used to provide
                       additional splitting control:
                              g:netrw_preview g:netrw_alto result
                                       0             0     |:aboveleft|
                                       0             1     |:belowright|
                                       1             0     |:topleft|
                                       1             1     |:botright|
                       To control sizing, see |g:netrw_winsize|

*g:netrw_scpport*      = "-P" : option to use to set port for scp
*g:netrw_sshport*      = "-p" : option to use to set port for ssh

*g:netrw_sepchr*       =\0xff
                       =\0x01 for enc == euc-jp (and perhaps it should be for
                          others, too, please let me know)
                          Separates priority codes from filenames internally.
                          See |netrw-p12|.

 *g:netrw_silent*      =0 : transfers done normally
                       =1 : transfers done silently

*g:netrw_use_errorwindow* =1 : messages from netrw will use a separate one
                             line window.  This window provides reliable
                             delivery of messages. (default)
                          =0 : messages from netrw will use echoerr ;
                             messages don't always seem to show up this
                             way, but one doesn't have to quit the window.

*g:netrw_win95ftp*     =1 if using Win95, will remove four trailing blank
                          lines that o/s's ftp "provides" on transfers
                       =0 force normal ftp behavior (no trailing line removal)

*g:netrw_cygwin*       =1 assume scp under windows is from cygwin. Also
                          permits network browsing to use ls with time and
                          size sorting (default if windows)
                       =0 assume Windows' scp accepts windows-style paths
                          Network browsing uses dir instead of ls
                          This option is ignored if you're using unix
```

```
 *g:netrw_use_nt_rcp*   =0 don't use the rcp of WinNT, Win2000 and WinXP
                        =1 use WinNT's rcp in binary mode         (default)

PATHS                                                    *netrw-path* {{{2

Paths to files are generally user-directory relative for most protocols.
It is possible that some protocol will make paths relative to some
associated directory, however.
>
        example:  vim scp://user@host/somefile
        example:  vim scp://user@host/subdir1/subdir2/somefile
<
where "somefile" is in the "user"'s home directory.  If you wish to get a
file using root-relative paths, use the full path:
>
        example:  vim scp://user@host//somefile
        example:  vim scp://user@host//subdir1/subdir2/somefile
<

==============================================================================
4. Network-Oriented File Transfer                        *netrw-xfer* {{{1

Network-oriented file transfer under Vim is implemented by a VimL-based script
(<netrw.vim>) using plugin techniques.  It currently supports both reading and
writing across networks using rcp, scp, ftp or ftp+<.netrc>, scp, fetch,
dav/cadaver, rsync, or sftp.

http is currently supported read-only via use of wget or fetch.

<netrw.vim> is a standard plugin which acts as glue between Vim and the
various file transfer programs.  It uses autocommand events (BufReadCmd,
FileReadCmd, BufWriteCmd) to intercept reads/writes with url-like filenames. >

        ex. vim ftp://hostname/path/to/file
<
The characters preceding the colon specify the protocol to use; in the
example, it's ftp.  The <netrw.vim> script then formulates a command or a
series of commands (typically ftp) which it issues to an external program
(ftp, scp, etc) which does the actual file transfer/protocol.  Files are read
from/written to a temporary file (under Unix/Linux, /tmp/...) which the
<netrw.vim> script will clean up.

Now, a word about Jan Minář's "FTP User Name and Password Disclosure"; first,
ftp is not a secure protocol.  User names and passwords are transmitted "in
the clear" over the internet; any snooper tool can pick these up; this is not
a netrw thing, this is a ftp thing.  If you're concerned about this, please
try to use scp or sftp instead.

Netrw re-uses the user id and password during the same vim session and so long
as the remote hostname remains the same.

Jan seems to be a bit confused about how netrw handles ftp; normally multiple
commands are performed in a "ftp session", and he seems to feel that the
uid/password should only be retained over one ftp session.  However, netrw
does every ftp operation in a separate "ftp session"; so remembering the
uid/password for just one "ftp session" would be the same as not remembering
the uid/password at all.  IMHO this would rapidly grow tiresome as one
browsed remote directories, for example.

On the other hand, thanks go to Jan M. for pointing out the many
vulnerabilities that netrw (and vim itself) had had in handling "crafted"
filenames.  The |shellescape()| and |fnameescape()| functions were written in
```

```
response by Bram Moolenaar to handle these sort of problems, and netrw has
been modified to use them.  Still, my advice is, if the "filename" looks like
a vim command that you aren't comfortable with having executed, don't open it.

                                *netrw-putty* *netrw-pscp* *netrw-psftp*
One may modify any protocol's implementing external application by setting a
variable (ex. scp uses the variable g:netrw_scp_cmd, which is defaulted to
"scp -q").  As an example, consider using PuTTY: >

        let g:netrw_scp_cmd = '"c:\Program Files\PuTTY\pscp.exe" -q -batch'
        let g:netrw_sftp_cmd= '"c:\Program Files\PuTTY\psftp.exe"'
<
(note: it has been reported that windows 7 with putty v0.6's "-batch" option
       doesn't work, so it's best to leave it off for that system)

See |netrw-p8| for more about putty, pscp, psftp, etc.

Ftp, an old protocol, seems to be blessed by numerous implementations.
Unfortunately, some implementations are noisy (ie., add junk to the end of the
file).  Thus, concerned users may decide to write a NetReadFixup() function
that will clean up after reading with their ftp.  Some Unix systems (ie.,
FreeBSD) provide a utility called "fetch" which uses the ftp protocol but is
not noisy and more convenient, actually, for <netrw.vim> to use.
Consequently, if "fetch" is available (ie. executable), it may be preferable
to use it for ftp://... based transfers.

For rcp, scp, sftp, and http, one may use network-oriented file transfers
transparently; ie.
>
        vim rcp://[user@]machine/path
        vim scp://[user@]machine/path
<
If your ftp supports <.netrc>, then it too can be transparently used
if the needed triad of machine name, user id, and password are present in
that file.  Your ftp must be able to use the <.netrc> file on its own, however.
>
        vim ftp://[user@]machine[[:#]portnumber]/path
<
Windows provides an ftp (typically c:\Windows\System32\ftp.exe) which uses
an option, -s:filename (filename can and probably should be a full path)
which contains ftp commands which will be automatically run whenever ftp
starts.  You may use this feature to enter a user and password for one site: >
        userid
        password
<                               *netrw-windows-netrc*  *netrw-windows-s*
If |g:netrw_ftp_cmd| contains -s:[path/]MACHINE, then (on Windows machines
only) netrw will substitute the current machine name requested for ftp
connections for MACHINE.  Hence one can have multiple machine.ftp files
containing login and password for ftp.  Example: >

    let g:netrw_ftp_cmd= 'c:\Windows\System32\ftp -s:C:\Users\Myself\MACHINE'
    vim ftp://myhost.somewhere.net/

will use a file >

        C:\Users\Myself\myhost.ftp
<
Often, ftp will need to query the user for the userid and password.
The latter will be done "silently"; ie. asterisks will show up instead of
the actually-typed-in password.  Netrw will retain the userid and password
for subsequent read/writes from the most recent transfer so subsequent
transfers (read/write) to or from that machine will take place without
```

additional prompting.

netrw-urls

Reading	Writing	Uses
DAV:		
dav://host/path		cadaver
:Nread dav://host/path	:Nwrite dav://host/path	cadaver
DAV + SSL:		
davs://host/path		cadaver
:Nread davs://host/path	:Nwrite davs://host/path	cadaver
FETCH:		
fetch://[user@]host/path		
fetch://[user@]host:http/path	Not Available	fetch
:Nread fetch://[user@]host/path		
FILE:		
file:///*	file:///*	
file://localhost/*	file://localhost/*	
FTP: (*3)	(*3)	
ftp://[user@]host/path	ftp://[user@]host/path	ftp (*2)
:Nread ftp://host/path	:Nwrite ftp://host/path	ftp+.netrc
:Nread host path	:Nwrite host path	ftp+.netrc
:Nread host uid pass path	:Nwrite host uid pass path	ftp
HTTP: wget is executable: (*4)		
http://[user@]host/path	Not Available	wget
HTTP: fetch is executable (*4)		
http://[user@]host/path	Not Available	fetch
RCP:		
rcp://[user@]host/path	rcp://[user@]host/path	rcp
RSYNC:		
rsync://[user@]host/path	rsync://[user@]host/path	rsync
:Nread rsync://host/path	:Nwrite rsync://host/path	rsync
:Nread rcp://host/path	:Nwrite rcp://host/path	rcp
SCP:		
scp://[user@]host/path	scp://[user@]host/path	scp
:Nread scp://host/path	:Nwrite scp://host/path	scp (*1)
SFTP:		
sftp://[user@]host/path	sftp://[user@]host/path	sftp
:Nread sftp://host/path	:Nwrite sftp://host/path	sftp (*1)

(*1) For an absolute path use scp://machine//path.

(*2) if <.netrc> is present, it is assumed that it will work with your ftp client. Otherwise the script will prompt for user-id and password.

(*3) for ftp, "machine" may be machine#port or machine:port if a different port is needed than the standard ftp port

(*4) for http:..., if wget is available it will be used. Otherwise,

```
        if fetch is available it will be used.

Both the :Nread and the :Nwrite ex-commands can accept multiple filenames.


NETRC                                                   *netrw-netrc*

The <.netrc> file, typically located in your home directory, contains lines
therein which map a hostname (machine name) to the user id and password you
prefer to use with it.

The typical syntax for lines in a <.netrc> file is given as shown below.
Ftp under Unix usually supports <.netrc>; ftp under Windows usually doesn't.
>
        machine {full machine name} login {user-id} password "{password}"
        default login {user-id} password "{password}"

Your ftp client must handle the use of <.netrc> on its own, but if the
<.netrc> file exists, an ftp transfer will not ask for the user-id or
password.

        Note:
        Since this file contains passwords, make very sure nobody else can
        read this file!  Most programs will refuse to use a .netrc that is
        readable for others.  Don't forget that the system administrator can
        still read the file!  Ie. for Linux/Unix: chmod 600 .netrc

Even though Windows' ftp clients typically do not support .netrc, netrw has
a work-around: see |netrw-windows-s|.


PASSWORD                                                *netrw-passwd*

The script attempts to get passwords for ftp invisibly using |inputsecret()|,
a built-in Vim function.  See |netrw-userpass| for how to change the password
after one has set it.

Unfortunately there doesn't appear to be a way for netrw to feed a password to
scp.  Thus every transfer via scp will require re-entry of the password.
However, |netrw-ssh-hack| can help with this problem.


==============================================================================
5. Activation                                           *netrw-activate* {{{1

Network-oriented file transfers are available by default whenever Vim's
|'nocompatible'| mode is enabled.  Netrw's script files reside in your
system's plugin, autoload, and syntax directories; just the
plugin/netrwPlugin.vim script is sourced automatically whenever you bring up
vim.  The main script in autoload/netrw.vim is only loaded when you actually
use netrw.  I suggest that, at a minimum, you have at least the following in
your <.vimrc> customization file: >

        set nocp
        if version >= 600
          filetype plugin indent on
        endif
<
By also including the following lines in your .vimrc, one may have netrw
immediately activate when using [g]vim without any filenames, showing the
current directory: >
```

```
        " Augroup VimStartup:
        augroup VimStartup
          au!
          au VimEnter * if expand("%") == "" | e . | endif
        augroup END
<

==============================================================================
6. Transparent Remote File Editing              *netrw-transparent* {{{1

Transparent file transfers occur whenever a regular file read or write
(invoked via an |:autocmd| for |BufReadCmd|, |BufWriteCmd|, or |SourceCmd|
events) is made.  Thus one may read, write, or source  files across networks
just as easily as if they were local files! >

        vim ftp://[user@]machine/path
        ...
        :wq

See |netrw-activate| for more on how to encourage your vim to use plugins
such as netrw.


==============================================================================
7. Ex Commands                                          *netrw-ex* {{{1

The usual read/write commands are supported.  There are also a few
additional commands available.  Often you won't need to use Nwrite or
Nread as shown in |netrw-transparent| (ie. simply use >
  :e url
  :r url
  :w url
instead, as appropriate) -- see |netrw-urls|.  In the explanations
below, a {netfile} is an url to a remote file.

                                                *:Nwrite*  *:Nw*
:[range]Nw[rite]        Write the specified lines to the current
                file as specified in b:netrw_lastfile.
                (related: |netrw-nwrite|)

:[range]Nw[rite] {netfile} [{netfile}]...
                Write the specified lines to the {netfile}.

                                                *:Nread*   *:Nr*
:Nr[ead]        Read the lines from the file specified in b:netrw_lastfile
                into the current buffer.  (related: |netrw-nread|)

:Nr[ead] {netfile} {netfile}...
                Read the {netfile} after the current line.

                                                *:Nsource* *:Ns*
:Ns[ource] {netfile}
                Source the {netfile}.
                To start up vim using a remote .vimrc, one may use
                the following (all on one line) (tnx to Antoine Mechelynck) >
                vim -u NORC -N
                 --cmd "runtime plugin/netrwPlugin.vim"
                 --cmd "source scp://HOSTNAME/.vimrc"
<                (related: |netrw-source|)

:call NetUserPass()                             *NetUserPass()*
                If g:netrw_uid and s:netrw_passwd don't exist,
```

```
                this function will query the user for them.
                (related: |netrw-userpass|)

:call NetUserPass("userid")
                This call will set the g:netrw_uid and, if
                the password doesn't exist, will query the user for it.
                (related: |netrw-userpass|)

:call NetUserPass("userid","passwd")
                This call will set both the g:netrw_uid and s:netrw_passwd.
                The user-id and password are used by ftp transfers.  One may
                effectively remove the user-id and password by using empty
                strings (ie. "").
                (related: |netrw-userpass|)

:NetrwSettings  This command is described in |netrw-settings| -- used to
                display netrw settings and change netrw behavior.


==============================================================================
8. Variables and Options                *netrw-var* *netrw-settings* {{{1

(also see: |netrw-options| |netrw-variables| |netrw-protocol|
           |netrw-browser-settings| |netrw-browser-options| )

The <netrw.vim> script provides several variables which act as options to
affect <netrw.vim>'s file transfer behavior.  These variables typically may be
set in the user's <.vimrc> file: (see also |netrw-settings| |netrw-protocol|)
                                                *netrw-options*
>
                        -------------
                        Netrw Options
                        -------------
        Option                  Meaning
        --------------          ------------------------------------------------
<
        b:netrw_col             Holds current cursor position (during NetWrite)
        g:netrw_cygwin          =1 assume scp under windows is from cygwin
                                                        (default/windows)
                                =0 assume scp under windows accepts windows
                                   style paths          (default/else)
        g:netrw_ftp             =0 use default ftp      (uid password)
        g:netrw_ftpmode         ="binary"               (default)
                                ="ascii"                (your choice)
        g:netrw_ignorenetrc     =1                      (default)
                                   if you have a <.netrc> file but you don't
                                   want it used, then set this variable.  Its
                                   mere existence is enough to cause <.netrc>
                                   to be ignored.
        b:netrw_lastfile        Holds latest method/machine/path.
        b:netrw_line            Holds current line number     (during NetWrite)
        g:netrw_silent          =0 transfers done normally
                                =1 transfers done silently
        g:netrw_uid             Holds current user-id for ftp.
        g:netrw_use_nt_rcp      =0 don't use WinNT/2K/XP's rcp (default)
                                =1 use WinNT/2K/XP's rcp, binary mode
        g:netrw_win95ftp        =0 use unix-style ftp even if win95/98/ME/etc
                                =1 use default method to do ftp >
        ------------------------------------------------------------------------
<
                                                *netrw-internal-variables*
The script will also make use of the following variables internally, albeit
```

```
temporarily.
>
                            ------------------
                            Temporary Variables
                            ------------------
        Variable               Meaning
        --------               -----------------------------------
<
        b:netrw_method         Index indicating rcp/ftp+.netrc/ftp
        w:netrw_method         (same as b:netrw_method)
        g:netrw_machine        Holds machine name parsed from input
        b:netrw_fname          Holds filename being accessed >
        ------------------------------------------------------------
<
                                                        *netrw-protocol*

Netrw supports a number of protocols.  These protocols are invoked using the
variables listed below, and may be modified by the user.
>
                            ------------------------
                            Protocol Control Options
                            ------------------------
    Option            Type        Setting         Meaning
    ---------         --------    -------------   --------------------------
<    netrw_ftp         variable    =doesn't exist  userid set by "user userid"
                                   =0              userid set by "user userid"
                                   =1              userid set by "userid"
    NetReadFixup      function    =doesn't exist  no change
                                   =exists         Allows user to have files
                                                   read via ftp automatically
                                                   transformed however they wish
                                                   by NetReadFixup()
    g:netrw_dav_cmd      var   ="cadaver"      if cadaver  is executable
    g:netrw_dav_cmd      var   ="curl -o"      elseif curl is executable
    g:netrw_fetch_cmd    var   ="fetch -o"     if fetch is available
    g:netrw_ftp_cmd      var   ="ftp"
    g:netrw_http_cmd     var   ="fetch -o"     if      fetch is available
    g:netrw_http_cmd     var   ="wget -O"      else if wget  is available
    g:netrw_http_put_cmd var   ="curl -T"
    |g:netrw_list_cmd|    var   ="ssh USEPORT HOSTNAME ls -Fa"
    g:netrw_rcp_cmd      var   ="rcp"
    g:netrw_rsync_cmd    var   ="rsync -a"
    g:netrw_scp_cmd      var   ="scp -q"
    g:netrw_sftp_cmd     var   ="sftp" >
    ------------------------------------------------------------------------
<
                                                                *netrw-ftp*

The g:netrw_..._cmd options (|g:netrw_ftp_cmd| and |g:netrw_sftp_cmd|)
specify the external program to use handle the ftp protocol.  They may
include command line options (such as -p for passive mode). Example: >

        let g:netrw_ftp_cmd= "ftp -p"
<
Browsing is supported by using the |g:netrw_list_cmd|; the substring
"HOSTNAME" will be changed via substitution with whatever the current request
is for a hostname.

Two options (|g:netrw_ftp| and |netrw-fixup|) both help with certain ftp's
that give trouble .  In order to best understand how to use these options if
ftp is giving you troubles, a bit of discussion is provided on how netrw does
ftp reads.
```

```
For ftp, netrw typically builds up lines of one of the following formats in a
temporary file:
>
  IF g:netrw_ftp !exists or is not 1     IF g:netrw_ftp exists and is 1
  ----------------------------------     -----------------------------
<
       open machine [port]                    open machine [port]
       user userid password                   userid password
       [g:netrw_ftpmode]                      password
       [g:netrw_ftpextracmd]                  [g:netrw_ftpmode]
       get filename tempfile                  [g:netrw_extracmd]
                                              get filename tempfile >
  ------------------------------------------------------------------------
<
The |g:netrw_ftpmode| and |g:netrw_ftpextracmd| are optional.

Netrw then executes the lines above by use of a filter:
>
        :%! {g:netrw_ftp_cmd} -i [-n]
<
where
        g:netrw_ftp_cmd is usually "ftp",
        -i tells ftp not to be interactive
        -n means don't use netrc and is used for Method #3 (ftp w/o <.netrc>)

If <.netrc> exists it will be used to avoid having to query the user for
userid and password.  The transferred file is put into a temporary file.
The temporary file is then read into the main editing session window that
requested it and the temporary file deleted.

If your ftp doesn't accept the "user" command and immediately just demands a
userid, then try putting "let netrw_ftp=1" in your <.vimrc>.

                                                                *netrw-cadaver*
To handle the SSL certificate dialog for untrusted servers, one may pull
down the certificate and place it into /usr/ssl/cert.pem.  This operation
renders the server treatment as "trusted".

                                                  *netrw-fixup* *netreadfixup*
If your ftp for whatever reason generates unwanted lines (such as AUTH
messages) you may write a NetReadFixup() function:
>
    function! NetReadFixup(method,line1,line2)
      " a:line1: first new line in current file
      " a:line2: last  new line in current file
      if     a:method == 1 "rcp
      elseif a:method == 2 "ftp + <.netrc>
      elseif a:method == 3 "ftp + machine,uid,password,filename
      elseif a:method == 4 "scp
      elseif a:method == 5 "http/wget
      elseif a:method == 6 "dav/cadaver
      elseif a:method == 7 "rsync
      elseif a:method == 8 "fetch
      elseif a:method == 9 "sftp
      else               " complain
      endif
    endfunction
>
The NetReadFixup() function will be called if it exists and thus allows you to
customize your reading process.  As a further example, <netrw.vim> contains
just such a function to handle Windows 95 ftp.  For whatever reason, Windows
```

```
95's ftp dumps four blank lines at the end of a transfer, and so it is
desirable to automate their removal.  Here's some code taken from <netrw.vim>
itself:
>
    if has("win95") && g:netrw_win95ftp
     fun! NetReadFixup(method, line1, line2)
       if method == 3   " ftp (no <.netrc>)
        let fourblanklines= line2 - 3
        silent fourblanklines.",".line2."g/^\s*/d"
       endif
     endfunction
    endif
>
(Related topics: |ftp| |netrw-userpass| |netrw-start|)

==============================================================================
9. Browsing             *netrw-browsing* *netrw-browse* *netrw-help* {{{1
                        *netrw-browser*  *netrw-dir*    *netrw-list*

INTRODUCTION TO BROWSING                          *netrw-intro-browse* {{{2
        (Quick References: |netrw-quickmaps| |netrw-quickcoms|)

Netrw supports the browsing of directories on your local system and on remote
hosts; browsing includes listing files and directories, entering directories,
editing files therein, deleting files/directories, making new directories,
moving (renaming) files and directories, copying files and directories, etc.
One may mark files and execute any system command on them!  The Netrw browser
generally implements the previous explorer's maps and commands for remote
directories, although details (such as pertinent global variable names)
necessarily differ.  To browse a directory, simply "edit" it! >

        vim /your/directory/
        vim .
        vim c:\your\directory\
<
(Related topics: |netrw-cr|  |netrw-o|  |netrw-p| |netrw-P| |netrw-t|
                 |netrw-mf|  |netrw-mx| |netrw-D| |netrw-R| |netrw-v| )

The Netrw remote file and directory browser handles two protocols: ssh and
ftp.  The protocol in the url, if it is ftp, will cause netrw also to use ftp
in its remote browsing.  Specifying any other protocol will cause it to be
used for file transfers; but the ssh protocol will be used to do remote
browsing.

To use Netrw's remote directory browser, simply attempt to read a "file" with
a trailing slash and it will be interpreted as a request to list a directory:
>
        vim [protocol]://[user@]hostname/path/
<
where [protocol] is typically scp or ftp.  As an example, try: >

        vim ftp://ftp.home.vim.org/pub/vim/
<
For local directories, the trailing slash is not required.  Again, because it's
easy to miss: to browse remote directories, the url must terminate with a
slash!

If you'd like to avoid entering the password repeatedly for remote directory
listings with ssh or scp, see |netrw-ssh-hack|.  To avoid password entry with
ftp, see |netrw-netrc| (if your ftp supports it).

There are several things you can do to affect the browser's display of files:
```

```
        * To change the listing style, press the "i" key (|netrw-i|).
          Currently there are four styles: thin, long, wide, and tree.
          To make that change "permanent", see |g:netrw_liststyle|.

        * To hide files (don't want to see those xyz~ files anymore?) see
          |netrw-ctrl-h|.

        * Press s to sort files by name, time, or size.

See |netrw-browse-cmds| for all the things you can do with netrw!

                        *netrw-getftype* *netrw-filigree* *netrw-ftype*
The |getftype()| function is used to append a bit of filigree to indicate
filetype to locally listed files:

        directory  : /
        executable : *
        fifo       : |
        links      : @
        sockets    : =

The filigree also affects the |g:netrw_sort_sequence|.


QUICK HELP                                              *netrw-quickhelp* {{{2
                    (Use ctrl-] to select a topic)~
        Intro to Browsing.............................|netrw-intro-browse|
          Quick Reference: Maps.......................|netrw-quickmap|
          Quick Reference: Commands...................|netrw-browse-cmds|
        Hiding
          Edit hiding list............................|netrw-ctrl-h|
          Hiding Files or Directories.................|netrw-a|
          Hiding/Unhiding by suffix...................|netrw-mh|
          Hiding  dot-files...........................|netrw-gh|
        Listing Style
          Select listing style (thin/long/wide/tree)....|netrw-i|
          Associated setting variable...................|g:netrw_liststyle|
          Shell command used to perform listing.........|g:netrw_list_cmd|
          Quick file info...............................|netrw-qf|
        Sorted by
          Select sorting style (name/time/size).........|netrw-s|
          Editing the sorting sequence..................|netrw-S|
          Sorting options...............................|g:netrw_sort_options|
          Associated setting variable...................|g:netrw_sort_sequence|
          Reverse sorting order.........................|netrw-r|


                                *netrw-quickmap* *netrw-quickmaps*
QUICK REFERENCE: MAPS                          *netrw-browse-maps* {{{2
>
          ---                   ----------------                        ----
          Map                   Quick Explanation                       Link
          ---                   ----------------                        ----
<         <F1>   Causes Netrw to issue help
          <cr>   Netrw will enter the directory or read the file      |netrw-cr|
          <del>  Netrw will attempt to remove the file/directory      |netrw-del|
          <c-h>  Edit file hiding list                                |netrw-ctrl-h|
          <c-l>  Causes Netrw to refresh the directory listing        |netrw-ctrl-l|
          <c-r>  Browse using a gvim server                           |netrw-ctrl-r|
          <c-tab> Shrink/expand a netrw/explore window                |netrw-c-tab|
            -    Makes Netrw go up one directory                      |netrw--|
```

```
   a    Toggles between normal display,                      |netrw-a|
        hiding (suppress display of files matching g:netrw_list_hide)
        showing (display only files which match g:netrw_list_hide)
   c    Make browsing directory the current directory        |netrw-c|
   C    Setting the editing window                           |netrw-C|
   d    Make a directory                                     |netrw-d|
   D    Attempt to remove the file(s)/directory(ies)         |netrw-D|
   gb   Go to previous bookmarked directory                  |netrw-gb|
   gd   Force treatment as directory                         |netrw-gd|
   gf   Force treatment as file                              |netrw-gf|
   gh   Quick hide/unhide of dot-files                       |netrw-gh|
   gn   Make top of tree the directory below the cursor      |netrw-gn|
   i    Cycle between thin, long, wide, and tree listings    |netrw-i|
   mb   Bookmark current directory                           |netrw-mb|
   mc   Copy marked files to marked-file target directory    |netrw-mc|
   md   Apply diff to marked files (up to 3)                 |netrw-md|
   me   Place marked files on arg list and edit them         |netrw-me|
   mf   Mark a file                                          |netrw-mf|
   mF   Unmark files                                         |netrw-mF|
   mg   Apply vimgrep to marked files                        |netrw-mg|
   mh   Toggle marked file suffices' presence on hiding list |netrw-mh|
   mm   Move marked files to marked-file target directory    |netrw-mm|
   mp   Print marked files                                   |netrw-mp|
   mr   Mark files using a shell-style |regexp|               |netrw-mr|
   mt   Current browsing directory becomes markfile target   |netrw-mt|
   mT   Apply ctags to marked files                          |netrw-mT|
   mu   Unmark all marked files                              |netrw-mu|
   mv   Apply arbitrary vim   command to marked files        |netrw-mv|
   mx   Apply arbitrary shell command to marked files        |netrw-mx|
   mX   Apply arbitrary shell command to marked files en bloc|netrw-mX|
   mz   Compress/decompress marked files                     |netrw-mz|
   o    Enter the file/directory under the cursor in a new   |netrw-o|
        browser window.  A horizontal split is used.
   O    Obtain a file specified by cursor                    |netrw-O|
   p    Preview the file                                     |netrw-p|
   P    Browse in the previously used window                 |netrw-P|
   qb   List bookmarked directories and history              |netrw-qb|
   qf   Display information on file                          |netrw-qf|
   qF   Mark files using a quickfix list                     |netrw-qF|
   qL   Mark files using a |location-list|                     |netrw-qL|
   r    Reverse sorting order                                |netrw-r|
   R    Rename the designated file(s)/directory(ies)         |netrw-R|
   s    Select sorting style: by name, time, or file size    |netrw-s|
   S    Specify suffix priority for name-sorting             |netrw-S|
   t    Enter the file/directory under the cursor in a new tab|netrw-t|
   u    Change to recently-visited directory                 |netrw-u|
   U    Change to subsequently-visited directory             |netrw-U|
   v    Enter the file/directory under the cursor in a new   |netrw-v|
        browser window.  A vertical split is used.
   x    View file with an associated program                 |netrw-x|
   X    Execute filename under cursor via |system()|          |netrw-X|

   %    Open a new file in netrw's current directory         |netrw-%|

*netrw-mouse* *netrw-leftmouse* *netrw-middlemouse* *netrw-rightmouse*
<leftmouse>     (gvim only) selects word under mouse as if a <cr>
                had been pressed (ie. edit file, change directory)
<middlemouse>   (gvim only) same as P selecting word under mouse;
                see |netrw-P|
<rightmouse>    (gvim only) delete file/directory using word under
                mouse
<2-leftmouse>   (gvim only) when:
```

```
                         * in a netrw-selected file, AND
                         * |g:netrw_retmap| == 1       AND
                         * the user doesn't already have a <2-leftmouse>
                           mapping defined before netrw is autoloaded,
                        then a double clicked leftmouse button will return
                        to the netrw browser window.  See |g:netrw_retmap|.
        <s-leftmouse>   (gvim only) like mf, will mark files.  Dragging
                        the shifted leftmouse will mark multiple files.
                        (see |netrw-mf|)

        (to disable mouse buttons while browsing: |g:netrw_mousemaps|)

                                *netrw-quickcom* *netrw-quickcoms*
QUICK REFERENCE: COMMANDS       *netrw-explore-cmds* *netrw-browse-cmds* {{{2
     :NetrwClean[!]..........................................|netrw-clean|
     :NetrwSettings..........................................|netrw-settings|
     :Ntree..................................................|netrw-ntree|
     :Explore[!]  [dir] Explore directory of current file.....|netrw-explore|
     :Hexplore[!] [dir] Horizontal Split & Explore............|netrw-explore|
     :Lexplore[!] [dir] Left Explorer Toggle..................|netrw-explore|
     :Nexplore[!] [dir] Vertical Split & Explore..............|netrw-explore|
     :Pexplore[!] [dir] Vertical Split & Explore..............|netrw-explore|
     :Rexplore          Return to Explorer....................|netrw-explore|
     :Sexplore[!] [dir] Split & Explore directory ............|netrw-explore|
     :Texplore[!] [dir] Tab & Explore........................|netrw-explore|
     :Vexplore[!] [dir] Vertical Split & Explore..............|netrw-explore|


BANNER DISPLAY                                          *netrw-I*

One may toggle the banner display on and off by pressing "I".

Also See: |g:netrw_banner|


BOOKMARKING A DIRECTORY *netrw-mb* *netrw-bookmark* *netrw-bookmarks* {{{2

One may easily "bookmark" the currently browsed directory by using >

        mb
<
                                                                *.netrwbook*
Bookmarks are retained in between sessions in a $HOME/.netrwbook file, and are
kept in sorted order.

If there are marked files and/or directories, mb will add them to the bookmark
list.

*netrw-:NetrwMB*
Addtionally, one may use :NetrwMB to bookmark files or directories. >

        :NetrwMB[!] [files/directories]

< No bang: enters files/directories into Netrw's bookmark system

   No argument and in netrw buffer:
     if there are marked files        : bookmark marked files
     otherwise                        : bookmark file/directory under cursor
   No argument and not in netrw buffer: bookmarks current open file
   Has arguments                      : |glob()|s each arg and bookmarks them

 With bang: deletes files/directories from Netrw's bookmark system
```

The :NetrwMB command is available outside of netrw buffers (once netrw has been invoked in the session).

The file ".netrwbook" holds bookmarks when netrw (and vim) is not active. By default, it's stored on the first directory on the user's |'runtimepath'|.

Related Topics:
|netrw-gb| how to return (go) to a bookmark
|netrw-mB| how to delete bookmarks
|netrw-qb| how to list bookmarks
|g:netrw_home| controls where .netrwbook is kept

BROWSING *netrw-enter* *netrw-cr* {{{2

Browsing is simple: move the cursor onto a file or directory of interest. Hitting the <cr> (the return key) will select the file or directory. Directories will themselves be listed, and files will be opened using the protocol given in the original read request.

CAVEAT: There are four forms of listing (see |netrw-i|). Netrw assumes that two or more spaces delimit filenames and directory names for the long and wide listing formats. Thus, if your filename or directory name has two or more sequential spaces embedded in it, or any trailing spaces, then you'll need to use the "thin" format to select it.

The |g:netrw_browse_split| option, which is zero by default, may be used to cause the opening of files to be done in a new window or tab instead of the default. When the option is one or two, the splitting will be taken horizontally or vertically, respectively. When the option is set to three, a <cr> will cause the file to appear in a new tab.

When using the gui (gvim), one may select a file by pressing the <leftmouse> button. In addition, if

* |g:netrw_retmap| == 1 AND (its default value is 0)
* in a netrw-selected file, AND
* the user doesn't already have a <2-leftmouse> mapping defined before netrw is loaded

then a doubly-clicked leftmouse button will return to the netrw browser window.

Netrw attempts to speed up browsing, especially for remote browsing where one may have to enter passwords, by keeping and re-using previously obtained directory listing buffers. The |g:netrw_fastbrowse| variable is used to control this behavior; one may have slow browsing (no buffer re-use), medium speed browsing (re-use directory buffer listings only for remote directories), and fast browsing (re-use directory buffer listings as often as possible). The price for such re-use is that when changes are made (such as new files are introduced into a directory), the listing may become out-of-date. One may always refresh directory listing buffers by pressing ctrl-L (see |netrw-ctrl-l|).

netrw-s-cr
Squeezing the Current Tree-Listing Directory~

When the tree listing style is enabled (see |netrw-i|) and one is using gvim, then the <s-cr> mapping may be used to squeeze (close) the directory currently containing the cursor.

```
Otherwise, one may remap a key combination of one's own choice to get
this effect: >

    nmap <buffer> <silent> <nowait> YOURKEYCOMBO  <Plug>NetrwTreeSqueeze
<
Put this line in $HOME/ftplugin/netrw/netrw.vim; it needs to be generated
for netrw buffers only.

Related topics:
        |netrw-ctrl-r|  |netrw-o|       |netrw-p|
        |netrw-P|       |netrw-t|       |netrw-v|
Associated setting variables:
   |g:netrw_browse_split|       |g:netrw_fastbrowse|
   |g:netrw_ftp_list_cmd|       |g:netrw_ftp_sizelist_cmd|
   |g:netrw_ftp_timelist_cmd|   |g:netrw_ssh_browse_reject|
   |g:netrw_ssh_cmd|            |g:netrw_use_noswf|


BROWSING WITH A HORIZONTALLY SPLIT WINDOW       *netrw-o* *netrw-horiz* {{{2

Normally one enters a file or directory using the <cr>.  However, the "o" map
allows one to open a new window to hold the new directory listing or file.  A
horizontal split is used.  (for vertical splitting, see |netrw-v|)

Normally, the o key splits the window horizontally with the new window and
cursor at the top.

Associated setting variables: |g:netrw_alto| |g:netrw_winsize|

Related topics:
        |netrw-ctrl-r|  |netrw-o|       |netrw-p|
        |netrw-P|       |netrw-t|       |netrw-v|
Associated setting variables:
   |g:netrw_alto|    control above/below splitting
   |g:netrw_winsize| control initial sizing

BROWSING WITH A NEW TAB                         *netrw-t* {{{2

Normally one enters a file or directory using the <cr>.  The "t" map
allows one to open a new window holding the new directory listing or file in
a new tab.

If you'd like to have the new listing in a background tab, use |gT|.

Related topics:
        |netrw-ctrl-r|  |netrw-o|       |netrw-p|
        |netrw-P|       |netrw-t|       |netrw-v|
Associated setting variables:
   |g:netrw_winsize| control initial sizing

BROWSING WITH A VERTICALLY SPLIT WINDOW                 *netrw-v* {{{2

Normally one enters a file or directory using the <cr>.  However, the "v" map
allows one to open a new window to hold the new directory listing or file.  A
vertical split is used.  (for horizontal splitting, see |netrw-o|)

Normally, the v key splits the window vertically with the new window and
cursor at the left.

There is only one tree listing buffer; using "v" on a displayed subdirectory
will split the screen, but the same buffer will be shown twice.
```

```
Related topics:
        |netrw-ctrl-r|  |netrw-o|       |netrw-p|
        |netrw-P|       |netrw-t|       |netrw-v|
Associated setting variables:
   |g:netrw_altv|    control right/left splitting
   |g:netrw_winsize| control initial sizing


BROWSING USING A GVIM SERVER                    *netrw-ctrl-r* {{{2

One may keep a browsing gvim separate from the gvim being used to edit.
Use the <c-r> map on a file (not a directory) in the netrw browser, and it
will use a gvim server (see |g:netrw_servername|).  Subsequent use of <cr>
(see |netrw-cr|) will re-use that server for editing files.

Related topics:
        |netrw-ctrl-r|  |netrw-o|       |netrw-p|
        |netrw-P|       |netrw-t|       |netrw-v|
Associated setting variables:
        |g:netrw_servername|   : sets name of server
        |g:netrw_browse_split| : controls how <cr> will open files


CHANGE LISTING STYLE  (THIN LONG WIDE TREE)                    *netrw-i* {{{2

The "i" map cycles between the thin, long, wide, and tree listing formats.

The thin listing format gives just the files' and directories' names.

The long listing is either based on the "ls" command via ssh for remote
directories or displays the filename, file size (in bytes), and the time and
date of last modification for local directories.  With the long listing
format, netrw is not able to recognize filenames which have trailing spaces.
Use the thin listing format for such files.

The wide listing format uses two or more contiguous spaces to delineate
filenames; when using that format, netrw won't be able to recognize or use
filenames which have two or more contiguous spaces embedded in the name or any
trailing spaces.  The thin listing format will, however, work with such files.
The wide listing format is the most compact.

The tree listing format has a top directory followed by files and directories
preceded by one or more "|"s, which indicate the directory depth.  One may
open and close directories by pressing the <cr> key while atop the directory
name.

One may make a preferred listing style your default; see |g:netrw_liststyle|.
As an example, by putting the following line in your .vimrc, >
        let g:netrw_liststyle= 3
the tree style will become your default listing style.

One typical way to use the netrw tree display is to: >

        vim .
        (use i until a tree display shows)
        navigate to a file
        v  (edit as desired in vertically split window)
        ctrl-w h  (to return to the netrw listing)
        P (edit newly selected file in the previous window)
        ctrl-w h  (to return to the netrw listing)
        P (edit newly selected file in the previous window)
```

```
        ...etc...
<
Associated setting variables: |g:netrw_liststyle| |g:netrw_maxfilenamelen|
                              |g:netrw_timefmt|   |g:netrw_list_cmd|

CHANGE FILE PERMISSION                                          *netrw-gp* {{{2

"gp" will ask you for a new permission for the file named under the cursor.
Currently, this only works for local files.

Associated setting variables: |g:netrw_chgperm|


CHANGING TO A BOOKMARKED DIRECTORY                      *netrw-gb*  {{{2

To change directory back to a bookmarked directory, use

        {cnt}gb

Any count may be used to reference any of the bookmarks.
Note that |netrw-qb| shows both bookmarks and history; to go
to a location stored in the history see |netrw-u| and |netrw-U|.

Related Topics:
        |netrw-mB| how to delete bookmarks
        |netrw-mb| how to make a bookmark
        |netrw-qb| how to list bookmarks


CHANGING TO A PREDECESSOR DIRECTORY             *netrw-u* *netrw-updir* {{{2

Every time you change to a new directory (new for the current session),
netrw will save the directory in a recently-visited directory history
list (unless |g:netrw_dirhistmax| is zero; by default, it's ten).  With the
"u" map, one can change to an earlier directory (predecessor).  To do
the opposite, see |netrw-U|.

The "u" map also accepts counts to go back in the history several slots.
For your convenience, qb (see |netrw-qb|) lists the history number which may
be used in that count.

                                                *.netrwhist*
See |g:netrw_dirhistmax| for how to control the quantity of history stack
slots.  The file ".netrwhist" holds history when netrw (and vim) is not
active.  By default, it's stored on the first directory on the user's
|'runtimepath'|.

Related Topics:
        |netrw-U| changing to a successor directory
        |g:netrw_home| controls where .netrwhist is kept


CHANGING TO A SUCCESSOR DIRECTORY               *netrw-U* *netrw-downdir* {{{2

With the "U" map, one can change to a later directory (successor).
This map is the opposite of the "u" map. (see |netrw-u|)  Use the
qb map to list both the bookmarks and history. (see |netrw-qb|)

The "U" map also accepts counts to go forward in the history several slots.

See |g:netrw_dirhistmax| for how to control the quantity of history stack
slots.
```

```
CHANGING TREE TOP                       *netrw-ntree*  *:Ntree*  *netrw-gn* {{{2

One may specify a new tree top for tree listings using >

        :Ntree [dirname]

Without a "dirname", the current line is used (and any leading depth
information is elided).
With a "dirname", the specified directory name is used.

The "gn" map will take the word below the cursor and use that for
changing the top of the tree listing.


NETRW CLEAN                                        *netrw-clean* *:NetrwClean* {{{2

With NetrwClean one may easily remove netrw from one's home directory;
more precisely, from the first directory on your |'runtimepath'|.

With NetrwClean!, netrw will attempt to remove netrw from all directories on
your |'runtimepath'|.  Of course, you have to have write/delete permissions
correct to do this.

With either form of the command, netrw will first ask for confirmation
that the removal is in fact what you want to do.  If netrw doesn't have
permission to remove a file, it will issue an error message.

                                                   *netrw-gx*
CUSTOMIZING BROWSING WITH A SPECIAL HANDLER         *netrw-x* *netrw-handler* {{{2
                                                   (also see |netrw_filehandler|)

Certain files, such as html, gif, jpeg, (word/office) doc, etc, files, are
best seen with a special handler (ie. a tool provided with your computer's
operating system).  Netrw allows one to invoke such special handlers by: >

        * when Exploring, hit the "x" key
        * when editing, hit gx with the cursor atop the special filename
<         (latter not available if the |g:netrw_nogx| variable exists)

Netrw determines which special handler by the following method:

  * if |g:netrw_browsex_viewer| exists, then it will be used to attempt to
    view files.  Examples of useful settings (place into your <.vimrc>): >

        :let g:netrw_browsex_viewer= "kfmclient exec"
<   or >
        :let g:netrw_browsex_viewer= "xdg-open"
<
    If g:netrw_browsex_viewer == '-', then netrwFileHandlers#Invoke() will be
    used instead (see |netrw_filehandler|).

  * for Windows 32 or 64, the url and FileProtocolHandler dlls are used.
  * for Gnome (with gnome-open): gnome-open is used.
  * for KDE (with kfmclient)   : kfmclient is used
  * for Mac OS X               : open is used.
  * otherwise the netrwFileHandler plugin is used.

The file's suffix is used by these various approaches to determine an
appropriate application to use to "handle" these files.  Such things as
OpenOffice (*.sfx), visualization (*.jpg, *.gif, etc), and PostScript (*.ps,
```

```
*.eps) can be handled.

The gx mapping extends to all buffers; apply "gx" while atop a word and netrw
will apply a special handler to it (like "x" works when in a netrw buffer).
One may also use visual mode (see |visual-start|) to select the text that the
special handler will use.  Normally gx uses expand("<cfile>") to pick up the
text under the cursor; one may change what |expand()| uses via the
|g:netrw_gx| variable.  Alternatively, one may select the text to be used by
gx via first making a visual selection (see |visual-block|) or by changing
the |'isfname'| option (which is global, so netrw doesn't modify it).

Associated setting variables:
        |g:netrw_gx|    control how gx picks up the text under the cursor
        |g:netrw_nogx|  prevent gx map while editing
        |g:netrw_suppress_gx_mesg| controls gx's suppression of browser messages

                                                        *netrw_filehandler*

When |g:netrw_browsex_viewer| exists and is "-", then netrw will attempt to
handle the special file with a vim function.  The "x" map applies a function
to a file, based on its extension.  Of course, the handler function must exist
for it to be called!
>
 Ex. mypgm.html   x -> NFH_html("scp://user@host/some/path/mypgm.html")

<       Users may write their own netrw File Handler functions to
        support more suffixes with special handling.  See
        <autoload/netrwFileHandlers.vim> for examples on how to make
        file handler functions.   As an example: >

        " NFH_suffix(filename)
        fun! NFH_suffix(filename)
        ..do something special with filename..
        endfun
<
These functions need to be defined in some file in your .vim/plugin
(vimfiles\plugin) directory.  Vim's function names may not have punctuation
characters (except for the underscore) in them.  To support suffices that
contain such characters, netrw will first convert the suffix using the
following table: >

    @ -> AT       ! -> EXCLAMATION    % -> PERCENT
    : -> COLON    = -> EQUAL          ? -> QUESTION
    , -> COMMA    - -> MINUS          ; -> SEMICOLON
    $ -> DOLLAR   + -> PLUS           ~ -> TILDE
<
So, for example: >

        file.rcs,v  ->  NFH_rcsCOMMAv()
<
If more such translations are necessary, please send me email: >
               NdrOchip at ScampbellPfamily.AbizM - NOSPAM
with a request.

Associated setting variable: |g:netrw_browsex_viewer|

                                                        *netrw-curdir*
DELETING BOOKMARKS                                      *netrw-mB* {{{2

To delete a bookmark, use >

        {cnt}mB
```

```
If there are marked files, then mB will remove them from the
bookmark list.

Alternatively, one may use :NetrwMB! (see |netrw-:NetrwMB|). >

        :NetrwMB! [files/directories]

Related Topics:
        |netrw-gb| how to return (go) to a bookmark
        |netrw-mb| how to make a bookmark
        |netrw-qb| how to list bookmarks


DELETING FILES OR DIRECTORIES   *netrw-delete* *netrw-D* *netrw-del* {{{2

If files have not been marked with |netrw-mf|:   (local marked file list)

    Deleting/removing files and directories involves moving the cursor to the
    file/directory to be deleted and pressing "D".  Directories must be empty
    first before they can be successfully removed.  If the directory is a
    softlink to a directory, then netrw will make two requests to remove the
    directory before succeeding.  Netrw will ask for confirmation before doing
    the removal(s).  You may select a range of lines with the "V" command
    (visual selection), and then pressing "D".

If files have been marked with |netrw-mf|:   (local marked file list)

    Marked files (and empty directories) will be deleted; again, you'll be
    asked to confirm the deletion before it actually takes place.

A further approach is to delete files which match a pattern.

    * use  :MF pattern  (see |netrw-:MF|); then press "D".

    * use mr (see |netrw-mr|) which will prompt you for pattern.
      This will cause the matching files to be marked.  Then,
      press "D".

The |g:netrw_rm_cmd|, |g:netrw_rmf_cmd|, and |g:netrw_rmdir_cmd| variables are
used to control the attempts to remove remote files and directories.  The
g:netrw_rm_cmd is used with files, and its default value is:

        g:netrw_rm_cmd: ssh HOSTNAME rm

The g:netrw_rmdir_cmd variable is used to support the removal of directories.
Its default value is:

        |g:netrw_rmdir_cmd|: ssh HOSTNAME rmdir

If removing a directory fails with g:netrw_rmdir_cmd, netrw then will attempt
to remove it again using the g:netrw_rmf_cmd variable.  Its default value is:

        |g:netrw_rmf_cmd|: ssh HOSTNAME rm -f

Related topics: |netrw-d|
Associated setting variable: |g:netrw_localrmdir| |g:netrw_rm_cmd|
                             |g:netrw_rmdir_cmd|   |g:netrw_ssh_cmd|


*netrw-explore*  *netrw-hexplore* *netrw-nexplore* *netrw-pexplore*
*netrw-rexplore* *netrw-sexplore* *netrw-texplore* *netrw-vexplore* *netrw-lexplore*
```

```
DIRECTORY EXPLORATION COMMANDS  {{{2

     :[N]Explore[!]  [dir]... Explore directory of current file      *:Explore*
     :[N]Hexplore[!] [dir]... Horizontal Split & Explore             *:Hexplore*
     :[N]Lexplore[!] [dir]... Left Explorer Toggle                   *:Lexplore*
     :[N]Sexplore[!] [dir]... Split&Explore current file's directory *:Sexplore*
     :[N]Vexplore[!] [dir]... Vertical   Split & Explore             *:Vexplore*
     :Texplore       [dir]... Tab & Explore                          *:Texplore*
     :Rexplore            ... Return to/from Explorer                *:Rexplore*

     Used with :Explore **/pattern : (also see |netrw-starstar|)
     :Nexplore.............. go to next matching file                *:Nexplore*
     :Pexplore.............. go to previous matching file            *:Pexplore*

                                                  *netrw-:Explore*
:Explore  will open the local-directory browser on the current file's
          directory (or on directory [dir] if specified).  The window will be
          split only if the file has been modified and |'hidden'| is not set,
          otherwise the browsing window will take over that window.  Normally
          the splitting is taken horizontally.
          Also see: |netrw-:Rexplore|
:Explore! is like :Explore, but will use vertical splitting.

                                                  *netrw-:Hexplore*
:Hexplore  [dir] does an :Explore with |:belowright| horizontal splitting.
:Hexplore! [dir] does an :Explore with |:aboveleft|  horizontal splitting.

                                                  *netrw-:Lexplore*
:[N]Lexplore [dir] toggles a full height Explorer window on the left hand side
          of the current tab.  It will open a netrw window on the current
          directory if [dir] is omitted; a :Lexplore [dir] will show the
          specified directory in the left-hand side browser display no matter
          from which window the command is issued.

          By default, :Lexplore will change an uninitialized |g:netrw_chgwin|
          to 2; edits will thus preferentially be made in window#2.

          The [N] specifies a |g:netrw_winsize| just for the new :Lexplore
          window.

          Those who like this method often also often like tree style displays;
          see |g:netrw_liststyle|.

          Also see: |netrw-C|           |g:netrw_browse_split|   |g:netrw_wiw|
                    |netrw-p| |netrw-P|   |g:netrw_chgwin|
                    |netrw-c-tab|       |g:netrw_winsize|

:[N]Lexplore! is like :Lexplore, except that the full-height Explorer window
          will open on the right hand side and an uninitialized |g:netrw_chgwin|
          will be set to 1.

                                                  *netrw-:Sexplore*
:[N]Sexplore will always split the window before invoking the local-directory
          browser.  As with Explore, the splitting is normally done
          horizontally.
:[N]Sexplore! [dir] is like :Sexplore, but the splitting will be done vertically.

                                                  *netrw-:Texplore*
:Texplore  [dir] does a |:tabnew| before generating the browser window

                                                  *netrw-:Vexplore*
:[N]Vexplore  [dir] does an :Explore with |:leftabove|  vertical splitting.
```

```
:[N]Vexplore! [dir] does an :Explore with |:rightbelow| vertical splitting.

The optional parameters are:

 [N]: This parameter will override |g:netrw_winsize| to specify the quantity of
      rows and/or columns the new explorer window should have.
      Otherwise, the |g:netrw_winsize| variable, if it has been specified by the
      user, is used to control the quantity of rows and/or columns new
      explorer windows should have.

 [dir]: By default, these explorer commands use the current file's directory.
        However, one may explicitly provide a directory (path) to use instead;
        ie. >

        :Explore /some/path
<
                                                    *netrw-:Rexplore*
:Rexplore  This command is a little different from the other Explore commands
           as it doesn't necessarily open an Explorer window.

           Return to Explorer~
           When one edits a file using netrw which can occur, for example,
           when pressing <cr> while the cursor is atop a filename in a netrw
           browser window, a :Rexplore issued while editing that file will
           return the display to that of the last netrw browser display in
           that window.

           Return from Explorer~
           Conversely, when one is editing a directory, issuing a :Rexplore
           will return to editing the file that was last edited in that
           window.

           The <2-leftmouse> map (which is only available under gvim and
           cooperative terms) does the same as :Rexplore.

Also see: |g:netrw_alto| |g:netrw_altv| |g:netrw_winsize|


*netrw-star* *netrw-starpat* *netrw-starstar* *netrw-starstarpat* *netrw-grep*
EXPLORING WITH STARS AND PATTERNS {{{2

When Explore, Sexplore, Hexplore, or Vexplore are used with one of the
following four patterns Explore generates a list of files which satisfy the
request for the local file system.  These exploration patterns will not work
with remote file browsing.

    */filepat   files in current directory which satisfy filepat
    **/filepat  files in current directory or below which satisfy the
                file pattern
    *//pattern  files in the current directory which contain the
                pattern (vimgrep is used)
    **//pattern files in the current directory or below which contain
                the pattern (vimgrep is used)
<
The cursor will be placed on the first file in the list.  One may then
continue to go to subsequent files on that list via |:Nexplore| or to
preceding files on that list with |:Pexplore|.  Explore will update the
directory and place the cursor appropriately.

A plain >
        :Explore
will clear the explore list.
```

```
If your console or gui produces recognizable shift-up or shift-down sequences,
then you'll likely find using shift-downarrow and shift-uparrow convenient.
They're mapped by netrw as follows:

        <s-down>  == Nexplore, and
        <s-up>    == Pexplore.

As an example, consider
>
        :Explore */*.c
        :Nexplore
        :Nexplore
        :Pexplore
<
The status line will show, on the right hand side of the status line, a
message like "Match 3 of 20".

Associated setting variables:
        |g:netrw_keepdir|          |g:netrw_browse_split|
        |g:netrw_fastbrowse|       |g:netrw_ftp_browse_reject|
        |g:netrw_ftp_list_cmd|     |g:netrw_ftp_sizelist_cmd|
        |g:netrw_ftp_timelist_cmd| |g:netrw_list_cmd|
        |g:netrw_liststyle|


DISPLAYING INFORMATION ABOUT FILE                                *netrw-qf* {{{2

With the cursor atop a filename, pressing "qf" will reveal the file's size
and last modification timestamp.  Currently this capability is only available
for local files.


EDIT FILE OR DIRECTORY HIDING LIST      *netrw-ctrl-h* *netrw-edithide* {{{2

The "<ctrl-h>" map brings up a requestor allowing the user to change the
file/directory hiding list contained in |g:netrw_list_hide|.  The hiding list
consists of one or more patterns delimited by commas.  Files and/or
directories satisfying these patterns will either be hidden (ie. not shown) or
be the only ones displayed (see |netrw-a|).

The "gh" mapping (see |netrw-gh|) quickly alternates between the usual
hiding list and the hiding of files or directories that begin with ".".

As an example, >
        let g:netrw_list_hide= '\(^\|\s\s\)\zs\.\S\+'
Effectively, this makes the effect of a |netrw-gh| command the initial setting.
What it means:

        \(^\|\s\s\)   : if the line begins with the following, -or-
                        two consecutive spaces are encountered
        \zs           : start the hiding match now
        \.            : if it now begins with a dot
        \S\+          : and is followed by one or more non-whitespace
                        characters

Associated setting variables: |g:netrw_hide| |g:netrw_list_hide|
Associated topics: |netrw-a| |netrw-gh| |netrw-mh|

                                        *netrw-sort-sequence*
EDITING THE SORTING SEQUENCE            *netrw-S* *netrw-sortsequence* {{{2
```

```
When "Sorted by" is name, one may specify priority via the sorting sequence
(g:netrw_sort_sequence).  The sorting sequence typically prioritizes the
name-listing by suffix, although any pattern will do.  Patterns are delimited
by commas.  The default sorting sequence is (all one line):

For Unix: >
        '[\/]$,\<core\%(\.\d\+\)\=,\.[a-np-z]$,\.h$,\.c$,\.cpp$,*,\.o$,\.obj$,
        \.info$,\.swp$,\.bak$,\~$'
<
Otherwise: >
        '[\/]$,\.[a-np-z]$,\.h$,\.c$,\.cpp$,*,\.o$,\.obj$,\.info$,
        \.swp$,\.bak$,\~$'
<
The lone * is where all filenames not covered by one of the other patterns
will end up.  One may change the sorting sequence by modifying the
g:netrw_sort_sequence variable (either manually or in your <.vimrc>) or by
using the "S" map.

Related topics:               |netrw-s|               |netrw-S|
Associated setting variables: |g:netrw_sort_sequence| |g:netrw_sort_options|


EXECUTING FILE UNDER CURSOR VIA SYSTEM()                        *netrw-X* {{{2

Pressing X while the cursor is atop an executable file will yield a prompt
using the filename asking for any arguments.  Upon pressing a [return], netrw
will then call |system()| with that command and arguments.  The result will
be displayed by |:echomsg|, and so |:messages| will repeat display of the
result.  Ansi escape sequences will be stripped out.


FORCING TREATMENT AS A FILE OR DIRECTORY        *netrw-gd* *netrw-gf* {{{2

Remote symbolic links (ie. those listed via ssh or ftp) are problematic
in that it is difficult to tell whether they link to a file or to a
directory.

To force treatment as a file: use >
        gf
<
To force treatment as a directory: use >
        gd
<

GOING UP                                                        *netrw--* {{{2

To go up a directory, press "-" or press the <cr> when atop the ../ directory
entry in the listing.

Netrw will use the command in |g:netrw_list_cmd| to perform the directory
listing operation after changing HOSTNAME to the host specified by the
user-prpvided url.  By default netrw provides the command as: >

        ssh HOSTNAME ls -FLa
<
where the HOSTNAME becomes the [user@]hostname as requested by the attempt to
read.  Naturally, the user may override this command with whatever is
preferred.  The NetList function which implements remote browsing
expects that directories will be flagged by a trailing slash.


HIDING FILES OR DIRECTORIES                     *netrw-a* *netrw-hiding* {{{2
```

```
Netrw's browsing facility allows one to use the hiding list in one of three
ways: ignore it, hide files which match, and show only those files which
match.

If no files have been marked via |netrw-mf|:

The "a" map allows the user to cycle through the three hiding modes.

The |g:netrw_list_hide| variable holds a comma delimited list of patterns
based on regular expressions (ex. ^.*\.obj$,^\.) which specify the hiding list.
(also see |netrw-ctrl-h|)  To set the hiding list, use the <c-h> map.  As an
example, to hide files which begin with a ".", one may use the <c-h> map to
set the hiding list to '^\..*' (or one may put let g:netrw_list_hide= '^\..*'
in one's <.vimrc>).  One may then use the "a" key to show all files, hide
matching files, or to show only the matching files.

        Example: \.[ch]$
                This hiding list command will hide/show all *.c and *.h files.

        Example: \.c$,\.h$
                This hiding list command will also hide/show all *.c and *.h
                files.

Don't forget to use the "a" map to select the mode (normal/hiding/show) you
want!

If files have been marked using |netrw-mf|, then this command will:

  if showing all files or non-hidden files:
   modify the g:netrw_list_hide list by appending the marked files to it
   and showing only non-hidden files.

  else if showing hidden files only:
   modify the g:netrw_list_hide list by removing the marked files from it
   and showing only non-hidden files.
  endif

                                        *netrw-gh* *netrw-hide*
As a quick shortcut, one may press >
        gh
to toggle between hiding files which begin with a period (dot) and not hiding
them.

Associated setting variables: |g:netrw_list_hide|  |g:netrw_hide|
Associated topics: |netrw-a| |netrw-ctrl-h| |netrw-mh|

                                        *netrw-gitignore*
Netrw provides a helper function 'netrw_gitignore#Hide()' that, when used with
|g:netrw_list_hide| automatically hides all git-ignored files.

'netrw_gitignore#Hide' searches for patterns in the following files: >

        './.gitignore'
        './.git/info/exclude'
        global gitignore file: `git config --global core.excludesfile`
        system gitignore file: `git config --system core.excludesfile`
<
Files that do not exist, are ignored.
Git-ignore patterns are taken from existing files, and converted to patterns for
hiding files. For example, if you had '*.log' in your '.gitignore' file, it
would be converted to '.*\.log'.
```

```
To use this function, simply assign its output to |g:netrw_list_hide| option.  >

        Example: let g:netrw_list_hide= netrw_gitignore#Hide()
                Git-ignored files are hidden in Netrw.

        Example: let g:netrw_list_hide= netrw_gitignore#Hide('my_gitignore_file')
                Function can take additional files with git-ignore patterns.

        Example: g:netrw_list_hide= netrw_gitignore#Hide() . '.*\.swp$'
                Combining 'netrw_gitignore#Hide' with custom patterns.
<

IMPROVING BROWSING                      *netrw-listhack* *netrw-ssh-hack* {{{2

Especially with the remote directory browser, constantly entering the password
is tedious.

For Linux/Unix systems, the book "Linux Server Hacks - 100 industrial strength
tips & tools" by Rob Flickenger (O'Reilly, ISBN 0-596-00461-3) gives a tip
for setting up no-password ssh and scp and discusses associated security
issues.  It used to be available at http://hacks.oreilly.com/pub/h/66 ,
but apparently that address is now being redirected to some "hackzine".
I'll attempt a summary based on that article and on a communication from
Ben Schmidt:

        1. Generate a public/private key pair on the local machine
           (ssh client): >
                ssh-keygen -t rsa
                (saving the file in ~/.ssh/id_rsa as prompted)
<
        2. Just hit the <CR> when asked for passphrase (twice) for no
           passphrase.  If you do use a passphrase, you will also need to use
           ssh-agent so you only have to type the passphrase once per session.
           If you don't use a passphrase, simply logging onto your local
           computer or getting access to the keyfile in any way will suffice
           to access any ssh servers which have that key authorized for login.

        3. This creates two files: >
                ~/.ssh/id_rsa
                ~/.ssh/id_rsa.pub
<
        4. On the target machine (ssh server): >
                cd
                mkdir -p .ssh
                chmod 0700 .ssh
<
        5. On your local machine (ssh client): (one line) >
                ssh {serverhostname}
                  cat '>>' '~/.ssh/authorized_keys2' < ~/.ssh/id_rsa.pub
<
           or, for OpenSSH, (one line) >
                ssh {serverhostname}
                  cat '>>' '~/.ssh/authorized_keys' < ~/.ssh/id_rsa.pub
<
You can test it out with >
        ssh {serverhostname}
and you should be log onto the server machine without further need to type
anything.

If you decided to use a passphrase, do: >
        ssh-agent $SHELL
```

```
        ssh-add
        ssh {serverhostname}
You will be prompted for your key passphrase when you use ssh-add, but not
subsequently when you use ssh.  For use with vim, you can use >
        ssh-agent vim
and, when next within vim, use >
        :!ssh-add
Alternatively, you can apply ssh-agent to the terminal you're planning on
running vim in: >
        ssh-agent xterm &
and do ssh-add whenever you need.

For Windows, folks on the vim mailing list have mentioned that Pageant helps
with avoiding the constant need to enter the password.

Kingston Fung wrote about another way to avoid constantly needing to enter
passwords:

    In order to avoid the need to type in the password for scp each time, you
    provide a hack in the docs to set up a non password ssh account. I found a
    better way to do that: I can use a regular ssh account which uses a
    password to access the material without the need to key-in the password
    each time. It's good for security and convenience. I tried ssh public key
    authorization + ssh-agent, implementing this, and it works! Here are two
    links with instructions:

    http://www.ibm.com/developerworks/library/l-keyc2/
    http://sial.org/howto/openssh/publickey-auth/


    Ssh hints:

        Thomer Gil has provided a hint on how to speed up netrw+ssh:
            http://thomer.com/howtos/netrw_ssh.html

        Alex Young has several hints on speeding ssh up:
            http://usevim.com/2012/03/16/editing-remote-files/


LISTING BOOKMARKS AND HISTORY           *netrw-qb* *netrw-listbookmark* {{{2

Pressing "qb" (query bookmarks) will list both the bookmarked directories and
directory traversal history.

Related Topics:
        |netrw-gb| how to return (go) to a bookmark
        |netrw-mb| how to make a bookmark
        |netrw-mB| how to delete bookmarks
        |netrw-u|  change to a predecessor directory via the history stack
        |netrw-U|  change to a successor   directory via the history stack

MAKING A NEW DIRECTORY                                  *netrw-d* {{{2

With the "d" map one may make a new directory either remotely (which depends
on the global variable g:netrw_mkdir_cmd) or locally (which depends on the
global variable g:netrw_localmkdir).  Netrw will issue a request for the new
directory's name.  A bare <CR> at that point will abort the making of the
directory.  Attempts to make a local directory that already exists (as either
a file or a directory) will be detected, reported on, and ignored.

Related topics: |netrw-D|
Associated setting variables:   |g:netrw_localmkdir|   |g:netrw_mkdir_cmd|
```

|g:netrw_remote_mkdir| |netrw-%|

MAKING THE BROWSING DIRECTORY THE CURRENT DIRECTORY *netrw-c* {{{2

By default, |g:netrw_keepdir| is 1. This setting means that the current directory will not track the browsing directory. (done for backwards compatibility with v6's file explorer).

Setting g:netrw_keepdir to 0 tells netrw to make vim's current directory track netrw's browsing directory.

However, given the default setting for g:netrw_keepdir of 1 where netrw maintains its own separate notion of the current directory, in order to make the two directories the same, use the "c" map (just type c). That map will set Vim's notion of the current directory to netrw's current browsing directory.

Associated setting variable: |g:netrw_keepdir|

MARKING FILES *netrw-:MF* *netrw-mf* {{{2
(also see |netrw-mr|)

Netrw provides several ways to mark files:

* One may mark files with the cursor atop a filename and then pressing "mf".

* With gvim, in addition one may mark files with <s-leftmouse>. (see |netrw-mouse|)

* One may use the :MF command, which takes a list of files (for local directories, the list may include wildcards -- see |glob()|) >

```
:MF *.c
```
<

(Note that :MF uses |<f-args>| to break the line at spaces)

* Mark files using the |argument-list| (|netrw-mA|)

* Mark files based upon a |location-list| (|netrw-qL|)

* Mark files based upon the quickfix list (|netrw-qF|) (|quickfix-error-lists|)

The following netrw maps make use of marked files:

|netrw-a|	Hide marked files/directories
|netrw-D|	Delete marked files/directories
|netrw-ma|	Move marked files' names to |arglist|
|netrw-mA|	Move |arglist| filenames to marked file list
|netrw-mb|	Append marked files to bookmarks
|netrw-mB|	Delete marked files from bookmarks
|netrw-mc|	Copy marked files to target
|netrw-md|	Apply vimdiff to marked files
|netrw-me|	Edit marked files
|netrw-mF|	Unmark marked files
|netrw-mg|	Apply vimgrep to marked files
|netrw-mm|	Move marked files to target
|netrw-mp|	Print marked files

```
    |netrw-mt|  Set target for |netrw-mm| and |netrw-mc|
    |netrw-mT|  Generate tags using marked files
    |netrw-mv|  Apply vim command to marked files
    |netrw-mx|  Apply shell command to marked files
    |netrw-mX|  Apply shell command to marked files, en bloc
    |netrw-mz|  Compress/Decompress marked files
    |netrw-O|   Obtain marked files
    |netrw-R|   Rename marked files

One may unmark files one at a time the same way one marks them; ie. place
the cursor atop a marked file and press "mf".  This process also works
with <s-leftmouse> using gvim.  One may unmark all files by pressing
"mu" (see |netrw-mu|).

Marked files are highlighted using the "netrwMarkFile" highlighting group,
which by default is linked to "Identifier" (see Identifier under
|group-name|).  You may change the highlighting group by putting something
like >

        highlight clear netrwMarkFile
        hi link netrwMarkFile ..whatever..
<
into $HOME/.vim/after/syntax/netrw.vim .

If the mouse is enabled and works with your vim, you may use <s-leftmouse> to
mark one or more files.  You may mark multiple files by dragging the shifted
leftmouse.  (see |netrw-mouse|)

                        *markfilelist* *global_markfilelist* *local_markfilelist*
All marked files are entered onto the global marked file list; there is only
one such list.  In addition, every netrw buffer also has its own buffer-local
marked file list; since netrw buffers are associated with specific
directories, this means that each directory has its own local marked file
list.  The various commands which operate on marked files use one or the other
of the marked file lists.

Known Problem: if one is using tree mode (|g:netrw_liststyle|) and several
directories have files with the same name,  then marking such a file will
result in all such files being highlighted as if they were all marked.  The
|markfilelist|, however, will only have the selected file in it.  This problem
is unlikely to be fixed.


UNMARKING FILES                                                 *netrw-mF* {{{2
        (also see |netrw-mf|, |netrw-mu|)

The "mF" command will unmark all files in the current buffer.  One may also use
mf (|netrw-mf|) on a specific, already marked, file to unmark just that file.

MARKING FILES BY LOCATION LIST                                  *netrw-qL* {{{2
        (also see |netrw-mf|)

One may convert |location-list|s into a marked file list using "qL".
You may then proceed with commands such as me (|netrw-me|) to edit them.


MARKING FILES BY QUICKFIX LIST                                  *netrw-qF* {{{2
        (also see |netrw-mf|)

One may convert |quickfix-error-lists| into a marked file list using "qF".
You may then proceed with commands such as me (|netrw-me|) to edit them.
Quickfix error lists are generated, for example, by calls to |:vimgrep|.
```

MARKING FILES BY REGULAR EXPRESSION *netrw-mr* {{{2
(also see |netrw-mf|)

One may also mark files by pressing "mr"; netrw will then issue a prompt, "Enter regexp: ". You may then enter a shell-style regular expression such as *.c$ (see |glob()|). For remote systems, glob() doesn't work -- so netrw converts "*" into ".*" (see |regexp|) and marks files based on that. In the future I may make it possible to use |regexp|s instead of glob()-style expressions (yet-another-option).

MARKED FILES, ARBITRARY VIM COMMAND *netrw-mv* {{{2
(See |netrw-mf| and |netrw-mr| for how to mark files)
(uses the local marked-file list)

The "mv" map causes netrw to execute an arbitrary vim command on each file on the local marked file list, individually:

* 1split
* sil! keepalt e file
* run vim command
* sil! keepalt wq!

A prompt, "Enter vim command: ", will be issued to elicit the vim command you wish used.

MARKED FILES, ARBITRARY SHELL COMMAND *netrw-mx* {{{2
(See |netrw-mf| and |netrw-mr| for how to mark files)
(uses the local marked-file list)

Upon activation of the "mx" map, netrw will query the user for some (external) command to be applied to all marked files. All "%"s in the command will be substituted with the name of each marked file in turn. If no "%"s are in the command, then the command will be followed by a space and a marked filename.

Example:
(mark files)
mx
Enter command: cat

The result is a series of shell commands:
cat 'file1'
cat 'file2'
...

MARKED FILES, ARBITRARY SHELL COMMAND, EN BLOC *netrw-mX* {{{2
(See |netrw-mf| and |netrw-mr| for how to mark files)
(uses the global marked-file list)

Upon activation of the 'mX' map, netrw will query the user for some (external) command to be applied to all marked files on the global marked file list. The "en bloc" means that one command will be executed on all the files at once: >

command files

This approach is useful, for example, to select files and make a tarball: >

(mark files)

```
        mX
        Enter command: tar cf mynewtarball.tar
<
The command that will be run with this example:

        tar cf mynewtarball.tar 'file1' 'file2' ...


MARKED FILES: ARGUMENT LIST                              *netrw-ma* *netrw-mA*
            (See |netrw-mf| and |netrw-mr| for how to mark files)
                      (uses the global marked-file list)

Using ma, one moves filenames from the marked file list to the argument list.
Using mA, one moves filenames from the argument list to the marked file list.

See Also: |netrw-qF| |argument-list| |:args|


MARKED FILES: COMPRESSION AND DECOMPRESSION              *netrw-mz* {{{2
            (See |netrw-mf| and |netrw-mr| for how to mark files)
                      (uses the local marked file list)

If any marked files are compressed,   then "mz" will decompress them.
If any marked files are decompressed, then "mz" will compress them
using the command specified by |g:netrw_compress|; by default,
that's "gzip".

For decompression, netrw uses a |Dictionary| of suffices and their
associated decompressing utilities; see |g:netrw_decompress|.

Remember that one can mark multiple files by regular expression
(see |netrw-mr|); this is particularly useful to facilitate compressing and
decompressing a large number of files.

Associated setting variables: |g:netrw_compress| |g:netrw_decompress|

MARKED FILES: COPYING                                          *netrw-mc* {{{2
            (See |netrw-mf| and |netrw-mr| for how to mark files)
                      (Uses the global marked file list)

Select a target directory with mt (|netrw-mt|).  Then change directory,
select file(s) (see |netrw-mf|), and press "mc".  The copy is done
from the current window (where one does the mf) to the target.

If one does not have a target directory set with |netrw-mt|, then netrw
will query you for a directory to copy to.

One may also copy directories and their contents (local only) to a target
directory.

Associated setting variables:
        |g:netrw_localcopycmd|
        |g:netrw_localcopydircmd|
        |g:netrw_ssh_cmd|

MARKED FILES: DIFF                                             *netrw-md* {{{2
            (See |netrw-mf| and |netrw-mr| for how to mark files)
                      (uses the global marked file list)

Use |vimdiff| to visualize difference between selected files (two or
three may be selected for this).  Uses the global marked file list.
```

```
MARKED FILES: EDITING                                          *netrw-me* {{{2
            (See |netrw-mf| and |netrw-mr| for how to mark files)
                      (uses the global marked file list)

The "me" command will place the marked files on the |arglist| and commence
editing them.  One may return the to explorer window with |:Rexplore|.
(use |:n| and |:p| to edit next and previous files in the arglist)

MARKED FILES: GREP                                             *netrw-mg* {{{2
            (See |netrw-mf| and |netrw-mr| for how to mark files)
                      (uses the global marked file list)

The "mg" command will apply |:vimgrep| to the marked files.
The command will ask for the requested pattern; one may then enter: >

        /pattern/[g][j]
        ! /pattern/[g][j]
        pattern
<
With /pattern/, editing will start with the first item on the |quickfix| list
that vimgrep sets up (see |:copen|, |:cnext|, |:cprevious|, |:cclose|).  The
|:vimgrep|
command is in use, so without 'g' each line is added to quickfix list only
once; with 'g' every match is included.

With /pattern/j, "mg" will winnow the current marked file list to just those
marked files also possessing the specified pattern.  Thus, one may use >

        mr ...file-pattern...
        mg /pattern/j
<
to have a marked file list satisfying the file-pattern but also restricted to
files containing some desired pattern.


MARKED FILES: HIDING AND UNHIDING BY SUFFIX                    *netrw-mh* {{{2
            (See |netrw-mf| and |netrw-mr| for how to mark files)
                      (uses the local marked file list)

The "mh" command extracts the suffices of the marked files and toggles their
presence on the hiding list.  Please note that marking the same suffix
this way multiple times will result in the suffix's presence being toggled
for each file (so an even quantity of marked files having the same suffix
is the same as not having bothered to select them at all).

Related topics: |netrw-a| |g:netrw_list_hide|

MARKED FILES: MOVING                                           *netrw-mm* {{{2
            (See |netrw-mf| and |netrw-mr| for how to mark files)
                      (uses the global marked file list)

        WARNING: moving files is more dangerous than copying them.
        A file being moved is first copied and then deleted; if the
        copy operation fails and the delete succeeds, you will lose
        the file.  Either try things out with unimportant files
        first or do the copy and then delete yourself using mc and D.
        Use at your own risk!

Select a target directory with mt (|netrw-mt|).  Then change directory,
select file(s) (see |netrw-mf|), and press "mm".  The move is done
from the current window (where one does the mf) to the target.
```

```
Associated setting variable: |g:netrw_localmovecmd| |g:netrw_ssh_cmd|

MARKED FILES: PRINTING                                          *netrw-mp* {{{2
            (See |netrw-mf| and |netrw-mr| for how to mark files)
                      (uses the local marked file list)

When "mp" is used, netrw will apply the |:hardcopy| command to marked files.
What netrw does is open each file in a one-line window, execute hardcopy, then
close the one-line window.


MARKED FILES: SOURCING                                          *netrw-ms* {{{2
            (See |netrw-mf| and |netrw-mr| for how to mark files)
                      (uses the local marked file list)

With "ms", netrw will source the marked files (using vim's |:source| command)


MARKED FILES: SETTING THE TARGET DIRECTORY                      *netrw-mt* {{{2
     (See |netrw-mf| and |netrw-mr| for how to mark files)

Set the marked file copy/move-to target (see |netrw-mc| and |netrw-mm|):

  * If the cursor is atop a file name, then the netrw window's currently
    displayed directory is used for the copy/move-to target.

  * Also, if the cursor is in the banner, then the netrw window's currently
    displayed directory is used for the copy/move-to target.
    Unless the target already is the current directory.  In which case,
    typing "mf" clears the target.

  * However, if the cursor is atop a directory name, then that directory is
    used for the copy/move-to target

  * One may use the :MT [directory] command to set the target   *netrw-:MT*
    This command uses |<q-args>|, so spaces in the directory name are
    permitted without escaping.

  * With mouse-enabled vim or with gvim, one may select a target by using
    <c-leftmouse>

There is only one copy/move-to target at a time in a vim session; ie. the
target is a script variable (see |s:var|) and is shared between all netrw
windows (in an instance of vim).

When using menus and gvim, netrw provides a "Targets" entry which allows one
to pick a target from the list of bookmarks and history.

Related topics:
      Marking Files.....................................|netrw-mf|
      Marking Files by Regular Expression...............|netrw-mr|
      Marked Files: Target Directory Using Bookmarks.....|netrw-Tb|
      Marked Files: Target Directory Using History.......|netrw-Th|


MARKED FILES: TAGGING                                           *netrw-mT* {{{2
            (See |netrw-mf| and |netrw-mr| for how to mark files)
                      (uses the global marked file list)

The "mT" mapping will apply the command in |g:netrw_ctags| (by default, it is
"ctags") to marked files.  For remote browsing, in order to create a tags file
netrw will use ssh (see |g:netrw_ssh_cmd|), and so ssh must be available for
```

this to work on remote systems. For your local system, see |ctags| on how to
get a version. I myself use hdrtags, currently available at
http://www.drchip.org/astronaut/src/index.html , and have >

        let g:netrw_ctags= "hdrtag"
<
in my <.vimrc>.

When a remote set of files are tagged, the resulting tags file is "obtained";
ie. a copy is transferred to the local system's directory. The now local tags
file is then modified so that one may use it through the network. The
modification made concerns the names of the files in the tags; each filename is
preceded by the netrw-compatible url used to obtain it. When one subsequently
uses one of the go to tag actions (|tags|), the url will be used by netrw to
edit the desired file and go to the tag.

Associated setting variables: |g:netrw_ctags| |g:netrw_ssh_cmd|

MARKED FILES: TARGET DIRECTORY USING BOOKMARKS *netrw-Tb* {{{2

Sets the marked file copy/move-to target.

The |netrw-qb| map will give you a list of bookmarks (and history).
One may choose one of the bookmarks to become your marked file
target by using [count]Tb (default count: 1).

Related topics:
      Copying files to target.....|netrw-mc|
      Listing Bookmarks and History.....|netrw-qb|
      Marked Files: Setting The Target Directory.....|netrw-mt|
      Marked Files: Target Directory Using History.....|netrw-Th|
      Marking Files.....|netrw-mf|
      Marking Files by Regular Expression.....|netrw-mr|
      Moving files to target.....|netrw-mm|

MARKED FILES: TARGET DIRECTORY USING HISTORY *netrw-Th* {{{2

Sets the marked file copy/move-to target.

The |netrw-qb| map will give you a list of history (and bookmarks).
One may choose one of the history entries to become your marked file
target by using [count]Th (default count: 0; ie. the current directory).

Related topics:
      Copying files to target.....|netrw-mc|
      Listing Bookmarks and History.....|netrw-qb|
      Marked Files: Setting The Target Directory.....|netrw-mt|
      Marked Files: Target Directory Using Bookmarks.....|netrw-Tb|
      Marking Files.....|netrw-mf|
      Marking Files by Regular Expression.....|netrw-mr|
      Moving files to target.....|netrw-mm|

MARKED FILES: UNMARKING *netrw-mu* {{{2
     (See |netrw-mf|, |netrw-mF|)

The "mu" mapping will unmark all currently marked files. This command differs
from "mF" as the latter only unmarks files in the current directory whereas
"mu" will unmark global and all buffer-local marked files.
(see |netrw-mF|)

```
                                *netrw-browser-settings*
NETRW BROWSER VARIABLES         *netrw-browser-options* *netrw-browser-var* {{{2

(if you're interested in the netrw file transfer settings, see |netrw-options|
 and |netrw-protocol|)

The <netrw.vim> browser provides settings in the form of variables which
you may modify; by placing these settings in your <.vimrc>, you may customize
your browsing preferences.  (see also: |netrw-settings|)
>
   ---                          -----------
   Var                          Explanation
   ---                          -----------
<  *g:netrw_altfile*            some like |CTRL-^| to return to the last
                                edited file.  Choose that by setting this
                                parameter to 1.
                                Others like |CTRL-^| to return to the
                                netrw browsing buffer.  Choose that by setting
                                this parameter to 0.
                                 default: =0

  *g:netrw_alto*                change from above splitting to below splitting
                                by setting this variable (see |netrw-o|)
                                 default: =&sb           (see |'sb'|)

  *g:netrw_altv*                change from left splitting to right splitting
                                by setting this variable (see |netrw-v|)
                                 default: =&spr          (see |'spr'|)

  *g:netrw_banner*              enable/suppress the banner
                                =0: suppress the banner
                                =1: banner is enabled (default)

  *g:netrw_bannerbackslash*     if this variable exists and is not zero, the
                                banner will be displayed with backslashes
                                rather than forward slashes.

  *g:netrw_browse_split*        when browsing, <cr> will open the file by:
                                =0: re-using the same window  (default)
                                =1: horizontally splitting the window first
                                =2: vertically   splitting the window first
                                =3: open file in new tab
                                =4: act like "P" (ie. open previous window)
                                    Note that |g:netrw_preview| may be used
                                    to get vertical splitting instead of
                                    horizontal splitting.
                                =[servername,tab-number,window-number]
                                    Given a |List| such as this, a remote server
                                    named by the "servername" will be used for
                                    editing.  It will also use the specified tab
                                    and window numbers to perform editing
                                    (see |clientserver|, |netrw-ctrl-r|)
                                This option does not affect |:Lexplore|
                                windows.

                                Related topics:
                                    |g:netrw_alto|      |g:netrw_altv|
                                    |netrw-C|           |netrw-cr|
                                    |netrw-ctrl-r|

  *g:netrw_browsex_viewer*      specify user's preference for a viewer: >
```

```
                                        "kfmclient exec"
                                        "gnome-open"
<                                   If >
                                        "-"
<                                   is used, then netrwFileHandler() will look for
                                    a script/function to handle the given
                                    extension.  (see |netrw_filehandler|).

  *g:netrw_chgperm*                 Unix/Linux: "chmod PERM FILENAME"
                                    Windows:    "cacls FILENAME /e /p PERM"
                                    Used to change access permission for a file.

  *g:netrw_compress*                ="gzip"
                                        Will compress marked files with this
                                        command

  *g:Netrw_corehandler*             Allows one to specify something additional
                                    to do when handling <core> files via netrw's
                                    browser's "x" command (see |netrw-x|).  If
                                    present, g:Netrw_corehandler specifies
                                    either one or more function references
                                    (see |Funcref|).  (the capital g:Netrw...
                                    is required its holding a function reference)


  *g:netrw_ctags*                   ="ctags"
                                    The default external program used to create
                                    tags

  *g:netrw_cursor*                  = 2 (default)
                                    This option controls the use of the
                                    |'cursorline'| (cul) and |'cursorcolumn'|
                                    (cuc) settings by netrw:

                                    Value   Thin-Long-Tree      Wide
                                     =0      u-cul u-cuc      u-cul u-cuc
                                     =1      u-cul u-cuc        cul u-cuc
                                     =2        cul u-cuc        cul u-cuc
                                     =3        cul u-cuc        cul   cuc
                                     =4        cul   cuc        cul   cuc

                                    Where
                                      u-cul : user's |'cursorline'|   setting used
                                      u-cuc : user's |'cursorcolumn'| setting used
                                      cul   : |'cursorline'|  locally set
                                      cuc   : |'cursorcolumn'| locally set

  *g:netrw_decompress*              = { ".gz"  : "gunzip" ,
                                        ".bz2" : "bunzip2" ,
                                        ".zip" : "unzip" ,
                                        ".tar" : "tar -xf"}
                                      A dictionary mapping suffices to
                                      decompression programs.

  *g:netrw_dirhistmax*                =10: controls maximum quantity of past
                                         history.  May be zero to supppress
                                         history.
                                         (related: |netrw-qb| |netrw-u| |netrw-U|)

  *g:netrw_dynamic_maxfilenamelen* =32: enables dynamic determination of
                                     |g:netrw_maxfilenamelen|, which affects
                                     local file long listing.
```

```
*g:netrw_errorlvl*          =0: error levels greater than or equal to
                                this are permitted to be displayed
                                0: notes
                                1: warnings
                                2: errors

*g:netrw_fastbrowse*        =0: slow speed directory browsing;
                                never re-uses directory listings;
                                always obtains directory listings.
                            =1: medium speed directory browsing;
                                re-use directory listings only
                                when remote directory browsing.
                                (default value)
                            =2: fast directory browsing;
                                only obtains directory listings when the
                                directory hasn't been seen before
                                (or |netrw-ctrl-l| is used).

                            Fast browsing retains old directory listing
                            buffers so that they don't need to be
                            re-acquired.  This feature is especially
                            important for remote browsing.  However, if
                            a file is introduced or deleted into or from
                            such directories, the old directory buffer
                            becomes out-of-date.  One may always refresh
                            such a directory listing with |netrw-ctrl-l|.
                            This option gives the user the choice of
                            trading off accuracy (ie. up-to-date listing)
                            versus speed.

*g:netrw_ffkeep*            (default: doesn't exist)
                            If this variable exists and is zero, then
                            netrw will not do a save and restore for
                            |'fileformat'|.

*g:netrw_fname_escape*      =' ?&;%'
                            Used on filenames before remote reading/writing

*g:netrw_ftp_browse_reject* ftp can produce a number of errors and warnings
                            that can show up as "directories" and "files"
                            in the listing.  This pattern is used to
                            remove such embedded messages.  By default its
                            value is:
                             '^total\s\+\d\+$\|
                             ^Trying\s\+\d\+.*$\|
                             ^KERBEROS_V\d rejected\|
                             ^Security extensions not\|
                             No such file\|
                             : connect to address [0-9a-fA-F:]*
                             : No route to host$'

*g:netrw_ftp_list_cmd*      options for passing along to ftp for directory
                            listing.  Defaults:
                             unix or g:netrw_cygwin set: : "ls -lF"
                             otherwise                   "dir"


*g:netrw_ftp_sizelist_cmd*  options for passing along to ftp for directory
                            listing, sorted by size of file.
                            Defaults:
                             unix or g:netrw_cygwin set: : "ls -slF"
```

```
                                otherwise                    "dir"

  *g:netrw_ftp_timelist_cmd*    options for passing along to ftp for directory
                                listing, sorted by time of last modification.
                                Defaults:
                                 unix or g:netrw_cygwin set: : "ls -tlF"
                                 otherwise                    "dir"

  *g:netrw_glob_escape*         ='[]*?`{~$'  (unix)
                                ='[]*?`{$'  (windows
                                These characters in directory names are
                                escaped before applying glob()

  *g:netrw_gx*                  ="<cfile>"
                                This option controls how gx (|netrw-gx|) picks
                                up the text under the cursor.  See |expand()|
                                for possibilities.

  *g:netrw_hide*                Controlled by the "a" map (see |netrw-a|)
                                =0 : show all
                                =1 : show not-hidden files
                                =2 : show hidden files only
                                 default: =0

  *g:netrw_home*                The home directory for where bookmarks and
                                history are saved (as .netrwbook and
                                .netrwhist).
                                 default: the first directory on the
                                         |'runtimepath'|

  *g:netrw_keepdir*             =1 (default) keep current directory immune from
                                   the browsing directory.
                                =0 keep the current directory the same as the
                                   browsing directory.
                                The current browsing directory is contained in
                                b:netrw_curdir (also see |netrw-c|)

  *g:netrw_keepj*               ="keepj" (default) netrw attempts to keep the
                                         |:jumps| table unaffected.
                                =""      netrw will not use |:keepjumps| with
                                         exceptions only for the
                                         saving/restoration of position.

  *g:netrw_list_cmd*            command for listing remote directories
                                 default: (if ssh is executable)
                                          "ssh HOSTNAME ls -FLa"

*g:netrw_list_cmd_options*      If this variable exists, then its contents are
                                appended to the g:netrw_list_cmd.  For
                                example, use "2>/dev/null" to get rid of banner
                                messages on unix systems.


  *g:netrw_liststyle*           Set the default listing style:
                                = 0: thin listing (one file per line)
                                = 1: long listing (one file per line with time
                                     stamp information and file size)
                                = 2: wide listing (multiple files in columns)
                                = 3: tree style listing

  *g:netrw_list_hide*           comma separated pattern list for hiding files
                                Patterns are regular expressions (see |regexp|)
```

```
                                There's some special support for git-ignore
                                files: you may add the output from the helper
                                function 'netrw_gitignore#Hide() automatically
                                hiding all gitignored files.
                                For more details see |netrw-gitignore|.

                                Examples:
                                 let g:netrw_list_hide= '.*\.swp$'
                                 let g:netrw_list_hide=
netrw_gitignore#Hide().'.*\.swp$'
                                default: ""

  *g:netrw_localcopycmd*        ="cp" Linux/Unix/MacOS/Cygwin
                                ="copy" Windows
                                Copies marked files (|netrw-mf|) to target
                                directory (|netrw-mt|, |netrw-mc|)

 *g:netrw_localcopydircmd*      ="cp -R"        Linux/Unix/MacOS/Cygwin
                                ="xcopy /e /c /h/ /i /k"        Windows
                                Copies directories to target directory.
                                (|netrw-mc|, |netrw-mt|)

  *g:netrw_localmkdir*          command for making a local directory
                                 default: "mkdir"

  *g:netrw_localmovecmd*        ="mv" Linux/Unix/MacOS/Cygwin
                                ="move" Windows
                                Moves marked files (|netrw-mf|) to target
                                directory (|netrw-mt|, |netrw-mm|)

  *g:netrw_localrmdir*          remove directory command (rmdir)
                                 default: "rmdir"

  *g:netrw_maxfilenamelen*      =32 by default, selected so as to make long
                                    listings fit on 80 column displays.
                                If your screen is wider, and you have file
                                or directory names longer than 32 bytes,
                                you may set this option to keep listings
                                columnar.

  *g:netrw_mkdir_cmd*           command for making a remote directory
                                via ssh  (also see |g:netrw_remote_mkdir|)
                                 default: "ssh USEPORT HOSTNAME mkdir"

  *g:netrw_mousemaps*             =1 (default) enables mouse buttons while
                                   browsing to:
                                     leftmouse       : open file/directory
                                     shift-leftmouse : mark file
                                     middlemouse     : same as P
                                     rightmouse      : remove file/directory
                                =0: disables mouse maps

  *g:netrw_nobeval*             doesn't exist (default)
                                If this variable exists, then balloon
                                evaluation will be suppressed
                                (see |'ballooneval'|)

 *g:netrw_sizestyle*            not defined: actual bytes (default)
                                ="b" : actual bytes       (default)
                                ="h" : human-readable (ex. 5k, 4m, 3g)
                                       uses 1000 base
                                ="H" : human-readable (ex. 5K, 4M, 3G)
```

```
                                uses 1024 base
                              The long listing (|netrw-i|) and query-file
                              maps (|netrw-qf|) will display file size
                              using the specified style.

  *g:netrw_usetab*            if this variable exists and is non-zero, then
                              the <tab> map supporting shrinking/expanding a
                              Lexplore or netrw window will be enabled.
                              (see |netrw-c-tab|)

  *g:netrw_remote_mkdir*      command for making a remote directory
                              via ftp  (also see |g:netrw_mkdir_cmd|)
                               default: "mkdir"

  *g:netrw_retmap*            if it exists and is set to one, then:
                               * if in a netrw-selected file, AND
                               * no normal-mode <2-leftmouse> mapping exists,
                              then the <2-leftmouse> will be mapped for easy
                              return to the netrw browser window.
                               example: click once to select and open a file,
                                        double-click to return.

                              Note that one may instead choose to:
                               * let g:netrw_retmap= 1, AND
                               * nmap <silent> YourChoice <Plug>NetrwReturn
                              and have another mapping instead of
                              <2-leftmouse> to invoke the return.

                              You may also use the |:Rexplore| command to do
                              the same thing.

                                default: =0

  *g:netrw_rm_cmd*            command for removing remote files
                               default: "ssh USEPORT HOSTNAME rm"

  *g:netrw_rmdir_cmd*         command for removing remote directories
                               default: "ssh USEPORT HOSTNAME rmdir"

  *g:netrw_rmf_cmd*           command for removing remote softlinks
                               default: "ssh USEPORT HOSTNAME rm -f"

  *g:netrw_servername*        use this variable to provide a name for
                              |netrw-ctrl-r| to use for its server.
                               default: "NETRWSERVER"

  *g:netrw_sort_by*           sort by "name", "time", "size", or
                              "exten".
                               default: "name"

  *g:netrw_sort_direction*    sorting direction: "normal" or "reverse"
                               default: "normal"

  *g:netrw_sort_options*      sorting is done using |:sort|; this
                              variable's value is appended to the
                              sort command.  Thus one may ignore case,
                              for example, with the following in your
                              .vimrc: >
                                      let g:netrw_sort_options="i"
<                              default: ""

  *g:netrw_sort_sequence*     when sorting by name, first sort by the
```

```
                                comma-separated pattern sequence.  Note that
                                any filigree added to indicate filetypes
                                should be accounted for in your pattern.
                                 default: '[\/]$,*,\.bak$,\.o$,\.h$,
                                          \.info$,\.swp$,\.obj$'

  *g:netrw_special_syntax*      If true, then certain files will be shown
                                using special syntax in the browser:

                                        netrwBak     : *.bak
                                        netrwCompress: *.gz *.bz2 *.Z *.zip
                                        netrwData    : *.dat
                                        netrwHdr     : *.h
                                        netrwLib     : *.a *.so *.lib *.dll
                                        netrwMakefile: [mM]akefile *.mak
                                        netrwObj     : *.o *.obj
                                        netrwTags    : tags ANmenu ANtags
                                        netrwTilde   : *
                                        netrwTmp     : tmp* *tmp

                                These syntax highlighting groups are linked
                                to Folded or DiffChange by default
                                (see |hl-Folded| and |hl-DiffChange|), but
                                one may put lines like >
                                        hi link netrwCompress Visual
<                               into one's <.vimrc> to use one's own
                                preferences.  Alternatively, one may
                                put such specifications into
                                .vim/after/syntax/netrw.vim.

                                As an example, I myself use a dark-background
                                colorscheme with the following in
                                .vim/after/syntax/netrw.vim: >

 hi netrwCompress term=NONE cterm=NONE gui=NONE ctermfg=10 guifg=green  ctermbg=0
guibg=black
 hi netrwData     term=NONE cterm=NONE gui=NONE ctermfg=9 guifg=blue ctermbg=0
guibg=black
 hi netrwHdr      term=NONE cterm=NONE,italic gui=NONE guifg=SeaGreen1
 hi netrwLex      term=NONE cterm=NONE,italic gui=NONE guifg=SeaGreen1
 hi netrwYacc     term=NONE cterm=NONE,italic gui=NONE guifg=SeaGreen1
 hi netrwLib      term=NONE cterm=NONE gui=NONE ctermfg=14 guifg=yellow
 hi netrwObj      term=NONE cterm=NONE gui=NONE ctermfg=12 guifg=red
 hi netrwTilde    term=NONE cterm=NONE gui=NONE ctermfg=12 guifg=red
 hi netrwTmp      term=NONE cterm=NONE gui=NONE ctermfg=12 guifg=red
 hi netrwTags     term=NONE cterm=NONE gui=NONE ctermfg=12 guifg=red
 hi netrwDoc      term=NONE cterm=NONE gui=NONE ctermfg=220 ctermbg=27 guifg=yellow2
guibg=Blue3
 hi netrwSymLink  term=NONE cterm=NONE gui=NONE ctermfg=220 ctermbg=27 guifg=grey60
<
  *g:netrw_ssh_browse_reject*   ssh can sometimes produce unwanted lines,
                                messages, banners, and whatnot that one doesn't
                                want masquerading as "directories" and "files".
                                Use this pattern to remove such embedded
                                messages.  By default its value is:
                                        '^total\s\+\d\+$'

  *g:netrw_ssh_cmd*             One may specify an executable command
                                to use instead of ssh for remote actions
                                such as listing, file removal, etc.
                                 default: ssh
```

```
*g:netrw_suppress_gx_mesg*     =1 : browsers sometimes produce messages
                               which are normally unwanted intermixed
                               with the page.
                               However, when using links, for example,
                               those messages are what the browser produces.
                               By setting this option to 0, netrw will not
                               suppress browser messages.

 *g:netrw_tmpfile_escape*      =' &;'
                               escape() is applied to all temporary files
                               to escape these characters.

 *g:netrw_timefmt*             specify format string to vim's strftime().
                               The default, "%c", is "the preferred date
                               and time representation for the current
                               locale" according to my manpage entry for
                               strftime(); however, not all are satisfied
                               with it.  Some alternatives:
                                "%a %d %b %Y %T",
                                " %a %Y-%m-%d  %I-%M-%S %p"
                                default: "%c"

 *g:netrw_use_noswf*           netrw normally avoids writing swapfiles
                               for browser buffers.  However, under some
                               systems this apparently is causing nasty
                               ml_get errors to appear; if you're getting
                               ml_get errors, try putting
                                 let g:netrw_use_noswf= 0
                               in your .vimrc.
                                 default: 1

 *g:netrw_winsize*             specify initial size of new windows made with
                               "o" (see |netrw-o|), "v" (see |netrw-v|),
                               |:Hexplore| or |:Vexplore|.  The g:netrw_winsize
                               is an integer describing the percentage of the
                               current netrw buffer's window to be used for
                               the new window.
                                If g:netrw_winsize is less than zero, then
                               the absolute value of g:netrw_winsize lines
                               or columns will be used for the new window.
                                If g:netrw_winsize is zero, then a normal
                               split will be made (ie. |'equalalways'| will
                               take effect, for example).
                                default: 50  (for 50%)

 *g:netrw_wiw*                 =1 specifies the minimum window width to use
                               when shrinking a netrw/Lexplore window
                               (see |netrw-c-tab|).

 *g:netrw_xstrlen*             Controls how netrw computes string lengths,
                               including multi-byte characters' string
                               length. (thanks to N Weibull, T Mechelynck)
                               =0: uses Vim's built-in strlen()
                               =1: number of codepoints (Latin a + combining
                                   circumflex is two codepoints)  (DEFAULT)
                               =2: number of spacing codepoints (Latin a +
                                   combining circumflex is one spacing
                                   codepoint; a hard tab is one; wide and
                                   narrow CJK are one each; etc.)
                               =3: virtual length (counting tabs as anything
                                   between 1 and |'tabstop'|, wide CJK as 2
                                   rather than 1, Arabic alif as zero when
```

```
                                    immediately preceded by lam, one
                                    otherwise, etc)

  *g:NetrwTopLvlMenu*           This variable specifies the top level
                                menu name; by default, it's "Netrw.".  If
                                you wish to change this, do so in your
                                .vimrc.

NETRW BROWSING AND OPTION INCOMPATIBILITIES     *netrw-incompatible* {{{2

Netrw has been designed to handle user options by saving them, setting the
options to something that's compatible with netrw's needs, and then restoring
them.  However, the autochdir option: >
        :set acd
is problematic.  Autochdir sets the current directory to that containing the
file you edit; this apparently also applies to directories.  In other words,
autochdir sets the current directory to that containing the "file" (even if
that "file" is itself a directory).

NETRW SETTINGS WINDOW                           *netrw-settings-window* {{{2

With the NetrwSettings.vim plugin, >
        :NetrwSettings
will bring up a window with the many variables that netrw uses for its
settings.  You may change any of their values; when you save the file, the
settings therein will be used.  One may also press "?" on any of the lines for
help on what each of the variables do.

(also see: |netrw-browser-var| |netrw-protocol| |netrw-variables|)


==============================================================================
OBTAINING A FILE                                        *netrw-obtain* *netrw-O* {{{2

If there are no marked files:

    When browsing a remote directory, one may obtain a file under the cursor
    (ie.  get a copy on your local machine, but not edit it) by pressing the O
    key.

If there are marked files:

    The marked files will be obtained (ie. a copy will be transferred to your
    local machine, but not set up for editing).

Only ftp and scp are supported for this operation (but since these two are
available for browsing, that shouldn't be a problem).  The status bar will
then show, on its right hand side, a message like "Obtaining filename".  The
statusline will be restored after the transfer is complete.

Netrw can also "obtain" a file using the local browser.  Netrw's display
of a directory is not necessarily the same as Vim's "current directory",
unless |g:netrw_keepdir| is set to 0 in the user's <.vimrc>.  One may select
a file using the local browser (by putting the cursor on it) and pressing
"O" will then "obtain" the file; ie. copy it to Vim's current directory.

Related topics:
 * To see what the current directory is, use |:pwd|
 * To make the currently browsed directory the current directory, see |netrw-c|
 * To automatically make the currently browsed directory the current
   directory, see |g:netrw_keepdir|.
```

OPEN A NEW FILE IN NETRW'S CURRENT DIRECTORY *netrw-newfile* *netrw-createfile* *netrw-%* {{{2

To open a new file in netrw's current directory, press "%". This map will query the user for a new filename; an empty file by that name will be placed in the netrw's current directory (ie. b:netrw_curdir).

Related topics: |netrw-d|

PREVIEW WINDOW *netrw-p* *netrw-preview* {{{2

One may use a preview window by using the "p" key when the cursor is atop the desired filename to be previewed. The display will then split to show both the browser (where the cursor will remain) and the file (see |:pedit|). By default, the split will be taken horizontally; one may use vertical splitting if one has set |g:netrw_preview| first.

An interesting set of netrw settings is: >

```
let g:netrw_preview   = 1
let g:netrw_liststyle = 3
let g:netrw_winsize   = 30
```

These will:

1. Make vertical splitting the default for previewing files
2. Make the default listing style "tree"
3. When a vertical preview window is opened, the directory listing will use only 30% of the columns available; the rest of the window is used for the preview window.

Related: if you like this idea, you may also find :Lexplore (|netrw-:Lexplore|) or |g:netrw_chgwin| of interest

Also see: |g:netrw_chgwin| |netrw-P| |'previewwindow'| |CTRL-W_z| |:pclose|

PREVIOUS WINDOW *netrw-P* *netrw-prvwin* {{{2

To edit a file or directory under the cursor in the previously used (last accessed) window (see :he |CTRL-W_p|), press a "P". If there's only one window, then the one window will be horizontally split (by default).

If there's more than one window, the previous window will be re-used on the selected file/directory. If the previous window's associated buffer has been modified, and there's only one window with that buffer, then the user will be asked if s/he wishes to save the buffer first (yes, no, or cancel).

Related Actions |netrw-cr| |netrw-o| |netrw-t| |netrw-v|
Associated setting variables:

|g:netrw_alto|	control above/below splitting
|g:netrw_altv|	control right/left splitting
|g:netrw_preview|	control horizontal vs vertical splitting
|g:netrw_winsize|	control initial sizing

Also see: |g:netrw_chgwin| |netrw-p|

REFRESHING THE LISTING *netrw-refresh* *netrw-ctrl-l* *netrw-ctrl_l* {{{2

```
To refresh either a local or remote directory listing, press ctrl-l (<c-l>) or
hit the <cr> when atop the ./ directory entry in the listing.  One may also
refresh a local directory by using ":e .".


REVERSING SORTING ORDER         *netrw-r* *netrw-reverse* {{{2

One may toggle between normal and reverse sorting order by pressing the
"r" key.

Related topics:              |netrw-s|
Associated setting variable: |g:netrw_sort_direction|


RENAMING FILES OR DIRECTORIES   *netrw-move* *netrw-rename* *netrw-R* {{{2

If there are no marked files: (see |netrw-mf|)

    Renaming files and directories involves moving the cursor to the
    file/directory to be moved (renamed) and pressing "R".  You will then be
    queried for what you want the file/directory to be renamed to  You may select
    a range of lines with the "V" command (visual selection), and then
    press "R"; you will be queried for each file as to what you want it
    renamed to.

If there are marked files:  (see |netrw-mf|)

    Marked files will be renamed (moved).  You will be queried as above in
    order to specify where you want the file/directory to be moved.

    If you answer a renaming query with a "s/frompattern/topattern/", then
    subsequent files on the marked file list will be renamed by taking each
    name, applying that substitute, and renaming each file to the result.
    As an example : >

        mr  [query: reply with *.c]
        R   [query: reply with s/^\(.*\)\.c$/\1.cpp/]
<
    This example will mark all *.c files and then rename them to *.cpp
    files.

    The ctrl-X character has special meaning for renaming files: >

        <c-x>      : a single ctrl-x tells netrw to ignore the portion of the response
                     lying between the last '/' and the ctrl-x.

        <c-x><c-x> : a pair of contiguous ctrl-x's tells netrw to ignore any
                     portion of the string preceding the double ctrl-x's.
<
    WARNING:~

    Note that moving files is a dangerous operation; copies are safer.  That's
    because a "move" for remote files is actually a copy + delete -- and if
    the copy fails and the delete does not, you may lose the file.
    Use at your own risk.

The g:netrw_rename_cmd variable is used to implement remote renaming.  By
default its value is:

        ssh HOSTNAME mv

One may rename a block of files and directories by selecting them with
```

```
V (|linewise-visual|) when using thin style


SELECTING SORTING STYLE                *netrw-s* *netrw-sort* {{{2

One may select the sorting style by name, time, or (file) size.  The "s" map
allows one to circulate amongst the three choices; the directory listing will
automatically be refreshed to reflect the selected style.

Related topics:               |netrw-r| |netrw-S|
Associated setting variables: |g:netrw_sort_by| |g:netrw_sort_sequence|


SETTING EDITING WINDOW         *netrw-editwindow* *netrw-C* *netrw-:NetrwC* {{{2

One may select a netrw window for editing with the "C" mapping, using the
:NetrwC [win#] command, or by setting |g:netrw_chgwin| to the selected window
number.  Subsequent selection of a file to edit (|netrw-cr|) will use that
window.

        * C : by itself, will select the current window holding a netrw buffer
          for editing via |netrw-cr|.  The C mapping is only available while in
          netrw buffers.

        * [count]C : the count will be used as the window number to be used
          for subsequent editing via |netrw-cr|.

        * :NetrwC will set |g:netrw_chgwin| to the current window

        * :NetrwC win#  will set |g:netrw_chgwin| to the specified window
          number

Using >
        let g:netrw_chgwin= -1
will restore the default editing behavior
(ie. editing will use the current window).

Related topics:                 |netrw-cr| |g:netrw_browse_split|
Associated setting variables:   |g:netrw_chgwin|


SHRINKING OR EXPANDING A NETRW OR LEXPLORE WINDOW       *netrw-c-tab* {{{2

The <c-tab> key will toggle a netrw or |:Lexplore| window's width,
but only if |g:netrw_usetab| exists and is non-zero (and, of course,
only if your terminal supports differentiating <c-tab> from a plain
<tab>).

  * If the current window is a netrw window, toggle its width
    (between |g:netrw_wiw| and its original width)

  * Else if there is a |:Lexplore| window in the current tab, toggle
    its width

  * Else bring up a |:Lexplore| window

If |g:netrw_usetab| exists or is zero, or if there is a pre-existing mapping
for <c-tab>, then the <c-tab> will not be mapped.  One may map something other
than a <c-tab>, too: (but you'll still need to have had g:netrw_usetab set) >

        nmap <unique> (whatever)        <Plug>NetrwShrink
<
```

```
Related topics:                 |:Lexplore|
Associated setting variable:    |g:netrw_usetab|


USER SPECIFIED MAPS                                     *netrw-usermaps* {{{1

One may make customized user maps.  Specify a variable, |g:Netrw_UserMaps|,
to hold a |List| of lists of keymap strings and function names: >

        [["keymap-sequence","ExampleUserMapFunc"],...]
<
When netrw is setting up maps for a netrw buffer, if |g:Netrw_UserMaps|
exists, then the internal function netrw#UserMaps(islocal) is called.
This function goes through all the entries in the |g:Netrw_UserMaps| list:

        * sets up maps: >
                nno <buffer> <silent> KEYMAP-SEQUENCE
                :call s:UserMaps(islocal,"ExampleUserMapFunc")
<       * refreshes if result from that function call is the string
          "refresh"
        * if the result string is not "", then that string will be
          executed (:exe result)
        * if the result is a List, then the above two actions on results
          will be taken for every string in the result List

The user function is passed one argument; it resembles >

        fun! ExampleUserMapFunc(islocal)
<
where a:islocal is 1 if it's a local-directory system call or 0 when
remote-directory system call.

Use netrw#Expose("varname")          to access netrw-internal (script-local)
                                     variables.
Use netrw#Modify("varname",newvalue) to change netrw-internal variables.
Use netrw#Call("funcname"[,args])    to call a netrw-internal function with
                                     specified arguments.

Example: Get a copy of netrw's marked file list: >

        let netrwmarkfilelist= netrw#Expose("netrwmarkfilelist")
<
Example: Modify the value of netrw's marked file list: >

        call netrw#Modify("netrwmarkfilelist",[])
<
Example: Clear netrw's marked file list via a mapping on gu >
    " ExampleUserMap: {{{2
    fun! ExampleUserMap(islocal)
      call netrw#Modify("netrwmarkfilelist",[])
      call netrw#Modify('netrwmarkfilemtch_{bufnr("%")}',"")
      let retval= ["refresh"]
      return retval
    endfun
    let g:Netrw_UserMaps= [["gu","ExampleUserMap"]]
<

10. Problems and Fixes                                  *netrw-problems* {{{1

        (This section is likely to grow as I get feedback)
        (also see |netrw-debug|)
                                                                *netrw-p1*
```

P1. I use windows 95, and my ftp dumps four blank lines at the
end of every read.

See |netrw-fixup|, and put the following into your
<.vimrc> file:

```
let g:netrw_win95ftp= 1
```

netrw-p2

P2. I use Windows, and my network browsing with ftp doesn't sort by
time or size! -or- The remote system is a Windows server; why
don't I get sorts by time or size?

Windows' ftp has a minimal support for ls (ie. it doesn't
accept sorting options). It doesn't support the -F which
gives an explanatory character (ABC/ for "ABC is a directory").
Netrw then uses "dir" to get both its thin and long listings.
If you think your ftp does support a full-up ls, put the
following into your <.vimrc>: >

```
let g:netrw_ftp_list_cmd    = "ls -lF"
let g:netrw_ftp_timelist_cmd= "ls -tlF"
let g:netrw_ftp_sizelist_cmd= "ls -slF"
```

<

Alternatively, if you have cygwin on your Windows box, put
into your <.vimrc>: >

```
let g:netrw_cygwin= 1
```

<

This problem also occurs when the remote system is Windows.
In this situation, the various g:netrw_ftp_[time|size]list_cmds
are as shown above, but the remote system will not correctly
modify its listing behavior.

netrw-p3

P3. I tried rcp://user@host/ (or protocol other than ftp) and netrw
used ssh! That wasn't what I asked for...

Netrw has two methods for browsing remote directories: ssh
and ftp. Unless you specify ftp specifically, ssh is used.
When it comes time to do download a file (not just a directory
listing), netrw will use the given protocol to do so.

netrw-p4

P4. I would like long listings to be the default.

Put the following statement into your |.vimrc|: >

```
let g:netrw_liststyle= 1
```

<

Check out |netrw-browser-var| for more customizations that
you can set.

netrw-p5

P5. My times come up oddly in local browsing

Does your system's strftime() accept the "%c" to yield dates
such as "Sun Apr 27 11:49:23 1997"? If not, do a
"man strftime" and find out what option should be used. Then
put it into your |.vimrc|: >

```
                        let g:netrw_timefmt= "%X"  (where X is the option)
<
                                                                *netrw-p6*
        P6. I want my current directory to track my browsing.
            How do I do that?

            Put the following line in your |.vimrc|:
>
                let g:netrw_keepdir= 0
<
                                                                *netrw-p7*
        P7. I use Chinese (or other non-ascii) characters in my filenames, and
            netrw (Explore, Sexplore, Hexplore, etc) doesn't display them!

                (taken from an answer provided by Wu Yongwei on the vim
                mailing list)
                I now see the problem. Your code page is not 936, right? Vim
                seems only able to open files with names that are valid in the
                current code page, as are many other applications that do not
                use the Unicode version of Windows APIs. This is an OS-related
                issue. You should not have such problems when the system
                locale uses UTF-8, such as modern Linux distros.

                (...it is one more reason to recommend that people use utf-8!)

                                                                *netrw-p8*
        P8. I'm getting "ssh is not executable on your system" -- what do I
            do?

                (Dudley Fox) Most people I know use putty for windows ssh.  It
                is a free ssh/telnet application. You can read more about it
                here:

                http://www.chiark.greenend.org.uk/~sgtatham/putty/ Also:

                (Marlin Unruh) This program also works for me. It's a single
                executable, so he/she can copy it into the Windows\System32
                folder and create a shortcut to it.

                (Dudley Fox) You might also wish to consider plink, as it
                sounds most similar to what you are looking for. plink is an
                application in the putty suite.

           http://the.earth.li/~sgtatham/putty/0.58/htmldoc/Chapter7.html#plink

                (Vissale Neang) Maybe you can try OpenSSH for windows, which
                can be obtained from:

                http://sshwindows.sourceforge.net/

                It doesn't need the full Cygwin package.

                (Antoine Mechelynck) For individual Unix-like programs needed
                for work in a native-Windows environment, I recommend getting
                them from the GnuWin32 project on sourceforge if it has them:

                    http://gnuwin32.sourceforge.net/

                Unlike Cygwin, which sets up a Unix-like virtual machine on
                top of Windows, GnuWin32 is a rewrite of Unix utilities with
                Windows system calls, and its programs works quite well in the
                cmd.exe "Dos box".
```

```
                (dave) Download WinSCP and use that to connect to the server.
                In Preferences > Editors, set gvim as your editor:

                        - Click "Add..."
                        - Set External Editor (adjust path as needed, include
                          the quotes and !.! at the end):
                            "c:\Program Files\Vim\vim70\gvim.exe" !.!
                        - Check that the filetype in the box below is
                          {asterisk}.{asterisk} (all files), or whatever types
                          you want (cec: change {asterisk} to * ; I had to
                          write it that way because otherwise the helptags
                          system thinks it's a tag)
                        - Make sure it's at the top of the listbox (click it,
                          then click "Up" if it's not)
                If using the Norton Commander style, you just have to hit <F4>
                to edit a file in a local copy of gvim.

                (Vit Gottwald) How to generate public/private key and save
                public key it on server: >
  http://www.chiark.greenend.org.uk/~sgtatham/putty/0.60/htmldoc/Chapter8.html#pubkey-
gettingready
                        (8.3 Getting ready for public key authentication)
<
                How to use a private key with 'pscp': >

  http://www.chiark.greenend.org.uk/~sgtatham/putty/0.60/htmldoc/Chapter5.html
                        (5.2.4 Using public key authentication with PSCP)
<
                (Ben Schmidt) I find the ssh included with cwRsync is
                brilliant, and install cwRsync or cwRsyncServer on most
                Windows systems I come across these days. I guess COPSSH,
                packed by the same person, is probably even better for use as
                just ssh on Windows, and probably includes sftp, etc. which I
                suspect the cwRsync doesn't, though it might

                (cec) To make proper use of these suggestions above, you will
                need to modify the following user-settable variables in your
                .vimrc:

                |g:netrw_ssh_cmd| |g:netrw_list_cmd|  |g:netrw_mkdir_cmd|
                |g:netrw_rm_cmd|  |g:netrw_rmdir_cmd| |g:netrw_rmf_cmd|

                The first one (|g:netrw_ssh_cmd|) is the most important; most
                of the others will use the string in g:netrw_ssh_cmd by
                default.

                                                    *netrw-p9* *netrw-ml_get*
        P9. I'm browsing, changing directory, and bang!  ml_get errors
            appear and I have to kill vim.  Any way around this?

                Normally netrw attempts to avoid writing swapfiles for
                its temporary directory buffers.  However, on some systems
                this attempt appears to be causing ml_get errors to
                appear.  Please try setting |g:netrw_use_noswf| to 0
                in your <.vimrc>: >
                        let g:netrw_use_noswf= 0
<
                                                                *netrw-p10*
        P10. I'm being pestered with "[something] is a directory" and
             "Press ENTER or type command to continue" prompts...
```

```
            The "[something] is a directory" prompt is issued by Vim,
            not by netrw, and there appears to be no way to work around
            it.  Coupled with the default cmdheight of 1, this message
            causes the "Press ENTER..." prompt.  So:  read |hit-enter|;
            I also suggest that you set your |'cmdheight'| to 2 (or more) in
            your <.vimrc> file.

                                                              *netrw-p11*
P11. I want to have two windows; a thin one on the left and my editing
     window on the right.  How may I accomplish this?

     You probably want netrw running as in a side window.  If so, you
     will likely find that ":[N]Lexplore" does what you want.  The
     optional "[N]" allows you to select the quantity of columns you
     wish the |:Lexplore|r window to start with (see |g:netrw_winsize|
     for how this parameter works).

     Previous solution:

        * Put the following line in your <.vimrc>:
              let g:netrw_altv = 1
        * Edit the current directory:  :e .
        * Select some file, press v
        * Resize the windows as you wish (see |CTRL-W_<| and
          |CTRL-W_>|).  If you're using gvim, you can drag
          the separating bar with your mouse.
        * When you want a new file, use  ctrl-w h  to go back to the
          netrw browser, select a file, then press P  (see |CTRL-W_h|
          and |netrw-P|).  If you're using gvim, you can press
          <leftmouse> in the browser window and then press the
          <middlemouse> to select the file.


                                                              *netrw-p12*
P12. My directory isn't sorting correctly, or unwanted letters are
     appearing in the listed filenames, or things aren't lining
     up properly in the wide listing, ...

     This may be due to an encoding problem.  I myself usually use
     utf-8, but really only use ascii (ie. bytes from 32-126).
     Multibyte encodings use two (or more) bytes per character.
     You may need to change |g:netrw_sepchr| and/or |g:netrw_xstrlen|.

                                                              *netrw-p13*
P13. I'm a Windows + putty + ssh user, and when I attempt to browse,
     the directories are missing trailing "/"s so netrw treats them
     as file transfers instead of as attempts to browse
     subdirectories.  How may I fix this?

     (mikeyao) If you want to use vim via ssh and putty under Windows,
     try combining the use of pscp/psftp with plink.  pscp/psftp will
     be used to connect and plink will be used to execute commands on
     the server, for example: list files and directory using 'ls'.

     These are the settings I use to do this:
>
    " list files, it's the key setting, if you haven't set,
    " you will get a blank buffer
    let g:netrw_list_cmd = "plink HOSTNAME ls -Fa"
    " if you haven't add putty directory in system path, you should
    " specify scp/sftp command.  For examples:
    "let g:netrw_sftp_cmd = "d:\\dev\\putty\\PSFTP.exe"
```

```
            "let g:netrw_scp_cmd = "d:\\dev\\putty\\PSCP.exe"
<
                                                                *netrw-p14*
        P14. I would like to speed up writes using Nwrite and scp/ssh
             style connections.  How?  (Thomer M. Gil)

             Try using ssh's ControlMaster and ControlPath (see the ssh_config
             man page) to share multiple ssh connections over a single network
             connection. That cuts out the cryptographic handshake on each
             file write, sometimes speeding it up by an order of magnitude.
             (see  http://thomer.com/howtos/netrw_ssh.html)
             (included by permission)

             Add the following to your ~/.ssh/config: >

                 # you change "*" to the hostname you care about
                 Host *
                   ControlMaster auto
                   ControlPath /tmp/%r@%h:%p

<            Then create an ssh connection to the host and leave it running: >

                 ssh -N host.domain.com

<            Now remotely open a file with Vim's Netrw and enjoy the
             zippiness: >

                vim scp://host.domain.com//home/user/.bashrc
<
                                                                *netrw-p15*
        P15. How may I use a double-click instead of netrw's usual single click
             to open a file or directory?  (Ben Fritz)

             First, disable netrw's mapping with >
                    let g:netrw_mousemaps= 0
<            and then create a netrw buffer only mapping in
             $HOME/.vim/after/ftplugin/netrw.vim: >
                    nmap <buffer> <2-leftmouse> <CR>
<            Note that setting g:netrw_mousemaps to zero will turn off
             all netrw's mouse mappings, not just the <leftmouse> one.
             (see |g:netrw_mousemaps|)

                                                                *netrw-p16*
        P16. When editing remote files (ex. :e ftp://hostname/path/file),
             under Windows I get an |E303| message complaining that it's unable
             to open a swap file.

             (romainl) It looks like you are starting Vim from a protected
             directory.  Start netrw from your $HOME or other writable
             directory.

                                                                *netrw-p17*
        P17. Netrw is closing buffers on its own.
             What steps will reproduce the problem?
                1. :Explore, navigate directories, open a file
                2. :Explore, open another file
                3. Buffer opened in step 1 will be closed. o
            What is the expected output? What do you see instead?
                I expect both buffers to exist, but only the last one does.

           (Lance) Problem is caused by "set autochdir" in .vimrc.
           (drchip) I am able to duplicate this problem with |'acd'| set.
```

```
                    It appears that the buffers are not exactly closed;
                    a ":ls!" will show them (although ":ls" does not).

                                                              *netrw-P18*
        P18. How to locally edit a file that's only available via
             another server accessible via ssh?
             See http://stackoverflow.com/questions/12469645/
             "Using Vim to Remotely Edit A File on ServerB Only
              Accessible From ServerA"

                                                              *netrw-P19*
        P19. How do I get numbering on in directory listings?
                With |g:netrw_bufsettings|, you can control netrw's buffer
                settings; try putting >
                  let g:netrw_bufsettings="noma nomod nu nobl nowrap ro nornu"
<               in your .vimrc.  If you'd like to have relative numbering
                instead, try >
                  let g:netrw_bufsettings="noma nomod nonu nobl nowrap ro rnu"
<
                                                              *netrw-P20*
        P20. How may I have gvim start up showing a directory listing?
                Try putting the following code snippet into your .vimrc: >
                    augroup VimStartup
                      au!
                      au VimEnter * if expand("%") == "" && argc() == 0 &&
                      \ (v:servername =~ 'GVIM\d*' || v:servername == "")
                      \ | e . | endif
                    augroup END
<               You may use Lexplore instead of "e" if you're so inclined.
                This snippet assumes that you have client-server enabled
                (ie. a "huge" vim version).

                                                              *netrw-P21*
        P21. I've made a directory (or file) with an accented character, but
                netrw isn't letting me enter that directory/read that file:

                It's likely that the shell or o/s is using a different encoding
                than you have vim (netrw) using.  A patch to vim supporting
                "systemencoding" may address this issue in the future; for
                now, just have netrw use the proper encoding.  For example: >

                        au FileType netrw set enc=latin1
<
                                                              *netrw-P22*
        P22. I get an error message when I try to copy or move a file:

                **error** (netrw) tried using g:netrw_localcopycmd<cp>; it doesn't
work!

             What's wrong?

             Netrw uses several system level commands to do things (see

                 |g:netrw_localcopycmd|, |g:netrw_localmovecmd|,
                 |g:netrw_localrmdir|, |g:netrw_mkdir_cmd|).

            You may need to adjust the default commands for one or more of
            these commands by setting them properly in your .vimrc.  Another
            source of difficulty is that these commands use vim's local
            directory, which may not be the same as the browsing directory
            shown by netrw (see |g:netrw_keepdir|).
```

```
==============================================================================
11. Debugging Netrw Itself                                *netrw-debug* {{{1

Step 1: check that the problem you've encountered hasn't already been resolved
by obtaining a copy of the latest (often developmental) netrw at:

        http://www.drchip.org/astronaut/vim/index.html#NETRW

The <netrw.vim> script is typically installed on systems as something like:
>
        /usr/local/share/vim/vim7x/plugin/netrwPlugin.vim
        /usr/local/share/vim/vim7x/autoload/netrw.vim
                (see output of :echo &rtp)
<
which is loaded automatically at startup (assuming :set nocp).  If you
installed a new netrw, then it will be located at >

        $HOME/.vim/plugin/netrwPlugin.vim
        $HOME/.vim/autoload/netrw.vim
<
Step 2: assuming that you've installed the latest version of netrw,
check that your problem is really due to netrw.  Create a file
called netrw.vimrc with the following contents: >

        set nocp
        so $HOME/.vim/plugin/netrwPlugin.vim
<
Then run netrw as follows: >

        vim -u netrw.vimrc --noplugins -i NONE [some path here]
<
Perform whatever netrw commands you need to, and check that the problem is
still present.  This procedure sidesteps any issues due to personal .vimrc
settings, .viminfo file, and other plugins.  If the problem does not appear,
then you need to determine which setting in your .vimrc is causing the
conflict with netrw or which plugin(s) is/are involved.

Step 3: If the problem still is present, then get a debugging trace from
netrw:

        1. Get the <Decho.vim> script, available as:

             http://www.drchip.org/astronaut/vim/index.html#DECHO
           or
             http://vim.sourceforge.net/scripts/script.php?script_id=120

          Decho.vim is provided as a "vimball"; see |vimball-intro|.

        2. Edit the <netrw.vim> file by typing: >

                vim netrw.vim
                :DechoOn
                :wq
<
           To restore to normal non-debugging behavior, re-edit <netrw.vim>
           and type >

                vim netrw.vim
                :DechoOff
                :wq
<
```

```
           This command, provided by <Decho.vim>, will comment out all
           Decho-debugging statements (Dfunc(), Dret(), Decho(), Dredir()).

        3. Then bring up vim and attempt to evoke the problem by doing a
           transfer or doing some browsing.  A set of messages should appear
           concerning the steps that <netrw.vim> took in attempting to
           read/write your file over the network in a separate tab or
           server vim window.

           To save the file, use >

                :tabnext
                :set bt=
                :w! DBG

<          Furthermore, it'd be helpful if you would type >
                :Dsep <command>
<          where <command> is the command you're about to type next,
           thereby making it easier to associate which part of the
           debugging trace is due to which command.

           Please send that information to <netrw.vim>'s maintainer along
           with the o/s you're using and the vim version that you're using
           (see |:version|) >
                NdrOchip at ScampbellPfamily.AbizM - NOSPAM
<
==============================================================================
12. History                                              *netrw-history* {{{1

        v156:   Feb 18, 2016    * Changed =~ to =~# where appropriate
                Feb 23, 2016    * s:ComposePath(base,subdir) now uses
                                  fnameescape() on the base portion
                Mar 01, 2016    * (gt_macki) reported where :Explore would
                                  make file unlisted. Fixed (tst943)
                Apr 04, 2016    * (reported by John Little) netrw normally
                                  suppresses browser messages, but sometimes
                                  those "messages" are what is wanted.
                                  See |g:netrw_suppress_gx_mesg|
                Apr 06, 2016    * (reported by Carlos Pita) deleting a remote
                                  file was giving an error message.  Fixed.
                Apr 08, 2016    * (Charles Cooper) had a problem with an
                                  undefined b:netrw_curdir.  He also provided
                                  a fix.
                Apr 20, 2016    * Changed s:NetrwGetBuffer(); now uses
                                  dictionaries.  Also fixed the "No Name"
                                  buffer problem.
        v155:   Oct 29, 2015    * (Timur Fayzrakhmanov) reported that netrw's
                                  mapping of ctrl-l was not allowing refresh of
                                  other windows when it was done in a netrw
                                  window.
                Nov 05, 2015    * Improved s:TreeSqueezeDir() to use search()
                                  instead of a loop
                                * NetrwBrowse() will return line to
                                  w:netrw_bannercnt if cursor ended up in
                                  banner
                Nov 16, 2015    * Added a <Plug>NetrwTreeSqueeze (|netrw-s-cr|)
                Nov 17, 2015    * Commented out imaps -- perhaps someone can
                                  tell me how they're useful and should be
                                  retained?
                Nov 20, 2015    * Added |netrw-ma| and |netrw-mA| support
                Nov 20, 2015    * gx (|netrw-gx|) on an url downloaded the
                                  file in addition to simply bringing up the
```

```
                      url in a browser.  Fixed.
            Nov 23, 2015    * Added |g:netrw_sizestyle| support
            Nov 27, 2015    * Inserted a lot of <c-u>s into various netrw
                      maps.
            Jan 05, 2016    * |netrw-qL| implemented to mark files based
                      upon |location-list|s; similar to |netrw-qF|.
            Jan 19, 2016    * using - call delete(directoryname,"d") -
                      instead of using g:netrw_localrmdir if
                      v7.4 + patch#1107 is available
            Jan 28, 2016    * changed to using |winsaveview()| and
                      |winrestview()|
            Jan 28, 2016    * s:NetrwTreePath() now does a save and
                      restore of view
            Feb 08, 2016    * Fixed a tree-listing problem with remote
                      directories
    v154:   Feb 26, 2015    * (Yuri Kanivetsky) reported a situation where
                      a file was not treated properly as a file
                      due to g:netrw_keepdir == 1
            Mar 25, 2015    * (requested by Ben Friz) one may now sort by
                      extension
            Mar 28, 2015    * (requested by Matt Brooks) netrw has a lot
                      of buffer-local mappings; however, some
                      plugins (such as vim-surround) set up
                      conflicting mappings that cause vim to wait.
                      The "<nowait>" modifier has been included
                      with most of netrw's mappings to avoid that
                      delay.
            Jun 26, 2015    * |netrw-gn| mapping implemted
                            * :Ntree NotADir resulted in having
                      the tree listing expand in the error messages
                      window.  Fixed.
            Jun 29, 2015    * Attempting to delete a file remotely caused
                      an error with "keepsol" mentioned; fixed.
            Jul 08, 2015    * Several changes to keep the |:jumps| table
                      correct when working with
                      |g:netrw_fastbrowse| set to 2
                            * wide listing with accented characters fixed
                      (using %-S instead of %-s with a |printf()|
            Jul 13, 2015    * (Daniel Hahler) CheckIfKde() could be true
                      but kfmclient not installed.  Changed order
                      in netrw#BrowseX(): checks if kde and
                      kfmclient, then will use xdg-open on a unix
                      system (if xdg-open is executable)
            Aug 11, 2015    * (McDonnell) tree listing mode wouldn't
                      select a file in a open subdirectory.
                            * (McDonnell) when multiple subdirectories
                      were concurrently open in tree listing
                      mode, a ctrl-L wouldn't refresh properly.
                            * The netrw:target menu showed duplicate
                      entries
            Oct 13, 2015    * (mattn) provided an exception to handle
                      windows with shellslash set but no shell
            Oct 23, 2015    * if g:netrw_usetab and <c-tab> now used
                      to control whether NetrwShrink is used
                      (see |netrw-c-tab|)
    v153:   May 13, 2014    * added another |g:netrw_ffkeep| usage {{{2
            May 14, 2014    * changed s:PerformListing() so that it
                      always sets ft=netrw for netrw buffers
                      (ie. even when syntax highlighting is
                      off, not available, etc)
            May 16, 2014    * introduced the |netrw-ctrl-r| functionality
            May 17, 2014    * introduced the |netrw-:NetrwMB| functionality
```

```
                    * mb and mB (|netrw-mb|, |netrw-mB|) will
                      add/remove marked files from bookmark list
                May 20, 2014    * (Enno Nagel) reported that :Lex <dirname>
                      wasn't working.  Fixed.
                May 26, 2014    * restored test to prevent leftmouse window
                      resizing from causing refresh.
                      (see s:NetrwLeftmouse())
                    * fixed problem where a refresh caused cursor
                      to go just under the banner instead of
                      staying put
                May 28, 2014    * (László Bimba) provided a patch for opening
                      the |:Lexplore| window 100% high, optionally
                      on the right, and will work with remote
                      files.
                May 29, 2014    * implemented :NetrwC  (see |netrw-:NetrwC|)
                Jun 01, 2014    * Removed some "silent"s from commands used
                      to implemented scp://... and pscp://...
                      directory listing.  Permits request for
                      password to appear.
                Jun 05, 2014    * (Enno Nagel) reported that user maps "/"
                      caused problems with "b" and "w", which
                      are mapped (for wide listings only) to
                      skip over files rather than just words.
                Jun 10, 2014    * |g:netrw_gx| introduced to allow users to
                      override default "<cfile>" with the gx
                      (|netrw-gx|) map
                Jun 11, 2014    * gx (|netrw-gx|), with |'autowrite'| set,
                      will write modified files.  s:NetrwBrowseX()
                      will now save, turn off, and restore the
                      |'autowrite'| setting.
                Jun 13, 2014    * added visual map for gx use
                Jun 15, 2014    * (Enno Nagel) reported that with having hls
                      set and wide listing style in use, that the
                      b and w maps caused unwanted highlighting.
                Jul 05, 2014    * |netrw-mv| and |netrw-mX| commands included
                Jul 09, 2014    * |g:netrw_keepj| included, allowing optional
                      keepj
                Jul 09, 2014    * fixing bugs due to previous update
                Jul 21, 2014    * (Bruno Sutic) provided an updated
                      netrw_gitignore.vim
                Jul 30, 2014    * (Yavuz Yetim) reported that editing two
                      remote files of the same name caused the
                      second instance to have a "temporary"
                      name.  Fixed: now they use the same buffer.
                Sep 18, 2014    * (Yasuhiro Matsumoto) provided a patch which
                      allows scp and windows local paths to work.
                Oct 07, 2014    * gx (see |netrw-gx|) when atop a directory,
                      will now do |gf| instead
                Nov 06, 2014    * For cygwin: cygstart will be available for
                      netrw#BrowseX() to use if its executable.
                Nov 07, 2014    * Began support for file://... urls.  Will use
                      |g:netrw_file_cmd| (typically elinks or links)
                Dec 02, 2014    * began work on having mc (|netrw-mc|) copy
                      directories.  Works for linux machines,
                      cygwin+vim, but not for windows+gvim.
                Dec 02, 2014    * in tree mode, netrw was not opening
                      directories via symbolic links.
                Dec 02, 2014    * added resolved link information to
                      thin and tree modes
                Dec 30, 2014    * (issue#231) |:ls| was not showing
                      remote-file buffers reliably.  Fixed.
        v152:   Apr 08, 2014    * uses the |'noswapfile'| option (requires {{{2
```

```
                          vim 7.4 with patch 213)
                        * (Enno Nagel) turn |'rnu'| off in netrw
                          buffers.
                        * (Quinn Strahl) suggested that netrw
                          allow regular window splitting to occur,
                          thereby allowing |'equalalways'| to take
                          effect.
                        * (qingtian zhao) normally, netrw will
                          save and restore the |'fileformat'|;
                          however, sometimes that isn't wanted
              Apr 14, 2014    * whenever netrw marks a buffer as ro,
                          it will also mark it as nomod.
              Apr 16, 2014    * sftp protocol now supported by
                          netrw#Obtain(); this means that one
                          may use "mc" to copy a remote file
                          to a local file using sftp, and that
                          the |netrw-O| command can obtain remote
                          files via sftp.
                        * added [count]C support (see |netrw-C|)
              Apr 18, 2014    * when |g:netrw_chgwin| is one more than
                          the last window, then vertically split
                          the last window and use it as the
                          chgwin window.
              May 09, 2014    * SavePosn was "saving filename under cursor"
                          from a non-netrw window when using :Rex.
      v151:   Jan 22, 2014    * extended :Rexplore to return to buffer {{{2
                          prior to Explore or editing a directory
                        * (Ken Takata) netrw gave error when
                          clipboard was disabled.  Sol'n: Placed
                          several if has("clipboard") tests in.
                        * Fixed ftp://X@Y@Z// problem; X@Y now
                          part of user id, and only Z is part of
                          hostname.
                        * (A Loumiotis) reported that completion
                          using a directory name containing spaces
                          did not work.  Fixed with a retry in
                          netrw#Explore() which removes the
                          backslashes vim inserted.
              Feb 26, 2014    * :Rexplore now records the current file
                           using w:netrw_rexfile when returning via
                          |:Rexplore|
              Mar 08, 2014    * (David Kotchan) provided some patches
                          allowing netrw to work properly with
                          windows shares.
                        * Multiple one-liner help messages available
                          by pressing <cr> while atop the "Quick
                          Help" line
                        * worked on ShellCmdPost, FocusGained event
                          handling.
                        * |:Lexplore| path: will be used to update
                          a left-side netrw browsing directory.
              Mar 12, 2014    * |netrw-s-cr|: use <s-cr>  to close
                          tree directory implemented
              Mar 13, 2014    * (Tony Mechylynck) reported that using
                          the browser with ftp on a directory,
                          and selecting a gzipped txt file, that
                          an E19 occurred (which was issued by
                          gzip.vim).  Fixed.
              Mar 14, 2014    * Implemented :MF and :MT (see |netrw-:MF|
                          and |netrw-:MT|, respectively)
              Mar 17, 2014    * |:Ntree| [dir] wasn't working properly; fixed
              Mar 18, 2014    * Changed all uses of set to setl
```

```
                Mar 18, 2014    * Commented the netrw_btkeep line in
                                  s:NetrwOptionSave(); the effect is that
                                  netrw buffers will remain as |'bt'|=nofile.
                                  This should prevent swapfiles being created
                                  for netrw buffers.
                Mar 20, 2014    * Changed all uses of lcd to use s:NetrwLcd()
                                  instead.  Consistent error handling results
                                  and it also handles Window's shares
                                * Fixed |netrw-d| command when applied with ftp
                                * https: support included for netrw#NetRead()
    v150:       Jul 12, 2013    * removed a "keepalt" to allow ":e #" to {{{2
                                  return to the netrw directory listing
                Jul 13, 2013    * (Jonas Diemer) suggested changing
                                  a <cWORD> to <cfile>.
                Jul 21, 2013    * (Yuri Kanivetsky) reported that netrw's
                                  use of mkdir did not produce directories
                                  following the user's umask.
                Aug 27, 2013    * introduced |g:netrw_altfile| option
                Sep 05, 2013    * s:Strlen() now uses |strdisplaywidth()|
                                  when available, by default
                Sep 12, 2013    * (Selyano Baldo) reported that netrw wasn't
                                  opening some directories properly from the
                                  command line.
                Nov 09, 2013    * |:Lexplore| introduced
                                * (Ondrej Platek) reported an issue with
                                  netrw's trees (P15).  Fixed.
                                * (Jorge Solis) reported that "t" in
                                  tree mode caused netrw to forget its
                                  line position.
                Dec 05, 2013    * Added <s-leftmouse> file marking
                                  (see |netrw-mf|)
                Dec 05, 2013    * (Yasuhiro Matsumoto) Explore should use
                                  strlen() instead s:Strlen() when handling
                                  multibyte chars with strpart()
                                  (ie. strpart() is byte oriented, not
                                  display-width oriented).
                Dec 09, 2013    * (Ken Takata) Provided a patch; File sizes
                                  and a portion of timestamps were wrongly
                                  highlighted with the directory color when
                                  setting `:let g:netrw_liststyle=1` on Windows.
                                * (Paul Domaskis) noted that sometimes
                                  cursorline was activating in non-netrw
                                  windows.  All but one setting of cursorline
                                  was done via setl; there was one that was
                                  overlooked.  Fixed.
                Dec 24, 2013    * (esquifit) asked that netrw allow the
                                  /cygdrive prefix be a user-alterable
                                  parameter.
                Jan 02, 2014    * Fixed a problem with netrw-based ballon
                                  evaluation (ie. netrw#NetrwBaloonHelp()
                                  not having been loaded error messages)
                Jan 03, 2014    * Fixed a problem with tree listings
                                * New command installed: |:Ntree|
                Jan 06, 2014    * (Ivan Brennan) reported a problem with
                                  |netrw-P|.  Fixed.
                Jan 06, 2014    * Fixed a problem with |netrw-P| when the
                                  modified file was to be abandoned.
                Jan 15, 2014    * (Matteo Cavalleri) reported that when the
                                  banner is suppressed and tree listing is
                                  used, a blank line was left at the top of
                                  the display.  Fixed.
                Jan 20, 2014    * (Gideon Go) reported that, in tree listing
```

```
                                style, with a previous window open, that
                                the wrong directory was being used to open
                                a file.  Fixed. (P21)
        v149:   Apr 18, 2013    * in wide listing format, now have maps for {{{2
                                  w and b to move to next/previous file
                Apr 26, 2013    * one may now copy files in the same
                                  directory; netrw will issue requests for
                                  what names the files should be copied under
                Apr 29, 2013    * Trying Benzinger's problem again.  Seems
                                  that commenting out the BufEnter and
                                  installing VimEnter (only) works.  Weird
                                  problem!  (tree listing, vim -O Dir1 Dir2)
                May 01, 2013    * :Explore ftp://... wasn't working.  Fixed.
                May 02, 2013    * introduced |g:netrw_bannerbackslash| as
                                  requested by Paul Domaskis.
                Jul 03, 2013    * Explore now avoids splitting when a buffer
                                  will be hidden.
        v148:   Apr 16, 2013    * changed Netrw's Style menu to allow direct {{{2
                                  choice of listing style, hiding style, and
                                  sorting style

==============================================================================
13. Todo                                                *netrw-todo* {{{1

07/29/09 : banner       :|g:netrw_banner| can be used to suppress the
           suppression    banner.  This feature is new and experimental,
                          so its in the process of being debugged.
09/04/09 : "gp"         : See if it can be made to work for remote systems.
                        : See if it can be made to work with marked files.

==============================================================================
14. Credits                                             *netrw-credits* {{{1

        Vim editor      by Bram Moolenaar (Thanks, Bram!)
        dav             support by C Campbell
        fetch           support by Bram Moolenaar and C Campbell
        ftp             support by C Campbell <NdrOchip@ScampbellPfamily.AbizM>
        http            support by Bram Moolenaar <bram@moolenaar.net>
        rcp
        rsync           support by C Campbell (suggested by Erik Warendorph)
        scp             support by raf <raf@comdyn.com.au>
        sftp            support by C Campbell

        inputsecret(), BufReadCmd, BufWriteCmd contributed by C Campbell

        Jérôme Augé             -- also using new buffer method with ftp+.netrc
        Bram Moolenaar          -- obviously vim itself, :e and v:cmdarg use,
                                   fetch,...
        Yasuhiro Matsumoto      -- pointing out undo+0r problem and a solution
        Erik Warendorph         -- for several suggestions (g:netrw_..._cmd
                                   variables, rsync etc)
        Doug Claar              -- modifications to test for success with ftp
                                   operation

==============================================================================
Modelines: {{{1
 vim:tw=78:ts=8:ft=help:norl:fdm=marker
*pi_tar.txt*    For Vim version 8.0.  Last change: 2013 Apr 17

                      +====================+
                      | Tar File Interface |
                      +====================+
```

```
Author:  Charles E. Campbell  <NdrOchip@ScampbellPfamily.AbizM>
          (remove NOSPAM from Campbell's email first)
Copyright 2005-2012:                                    *tar-copyright*
        The VIM LICENSE (see |copyright|) applies to the files in this
        package, including tarPlugin.vim, tar.vim, and pi_tar.txt.  Like
        anything else that's except use "tar.vim" instead of "VIM".  Like
        anything else that's free, tar.vim and its associated files are
        provided *as is* and comes with no warranty of any kind, either
        expressed or implied.  No guarantees of merchantability.  No
        guarantees of suitability for any purpose.  By using this plugin, you
        agree that in no event will the copyright holder be liable for any
        damages resulting from the use of this software. Use at your own risk!

==============================================================================
1. Contents                                        *tar* *tar-contents*
   1. Contents.....................................................|tar-contents|
   2. Usage........................................................|tar-usage|
   3. Options......................................................|tar-options|
   4. History......................................................|tar-history|

==============================================================================
2. Usage                                           *tar-usage* *tar-manual*

   When one edits a *.tar file, this plugin will handle displaying a
   contents page.  Select a file to edit by moving the cursor atop
   the desired file, then hit the <return> key.  After editing, one may
   also write to the file.  Currently, one may not make a new file in
   tar archives via the plugin.

                                                   *:Vimuntar*
   VIMUNTAR~

   :Vimuntar [vimhome]

        This command copies, if necessary, the tarball to the .vim or vimfiles
        directory using the first writable directory in the |'runtimepath'|
        when no [vimhome] is specified.  Otherwise, the [vimhome] argument
        allows the user to specify that directory, instead.

        The copy is done using the command in *g:tar_copycmd* , which is >
                cp   for cygwin, unix, macunix
                copy for windows (32, 95, 64, 16)
<       The extraction is done with the command specified with
        *g:tar_extractcmd* , which by default is >
                "tar -xf"
<
                                                   *:TarDiff*
   DIFFERENCING SUPPORT~

   :TarDiff [filename]

        This command will attempt to show the differences between the tarball
        version of a file and the associated file on the system.  In order to
        find that file on the system, the script uses the path associated with
        the file mentioned in the tarball.  If the current directory is not
        correct for that path, :TarDiff will fail to find the associated file.

        If the [filename] is given, that that filename (and path) will be used
        to specify the associated file.
```

```
   PREVENTING LOADING~

   If for some reason you do not wish to use vim to examine tar'd files,
   you may put the following two variables into your <.vimrc> to prevent
   the tar plugin from loading: >

        let g:loaded_tarPlugin= 1
        let g:loaded_tar      = 1
<

==============================================================================
3. Options                                                  *tar-options*

   These options are variables that one may change, typically in one's
   <.vimrc> file.
                         Default
   Variable               Value   Explanation
   *g:tar_browseoptions*  "Ptf"   used to get a list of contents
   *g:tar_readoptions*    "OPxf"  used to extract a file from a tarball
   *g:tar_cmd*            "tar"   the name of the tar program
   *g:tar_nomax*            0     if true, file window will not be maximized
   *g:tar_secure*         undef   if exists:
                                        "--"s will be used to prevent unwanted
                                        option expansion in tar commands.
                                        Please be sure that your tar command
                                        accepts "--"; Posix compliant tar
                                        utilities do accept them.
                                  if not exists:
                                        The tar plugin will reject any tar
                                        files or member files that begin with
                                        "-"
                                  Not all tar's support the "--" which is why
                                  it isn't default.
   *g:tar_writeoptions*   "uf"    used to update/replace a file


==============================================================================
4. History                                                  *tar-history*


   v28 Jun 23, 2011 * a few more decompression options (tbz tb2 txz)
   v27 May 31, 2011 * moved cygwin detection before g:tar_copycmd handling
                    * inserted additional |:keepj| modifiers
                    * changed silent  to  sil!  (|:silent|)
   v26 Aug 09, 2010 * uses buffer-local instead of window variables to hold
                      tarfile name
                    * inserted keepj before 0d to protect jump list
   v25 Jun 19, 2010 * (Jan Steffens) added support for xz compression
   v24 Apr 07, 2009 * :Untarvim command implemented
       Sep 28, 2009 * Added lzma support
   v22 Aug 08, 2008 * security fixes
   v16 Jun 06, 2008 * tarfile:: used instead of tarfile: when editing files
                      inside tarballs.  Fixes a problem with tarballs called
                      things like c:\abc.tar. (tnx to Bill McCarthy)
   v14 May 09, 2008 * arno caught a security bug
       May 28, 2008 * various security improvements.  Now requires patch 299
                      which provides the fnameescape() function
       May 30, 2008 * allows one to view *.gz and *.bz2 files that are in
                    *.tar files.
   v12 Sep 07, 2007 * &shq now used if not the empty string for g:tar_shq
   v10 May 02, 2006 * now using "redraw then echo" to show messages, instead
                      of "echo and prompt user"
```

```
   v9 May 02, 2006 * improved detection of masquerading as tar file
   v8 May 02, 2006 * allows editing of files that merely masquerade as tar
                     files
   v7 Mar 22, 2006 * work on making tar plugin work across network
      Mar 27, 2006 * g:tar_cmd now available for users to change the name
                     of the tar program to be used.  By default, of course,
                     it's "tar".
   v6 Dec 21, 2005 * writing to files not in directories caused problems -
                     fixed (pointed out by Christian Robinson)
   v5 Nov 22, 2005 * report option workaround installed
   v3 Sep 16, 2005 * handles writing files in an archive back to the
                     archive
      Oct 18, 2005 * <amatch> used instead of <afile> in autocmds
      Oct 18, 2005 * handles writing to compressed archives
      Nov 03, 2005 * handles writing tarfiles across a network using
                     netrw#NetWrite()
   v2              * converted to use Vim7's new autoload feature by
                     Bram Moolenaar
   v1 (original)   * Michael Toren (see http://michael.toren.net/code/)

==============================================================================
vim:tw=78:ts=8:ft=help
*pi_vimball.txt*        For Vim version 8.0.  Last change: 2016 Apr 11

                               ---------------
                               Vimball Archiver
                               ---------------

Author:  Charles E. Campbell  <NdrOchip@ScampbellPfamily.AbizM>
          (remove NOSPAM from Campbell's email first)
Copyright: (c) 2004-2015 by Charles E. Campbell         *Vimball-copyright*
        The VIM LICENSE (see |copyright|) applies to the files in this
        package, including vimballPlugin.vim, vimball.vim, and pi_vimball.txt.
        except use "vimball" instead of "VIM".  Like anything else that's free,
        vimball.vim and its associated files are provided *as is* and comes with
        no warranty of any kind, either expressed or implied.  No guarantees
        of merchantability.  No guarantees of suitability for any purpose.  By
        using this plugin, you agree that in no event will the copyright
        holder be liable for any damages resulting from the use of this
        software. Use at your own risk!

==============================================================================
1. Contents                                *vba* *vimball* *vimball-contents*

        1. Contents.....................................: |vimball-contents|
        2. Vimball Introduction.........................: |vimball-intro|
        3. Vimball Manual...............................: |vimball-manual|
           MkVimball....................................: |:MkVimball|
           UseVimball...................................: |:UseVimball|
           RmVimball....................................: |:RmVimball|
        4. Vimball History..............................: |vimball-history|


==============================================================================
2. Vimball Introduction                                        *vimball-intro*

        Vimball is intended to make life simpler for users of plugins.  All
        a user needs to do with a vimball is: >
                vim someplugin.vba
                :so %
                :q
<       and the plugin and all its components will be installed into their
```

appropriate directories. Note that one doesn't need to be in any particular directory when one does this. Plus, any help for the plugin will also be automatically installed.

If a user has decided to use the AsNeeded plugin, vimball is smart enough to put scripts nominally intended for .vim/plugin/ into .vim/AsNeeded/ instead.

Removing a plugin that was installed with vimball is really easy: >
```
vim
:RmVimball someplugin
```
< This operation is not at all easy for zips and tarballs, for example.

Vimball examines the user's |'runtimepath'| to determine where to put the scripts. The first directory mentioned on the runtimepath is usually used if possible. Use >
```
:echo &rtp
```
< to see that directory.

==

3. Vimball Manual *vimball-manual*

MAKING A VIMBALL *:MkVimball*

```
:[range]MkVimball[!] filename [path]
```

The range is composed of lines holding paths to files to be included in your new vimball, omitting the portion of the paths that is normally specified by the runtimepath (|'rtp'|). As an example: >
```
plugin/something.vim
doc/something.txt
```
< using >
```
:[range]MkVimball filename
```
<
on this range of lines will create a file called "filename.vba" which can be used by Vimball.vim to re-create these files. If the "filename.vba" file already exists, then MkVimball will issue a warning and not create the file. Note that these paths are relative to your .vim (vimfiles) directory, and the files should be in that directory. The vimball plugin normally uses the first |'runtimepath'| directory that exists as a prefix; don't use absolute paths, unless the user has specified such a path.

If you use the exclamation point (!), then MkVimball will create the "filename.vba" file, overwriting it if it already exists. This behavior resembles that for |:w|.

If you wish to force slashes into the filename, that can also be done by using the exclamation mark (ie. :MkVimball! path/filename).

The tip at http://vim.wikia.com/wiki/Using_VimBall_with_%27Make%27 has a good idea on how to automate the production of vimballs using make.

MAKING DIRECTORIES VIA VIMBALLS *g:vimball_mkdir*

First, the |mkdir()| command is tried (not all systems support it).

If it doesn't exist, then if g:vimball_mkdir doesn't exist, it is set as follows: >
```
|g:netrw_localmkdir|, if it exists
```

```
          "mkdir"             , if it is executable
          "makedir"           , if it is executable
          Otherwise           , it is undefined.
<       One may explicitly specify the directory making command using
        g:vimball_mkdir.  This command is used to make directories that
        are needed as indicated by the vimball.


CONTROLLING THE VIMBALL EXTRACTION DIRECTORY            *g:vimball_home*

        You may override the use of the |'runtimepath'| by specifying a
        variable, g:vimball_home.

                                                        *vimball-extract*
                vim filename.vba

        Simply editing a Vimball will cause Vimball.vim to tell the user to
        source the file to extract its contents.

        Extraction will only proceed if the first line of a putative vimball
        file holds the "Vimball Archiver by Charles E. Campbell" line.

LISTING FILES IN A VIMBALL                                  *:VimballList*

                :VimballList

        This command will tell Vimball to list the files in the archive, along
        with their lengths in lines.

MANUALLY INVOKING VIMBALL EXTRACTION                        *:UseVimball*

                :UseVimball [path]

        This command is contained within the vimball itself; it invokes the
        vimball#Vimball() routine which is responsible for unpacking the
        vimball.  One may choose to execute it by hand instead of sourcing
        the vimball; one may also choose to specify a path for the
        installation, thereby overriding the automatic choice of the first
        existing directory on the |'runtimepath'|.

REMOVING A VIMBALL                                          *:RmVimball*

                :RmVimball vimballfile [path]

        This command removes all files generated by the specified vimball
        (but not any directories it may have made).  One may choose a path
        for de-installation, too (see |'runtimepath'|); otherwise, the
        default is the first existing directory on the |'runtimepath'|.
        To implement this, a file (.VimballRecord) is made in that directory
        containing a record of what files need to be removed for all vimballs
        used thus far.

PREVENTING LOADING

        If for some reason you don't want to be able to extract plugins
        using vimballs: you may prevent the loading of vimball.vim by
        putting the following two variables in your <.vimrc>: >

                let g:loaded_vimballPlugin= 1
                let g:loaded_vimball      = 1
<
WINDOWS                                                 *vimball-windows*
```

```
        Many vimball files are compressed with gzip.  Windows, unfortunately,
        does not come provided with a tool to decompress gzip'ped files.
        Fortunately, there are a number of tools available for Windows users
        to un-gzip files:
>
            Item     Tool/Suite   Free   Website
            ----     ----------   ----   -------
            7zip        tool       y     http://www.7-zip.org/
            Winzip      tool       n     http://www.winzip.com/downwz.htm
            unxutils    suite      y     http://unxutils.sourceforge.net/
            cygwin      suite      y     http://www.cygwin.com/
            GnuWin32    suite      y     http://gnuwin32.sourceforge.net/
            MinGW       suite      y     http://www.mingw.org/
<

==============================================================================
4. Vimball History                                    *vimball-history* {{{1

        37 : Jul 18, 2014 * (by request of T. Miedema) added augroup around
                            the autocmds in vimballPlugin.vim
             Jul 06, 2015 * there are two uses of tabc; changed to tabc!
        34 : Sep 22, 2011 * "UseVimball path" now supports a non-full path by
                            prepending the current directory to it.
        33 : Apr 02, 2011 * Gave priority to *.vmb over *.vba
                          * Changed silent! to sil! (shorter)
                          * Safed |'swf'| setting (during vimball extraction,
                            its now turned off)
        32 : May 19, 2010 * (Christian Brabrandt) :so someplugin.vba and
                            :so someplugin.vba.gz (and the other supported
                            compression types) now works
                          * (Jan Steffens) added support for xz compression
                          * fenc extraction was erroneously picking up the
                            end of the line number when no file encoding
                            was present.  Fixed.
                          * By request, beginning the switchover from the vba
                            extension to vmb.  Currently both are supported;
                            MkVimball, however, now will create *.vmb files.
             Feb 11, 2011 * motoyakurotsu reported an error with vimball's
                            handling of zero-length files
             Feb 18, 2016 * Changed =~ to =~# where appropriate
        30 : Dec 08, 2008 * fnameescape() inserted to protect error
                            messaging using corrupted filenames from
                            causing problems
                          * RmVimball supports filenames that would
                            otherwise be considered to have "magic"
                            characters (ie. Abc[1].vba)
             Feb 18, 2009 * s:Escape(), g:vimball_shq, and g:netrw_shq
                            removed (shellescape() used directly)
             Oct 05, 2009 * (Nikolai Weibull) suggested that MkVimball
                            be allowed to use slashes in the filename.
        26 : May 27, 2008 * g:vimball_mkdir usage installed.  Makes the
                            $HOME/.vim (or $HOME\vimfiles) directory if
                            necessary.
             May 30, 2008 * (tnx to Bill McCarthy) found and fixed a bug:
                            vimball wasn't updating plugins to AsNeeded/
                            when it should
        25 : Mar 24, 2008 * changed vimball#Vimball() to recognize doc/*.??x
                            files as help files, too.
             Apr 18, 2008 * RmVimball command is now protected by saving and
                            restoring settings -- in particular, acd was
                            causing problems as reported by Zhang Shuhan
```

```
        24 : Nov 15, 2007 * g:vimball_path_escape used by s:Path() to
                            prevent certain characters from causing trouble
                            (defunct: |fnameescape()| and |shellescape()|
                            now used instead)
        22 : Mar 21, 2007 * uses setlocal instead of set during BufEnter
        21 : Nov 27, 2006 * (tnx to Bill McCarthy) vimball had a header
                            handling problem and it now changes \s to /s
        20 : Nov 20, 2006 * substitute() calls have all had the 'e' flag
                            removed.
        18 : Aug 01, 2006 * vimballs now use folding to easily display their
                            contents.
                          * if a user has AsNeeded/somefile, then vimball
                            will extract plugin/somefile to the AsNeeded/
                            directory
        17 : Jun 28, 2006 * changes all \s to /s internally for Windows
        16 : Jun 15, 2006 * A. Mechelynck's idea to allow users to specify
                            installation root paths implemented for
                            UseVimball, MkVimball, and RmVimball.
                          * RmVimball implemented
        15 : Jun 13, 2006 * bugfix
        14 : May 26, 2006 * bugfixes
        13 : May 01, 2006 * exists("&acd") used to determine if the acd
                            option exists
        12 : May 01, 2006 * bugfix - the |'acd'| option is not always defined
        11 : Apr 27, 2006 * VimballList would create missing subdirectories that
                            the vimball specified were needed.  Fixed.
        10 : Apr 27, 2006 * moved all setting saving/restoration to a pair of
                            functions.  Included some more settings in them
                            which frequently cause trouble.
        9  : Apr 26, 2006 * various changes to support Windows' predilection
                            for backslashes and spaces in file and directory
                            names.
        7  : Apr 25, 2006 * bypasses foldenable
                          * uses more exe and less norm! (:yank :put etc)
                          * does better at insuring a "Press ENTER" prompt
                            appears to keep its messages visible
        4  : Mar 31, 2006 * BufReadPost seems to fire twice; BufReadEnter
                            only fires once, so the "Source this file..."
                            message is now issued only once.
        3  : Mar 20, 2006 * removed query, now requires sourcing to be
                            extracted (:so %).  Message to that effect
                            included.
                          * :VimballList  now shows files that would be
                            extracted.
        2  : Mar 20, 2006 * query, :UseVimball included
        1  : Mar 20, 2006 * initial release


==============================================================================
vim:tw=78:ts=8:ft=help:fdm=marker
*pi_zip.txt*    For Vim version 8.0.  Last change: 2016 Sep 13

                                +====================+
                                | Zip File Interface |
                                +====================+

Author:  Charles E. Campbell  <NdrOchip@ScampbellPfamily.AbizM>
          (remove NOSPAM from Campbell's email first)
Copyright: Copyright (C) 2005-2015 Charles E Campbell    *zip-copyright*
        The VIM LICENSE (see |copyright|) applies to the files in this
        package, including zipPlugin.vim, zip.vim, and pi_zip.vim.  except use
        "zip.vim" instead of "VIM".  Like anything else that's free, zip.vim
```

and its associated files are provided *as is* and comes with no warranty of any kind, either expressed or implied. No guarantees of merchantability. No guarantees of suitability for any purpose. By using this plugin, you agree that in no event will the copyright holder be liable for any damages resulting from the use of this software. Use at your own risk!

==
1. Contents *zip* *zip-contents*

1. Contents.....|zip-contents|
2. Usage.....|zip-usage|
3. Additional Extensions.....|zip-extension|
4. History.....|zip-history|

==
2. Usage *zip-usage* *zip-manual*

When one edits a *.zip file, this plugin will handle displaying a contents page. Select a file to edit by moving the cursor atop the desired file, then hit the <return> key. After editing, one may also write to the file. Currently, one may not make a new file in zip archives via the plugin.

zip-x

x : may extract a listed file when the cursor is atop it

OPTIONS

g:zip_nomax

If this variable exists and is true, the file window will not be automatically maximized when opened.

g:zip_shq

Different operating systems may use one or more shells to execute commands. Zip will try to guess the correct quoting mechanism to allow spaces and whatnot in filenames; however, if it is incorrectly guessing the quote to use for your setup, you may use >
```
g:zip_shq
```
< which by default is a single quote under Unix (') and a double quote under Windows ("). If you'd rather have no quotes, simply set g:zip_shq to the empty string (let g:zip_shq= "") in your <.vimrc>.

g:zip_unzipcmd

Use this option to specify the program which does the duty of "unzip". It's used during browsing. By default: >
```
let g:zip_unzipcmd= "unzip"
```
<

g:zip_zipcmd

Use this option to specify the program which does the duty of "zip". It's used during the writing (updating) of a file already in a zip file; by default: >
```
let g:zip_zipcmd= "zip"
```
<

g:zip_extractcmd

This option specifies the program (and any options needed) used to extract a file from a zip archive. By default, >
```
let g:zip_extractcmd= g:zip_unzipcmd
```
<

PREVENTING LOADING~

If for some reason you do not wish to use vim to examine zipped files,

```
   you may put the following two variables into your <.vimrc> to prevent
   the zip plugin from loading: >

        let g:loaded_zipPlugin= 1
        let g:loaded_zip      = 1
<

==============================================================================
3. Additional Extensions                                        *zip-extension*

   Apparently there are a number of archivers which generate zip files that
   don't use the .zip extension (.jar, .xpi, etc).  To handle such files,
   place a line in your <.vimrc> file: >

        au BufReadCmd *.jar,*.xpi call zip#Browse(expand("<amatch>"))
<
   One can simply extend this line to accommodate additional extensions that
   should be treated as zip files.

   Alternatively, one may change *g:zipPlugin_ext* in one's .vimrc.
   Currently (11/30/15) it holds: >

        let g:zipPlugin_ext= '*.zip,*.jar,*.xpi,*.ja,*.war,*.ear,*.celzip,
       \ *.oxt,*.kmz,*.wsz,*.xap,*.docx,*.docm,*.dotx,*.dotm,*.potx,*.potm,
       \ *.ppsx,*.ppsm,*.pptx,*.pptm,*.ppam,*.sldx,*.thmx,*.xlam,*.xlsx,*.xlsm,
       \ *.xlsb,*.xltx,*.xltm,*.xlam,*.crtx,*.vdw,*.glox,*.gcsx,*.gqsx,*.epub'

==============================================================================
4. History                                                *zip-history* {{{1
   v28 Oct 08, 2014 * changed the sanity checks for executables to reflect
                      the command actually to be attempted in zip#Read()
                      and zip#Write()
                    * added the extraction of a file capability
       Nov 30, 2015 * added *.epub to the |g:zipPlugin_ext| list
       Sep 13, 2016 * added *.apk to the |g:zipPlugin_ext| list and
                      sorted the suffices.
   v27 Jul 02, 2013 * sanity check: zipfile must have "PK" as its first
                      two bytes.
                    * modified to allow zipfile: entries in quickfix lists
   v26 Nov 15, 2012 * (Jason Spiro) provided a lot of new extensions that
                      are synonyms for .zip
   v25 Jun 27, 2011 * using keepj with unzip -Z
                      (consistent with the -p variant)
                    * (Ben Staniford) now uses
                        has("win32unix") && executable("cygpath")
                      before converting to cygwin-style paths
   v24 Jun 21, 2010 * (Cédric Bosdonnat) unzip seems to need its filenames
                      fnameescape'd as well as shellquote'd
                    * (Motoya Kurotsu) inserted keepj before 0d to protect
                      jump list
   v17 May 09, 2008 * arno caught a security bug
   v15 Sep 07, 2007 * &shq now used if not the empty string for g:zip_shq
   v14 May 07, 2007 * using b:zipfile instead of w:zipfile to avoid problem
                      when editing alternate file to bring up a zipfile
   v10 May 02, 2006 * now using "redraw then echo" to show messages, instead
                      of "echo and prompt user"
                    * g:zip_shq provided to allow for quoting control for the
                      command being passed via :r! ... commands.
   v8 Apr 10, 2006 * Bram Moolenaar reported that he received an error message
                     due to "Pattern not found: ^.*\%0c"; this was caused by
                     stridx finding a Name... at the beginning of the line;
                     zip.vim tried 4,$s/^.*\%0c//, but that doesn't work.
```

```
                    Fixed.
   v7 Mar 22, 2006 * escaped some characters that can cause filename handling
                    problems.
   v6 Dec 21, 2005 * writing to files not in directories caused problems -
                    fixed (pointed out by Christian Robinson)
   v5 Nov 22, 2005 * report option workaround installed
   v3 Oct 18, 2005 * <amatch> used instead of <afile> in autocmds
   v2 Sep 16, 2005 * silenced some commands (avoiding hit-enter prompt)
                   * began testing under Windows; works thus far
                   * filetype detection fixed
      Nov 03, 2005 * handles writing zipfiles across a network using
                    netrw#NetWrite()
   v1 Sep 15, 2005 * Initial release, had browsing, reading, and writing

==============================================================================
vim:tw=78:ts=8:ft=help:fdm=marker
```