fugitive.txt A Git wrapper so awesome, it should be illegal

Author: Tim Pope <<http://tpo.pe/>>

License: Same terms as Vim itself (see license)

This plugin is only available if **'compatible'** is not set.

**INTRODUCTION** fugitive

Whenever you edit a file from a Git repository, a set of commands is defined

that serve as a gateway to Git.

**COMMANDS** fugitive-commands

These commands are local to the buffers in which they work (generally, buffers

that are part of Git repositories).

:Git fugitive-:G

:Git {args} Run an arbitrary git command. Similar to :!git [args]

:G {args} but chdir to the repository tree first. For some

subcommands, a Fugitive command is called instead.

:Git! {args} Like :Git, but capture the output into a temp file,

:Git --paginate {args} and :split that temp file. Use :0Git to

:Git -p {args} :edit the temp file instead. A temp file is always

used for diff and log commands.

:Gstatus

:Git Bring up a summary window vaguely akin to git-status.

:G Press g? or see fugitive-maps for usage.

:Gstatus

:Git-commit :Gcommit

:Git commit [args] A wrapper around git-commit. Unless the arguments

:Gcommit [args] given would skip the invocation of an editor (e.g.,

-m), a split window will be used to obtain a commit

message, or a new tab if -v is given. Write and close

the window (:wq) to finish the commit. To cancel, use

an empty message.

:Git-revert :Grevert

:Git revert [args] A wrapper around git-revert. Similar to :Gcommit.

:Grevert [args]

:Git-merge :Gmerge

:Git merge [args] Calls git-merge and loads errors and conflicted files

:Gmerge [args] into the quickfix list. Opens a :Gcommit style

split window for the commit message if the merge

succeeds. If called during a merge conflict, the

conflicted files from the current index are loaded

into the quickfix list.

:Git-pull :Gpull

:Git pull [args] Like :Gmerge, but for git-pull.

:Gpull [args]

:Git-rebase :Grebase

:Git rebase [args] Like :Gmerge, but for git-rebase. Interactive

:Grebase [args] rebase is experimentally supported.

:Git-push :Gpush

:Git push [args] Invoke git-push, load the results into the quickfix

:Gpush [args] list, and invoke :cwindow to reveal any errors.

:Dispatch is used if available for asynchronous

invocation.

:Git-fetch :Gfetch

:Git fetch [args] Like :Gpush, but for git-fetch.

:Gfetch [args]

:Git-blame :Gblame

:Git blame [flags] Run git-blame [flags] on the current file and open the

:Gblame [flags] results in a scroll-bound vertical split. The

following maps, which work on the cursor line commit

where sensible, are provided:

g? show this help

A resize to end of author column

C resize to end of commit column

D resize to end of date/time column

gq close blame, then :Gedit to return to work

tree version

<CR> close blame, and jump to patch that added line

(or directly to blob for boundary commit)

o jump to patch or blob in horizontal split

O jump to patch or blob in new tab

p jump to patch or blob in preview window

- reblame at commit

~ reblame at [count]th first grandparent

P reblame at [count]th parent (like HEAD^[count])

:[range]Gblame [flags] If a range is given, just that part of the file will

:Gblame [flags] {file} be blamed, and a horizontal split without

:Git blame ... scrollbinding is used. You can also give an arbitrary

filename.

:Ggrep :Gcgrep :Git-grep

:Ggrep[!] [args] :grep[!] with git-grep as **'grepprg'**.

:Git[!] grep [args]

:Glgrep

:Glgrep[!] [args] :lgrep[!] with git-grep as **'grepprg'**.

:0Git[!] grep [args]

:Git-difftool

:Git[!] difftool [args] Invoke git diff [args] and load the changes into the

quickfix list. Each changed hunk gets a separate

quickfix entry unless you pass an option like

--name-only or --name-status. Jumps to the first

change unless [!] is given.

:Git difftool -y [args] Invoke git diff [args], open each changed file in a

new tab, and invoke :Gdiffsplit! against the

appropriate commit.

:Git-mergetool

:Git mergetool [args] Like :Git-difftool, but target merge conflicts.

:Gclog :Glog

:Gclog[!] [args] Use git-log [args] to load the commit history into the

:Glog[!] [args] quickfix list. Jumps to the first commit unless [!]

is given.

:{range}Gclog[!] [args] Use git-log -L to load previous revisions of the given

range of the current file into the quickfix list.

The cursor is positioned on the first line of the

first diff hunk for each commit. Use :0Gclog to

target the entire file.

:Gllog

:Gllog [args] Like :Gclog, but use the location list instead of the

quickfix list.

:Gcd

:Gcd [directory] :cd relative to the repository.

:Glcd

:Glcd [directory] :lcd relative to the repository.

:Gedit fugitive-:Ge

:Gedit [object] :edit a fugitive-object.

:Gsplit

:Gsplit [object] :split a fugitive-object.

:Gvsplit

:Gvsplit [object] :vsplit a fugitive-object.

:Gtabedit

:Gtabedit [object] :tabedit a fugitive-object.

:Gpedit

:Gpedit [object] :pedit a fugitive-object.

:Gsplit! [args] :Gsplit! :Gvsplit!

:Gvsplit! [args] :Gtabedit! :Gpedit!

:Gtabedit! [args] Capture the output of git [args] to a temp file and

:Gpedit! [args] open it in a split, tab, or preview window. Use

:0Gsplit! to suppress the split and open it in the

current window.

:Gread fugitive-:Gr

:Gread [object] Empty the buffer and :read a fugitive-object.

When the argument is omitted, this is similar to

git-checkout on a work tree file or git-add on a stage

file, but without writing anything to disk.

:{range}Gread [object] :read in a fugitive-object after {range}.

:Gread! fugitive-:Gr!

:Gread! [args] Empty the buffer and :read the output of a Git

command. For example, :Gread! show HEAD:%.

:{range}Gread! [args] :read the output of a Git command after {range}.

:Gwrite fugitive-:Gw

:Gwrite Write to the current file's path and stage the results.

When run in a work tree file, it is effectively git

add. Elsewhere, it is effectively git-checkout. A

great deal of effort is expended to behave sensibly

when the work tree or index version of the file is

open in another buffer.

:Gwrite {path} You can give :Gwrite an explicit path of where in

the work tree to write. You can also give a path like

:0:foo.txt or :0:% to write to just that stage in

the index.

:Gwq

:Gwq [path] Like :Gwrite followed by :quit if the write

succeeded.

:Gwq! [path] Like :Gwrite! followed by :quit! if the write

succeeded.

:Gdiffsplit

:Gdiffsplit [object] Perform a vimdiff against the given file, or if a

commit is given, the current file in that commit.

With no argument, the version in the index or work

tree is used. The newer of the two files is placed to

the right or bottom, depending on **'diffopt'** and the

width of the window relative to **'textwidth'**. Use

Vim's do and dp to stage and unstage changes.

:Gdiffsplit!

:Gdiffsplit! Diff against any and all direct ancestors, retaining

focus on the current window. During a merge conflict,

this is a three-way diff against the "ours" and

"theirs" ancestors. Additional d2o and d3o maps are

provided to obtain the hunk from the "ours" or

"theirs" ancestor, respectively.

:Gdiffsplit! {object} Like :Gdiffsplit, but retain focus on the current

window.

:Gvdiffsplit

:Gvdiffsplit [object] Like :Gdiffsplit, but always split vertically.

:Ghdiffsplit :Gsdiff

:Ghdiffsplit [object] Like :Gdiffsplit, but always split horizontally.

:Gmove

:Gmove {destination} Wrapper around git-mv that renames the buffer

afterward. Add a ! to pass -f.

:Grename

:Grename {destination} Like :Gmove but operates relative to the parent

directory of the current file.

:Gdelete

:Gdelete Wrapper around git-rm that deletes the buffer

afterward. When invoked in an index file, --cached is

passed. Add a ! to pass -f and forcefully discard the

buffer.

:Gremove

:Gremove Like :Gdelete, but keep the (now empty) buffer around.

:Gbrowse

:Gbrowse Open the current file, blob, tree, commit, or tag

in your browser at the upstream hosting provider.

If a range is given, it is appropriately appended to

the URL as an anchor.

Upstream providers can be added by installing an

appropriate Vim plugin. For example, GitHub can be

supported by installing rhubarb.vim, available at

<<https://github.com/tpope/vim-rhubarb>>.

:Gbrowse {object} Like :Gbrowse, but for a given fugitive-object.

:Gbrowse [...]@{remote} Force using the given remote rather than the remote

for the current branch. The remote is used to

determine which upstream repository to link to.

:{range}Gbrowse [args] Appends an anchor to the URL that emphasizes the

selected lines. This also forces the URL to include a

commit rather than a branch name so it remains valid

if the file changes. You can give a range of "0" to

force this behavior without including an anchor.

:[range]Gbrowse! [args] Like :Gbrowse, but put the URL on the clipboard rather

than opening it.

**MAPS** fugitive-maps

These maps are available in both the :Gstatus buffer and Fugitive object

buffers, although not all maps make sense in all buffers. Mappings that

operate on the file or hunk under the cursor are generally available in visual

mode to operate on multiple files or partial hunks.

fugitive-staging-maps

Staging/unstaging maps

fugitive\_s

s Stage (add) the file or hunk under the cursor.

fugitive\_u

u Unstage (reset) the file or hunk under the cursor.

fugitive\_-

- Stage or unstage the file or hunk under the cursor.

fugitive\_U

U Unstage everything.

fugitive\_X

X Discard the change under the cursor. This uses

checkout or clean under the hood. A command is

echoed that shows how to undo the change. Consult

:messages to see it again. You can use this during

a merge conflict do discard "our" changes (--theirs)

in the "Unstaged" section or discard "their" changes

(--ours) in the "Staged" section.

fugitive\_=

= Toggle an inline diff of the file under the cursor.

fugitive\_>

> Insert an inline diff of the file under the cursor.

fugitive\_<

< Remove the inline diff of the file under the cursor.

fugitive\_gI

gI Open .git/info/exclude in a split and add the file

under the cursor. Use a count to open .gitignore.

fugitive\_I

I Invoke :Git add --patch or reset --patch on the file

P under the cursor. On untracked files, this instead

calls :Git add --intent-to-add.

fugitive\_d

Diff maps

fugitive\_dp

dp Invoke :Git! diff on the file under the cursor.

Deprecated in favor of inline diffs.

fugitive\_dd

dd Perform a :Gdiffsplit on the file under the cursor.

fugitive\_dv

dv Perform a :Gvdiffsplit on the file under the cursor.

fugitive\_ds fugitive\_dh

ds Perform a :Ghdiffsplit on the file under the cursor.

dh

fugitive\_dq

dq Close all but one diff buffer, and :diffoff! the

last one.

fugitive\_d?

d? Show this help.

fugitive-navigation-maps

Navigation maps

fugitive\_<CR>

<CR> Open the file or fugitive-object under the cursor.

in a blob, this and similar maps jump to the patch

from the diff where this was added, or where it was

removed if a count was given. If the line is still in

the work tree version, passing a count takes you to

it.

fugitive\_o

o Open the file or fugitive-object under the cursor in

a new split.

fugitive\_gO

gO Open the file or fugitive-object under the cursor in

a new vertical split.

fugitive\_O

O Open the file or fugitive-object under the cursor in

a new tab.

fugitive\_p

p Open the file or fugitive-object under the cursor in

a preview window. In the status buffer, 1p is

required to bypass the legacy usage instructions.

fugitive\_~

~ Open the current file in the [count]th first ancestor.

fugitive\_P

P Open the current file in the [count]th parent.

fugitive\_C

C Open the commit containing the current file.

fugitive\_CTRL-P fugitive\_(

( Jump to the previous file, hunk, or revision.

fugitive\_CTRL-N fugitive\_)

) Jump to the next file, hunk, or revision.

fugitive\_[c

[c Jump to previous hunk, expanding inline diffs

automatically. (This shadows the Vim built-in [c

that provides a similar operation in diff mode.)

fugitive\_]c

]c Jump to next hunk, expanding inline diffs

automatically. (This shadows the Vim built-in ]c

that provides a similar operation in diff mode.)

fugitive\_[/ fugitive\_[m

[/ Jump to previous file, collapsing inline diffs

[m automatically. (Mnemonic: "/" appears in filenames,

"m" appears in "filenames".)

fugitive\_]/ fugitive\_]m

]/ Jump to next file, collapsing inline diffs

]m automatically. (Mnemonic: "/" appears in filenames,

"m" appears in "filenames".)

fugitive\_i

i Jump to the next file or hunk, expanding inline diffs

automatically.

fugitive\_[[

[[ Jump [count] sections backward.

fugitive\_]]

]] Jump [count] sections forward.

fugitive\_[]

[] Jump [count] section ends backward.

fugitive\_][

][ Jump [count] section ends forward.

fugitive\_star

\* One the first column of a + or - diff line, search for

the corresponding - or + line. Otherwise, defer to

built-in star.

fugitive\_#

# Same as "\*", but search backward.

fugitive\_gu

gu Jump to file [count] in the "Untracked" or "Unstaged"

section.

fugitive\_gU

gU Jump to file [count] in the "Unstaged" section.

fugitive\_gs

gs Jump to file [count] in the "Staged" section.

fugitive\_gp

gp Jump to file [count] in the "Unpushed" section.

fugitive\_gP

gP Jump to file [count] in the "Unpulled" section.

fugitive\_gr

gr Jump to file [count] in the "Rebasing" section.

fugitive\_gi

gi Open .git/info/exclude in a split. Use a count to

open .gitignore.

fugitive\_c

Commit maps

cc Create a commit.

ca Amend the last commit and edit the message.

ce Amend the last commit without editing the message.

cw Reword the last commit.

cvc Create a commit with -v.

cva Amend the last commit with -v

cf Create a fixup! commit for the commit under the

cursor.

cF Create a fixup! commit for the commit under the

cursor and immediately rebase it.

cs Create a squash! commit for the commit under the

cursor.

cS Create a squash! commit for the commit under the

cursor and immediately rebase it.

cA Create a squash! commit for the commit under the

cursor and edit the message.

c<Space> Populate command line with ":Git commit ".

fugitive\_cr

crc Revert the commit under the cursor.

crn Revert the commit under the cursor in the index and

work tree, but do not actually commit the changes.

cr<Space> Populate command line with ":Git revert ".

fugitive\_cm

cm<Space> Populate command line with ":Git merge ".

c? Show this help.

fugitive\_cb

fugitive\_co

Checkout/branch maps

coo Check out the commit under the cursor.

cb<Space> Populate command line with ":Git branch ".

co<Space> Populate command line with ":Git checkout ".

cb? Show this help.

co?

fugitive\_cz

Stash maps

czz Push stash. Pass a [count] of 1 to add

--include-untracked or 2 to add --all.

czw Push stash of worktree. Like czz with

--include-index.

czA Apply topmost stash, or stash@{count}.

cza Apply topmost stash, or stash@{count}, preserving the

index.

czP Pop topmost stash, or stash@{count}.

czp Pop topmost stash, or stash@{count}, preserving the

index.

cz<Space> Populate command line with ":Git stash ".

cz? Show this help.

fugitive\_r

Rebase maps

ri Perform an interactive rebase. Uses ancestor of

u commit under cursor as upstream if available.

rf Perform an autosquash rebase without editing the todo

list. Uses ancestor of commit under cursor as

upstream if available.

ru Perform an interactive rebase against @{upstream}.

rp Perform an interactive rebase against @{push}.

rr Continue the current rebase.

rs Skip the current commit and continue the current

rebase.

ra Abort the current rebase.

re Edit the current rebase todo list.

rw Perform an interactive rebase with the commit under

the cursor set to reword.

rm Perform an interactive rebase with the commit under

the cursor set to edit.

rd Perform an interactive rebase with the commit under

the cursor set to drop.

r<Space> Populate command line with ":Git rebase ".

r? Show this help.

fugitive-misc-maps

Miscellaneous maps

fugitive\_gq fugitive\_q

gq Close the status buffer.

fugitive\_.

. Start a : command line with the file under the

cursor prepopulated.

fugitive\_g?

g? Show help for fugitive-maps.

fugitive-global-maps

Global maps

fugitive\_c\_CTRL-R\_CTRL-G

<C-R><C-G> On the command line, recall the path to the current

fugitive-object (that is, a representation of the

object recognized by :Gedit).

fugitive\_y\_CTRL-G

["x]y<C-G> Yank the path to the current fugitive-object.

g:fugitive\_no\_maps

Global maps can be disabled with the g:fugitive\_no\_maps option.

let g:fugitive\_no\_maps = 1

**SPECIFYING OBJECTS** fugitive-object fugitive-revision

Fugitive objects are either work tree files or Git revisions as defined in the

"SPECIFYING REVISIONS" section in the git-rev-parse man page, with expansions

inspired by cmdline-special layered on top. For commands that accept an

optional object, the default is the file in the index for work tree files and

the work tree file for everything else. Example objects follow.

Object Meaning

@ The commit referenced by @ aka HEAD

master The commit referenced by master

master^ The parent of the commit referenced by master

master...other The merge base of master and other

master: The tree referenced by master

./master The file named master in the working directory

:(top)master The file named master in the work tree

Makefile The file named Makefile in the work tree

@^:Makefile The file named Makefile in the parent of HEAD

:Makefile The file named Makefile in the index (writable)

@~2:% The current file in the grandparent of HEAD

:% The current file in the index

:1:% The current file's common ancestor during a conflict

:2:# The alternate file in the target branch during a conflict

:3:#5 The file from buffer #5 in the merged branch during a conflict

! The commit owning the current file

!:Makefile The file named Makefile in the commit owning the current file

!3^2 The second parent of the commit owning buffer #3

.git/config The repo config file

: Same as :Gstatus

**STATUSLINE** fugitive-statusline

FugitiveStatusline() fugitive#statusline()

Add %{FugitiveStatusline()} to your statusline to get an indicator including

the current branch and the currently edited file's commit. If you don't have

a statusline, this one matches the default when **'ruler'** is set:

set statusline=%<%f\ %h%m%r%{FugitiveStatusline()}%=%-14.(%l,%c%V%)\ %P

FugitiveHead(...) fugitive#head(...)

Use FugitiveHead() to return the name of the current branch. If the current

HEAD is detached, FugitiveHead() will return the empty string, unless the

optional argument is given, in which case the hash of the current commit will

be truncated to the given number of characters.

**ABOUT** fugitive-about

Grab the latest version or report a bug on GitHub:

<https://github.com/tpope/vim-fugitive>

vim:tw=78:et:ft=help:norl: