

Magit-Popup User Manual

for version 2.1

Jonas Bernoulli

Copyright (C) 2015 Jonas Bernoulli <jonas@bernoul.li>

You can redistribute this document and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This document is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

Table of Contents

1	Introduction	1
2	Usage	3
2.1	Customizing existing popups	4
2.2	Other options	5
3	Defining prefix and suffix commands	7
3.1	Defining prefix commands	7
3.2	Defining suffix commands	9

This library was written as a replacement for `magit-key-mode` which was used in Magit releases before 2.1.0. It is used to implement all "popups" in the current Magit release but a future release will switch to yet another implementation.

This library does not depend on any other Magit libraries and it is distributed as a separate package, which makes it possible to use it in packages that are not related to Magit. But keep in mind that it will be deprecated eventually.

2 Usage

Every popup buffers created with a prefix command contains a section named "Actions" listing the available suffix commands. Most buffers also contain a "Switches" and/or an "Options" section which list the two types of infix arguments separately.

Switches are arguments that can be toggled on or off. When a switch is active then it is shown in color, when it is off then it is shown in gray (of course the details depend on the color theme in use).

Options are arguments that have a value. When an option has a value then that is shown after the option itself. Because for some options the empty string is a valid value, options are additionally colorized like switches to indicate whether they are active or not.

The events bound to suffix commands are always single alphabetic characters. The bindings for arguments are always two events long. For switches the first key is always -, for options it is always =. The second key is always an alphabetic character.

By default popup buffers also feature a section listing commands common to all popups. To avoid conflicts with suffix commands, the bindings of these common commands are not alphabetic characters. This section is shown by default so that documentation-resistant users get a change to notice them.

`magit-popup-show-common-commands` [User Option]

This option controls whether the section which lists the commands that are common to all popups is initially show. We recommend you set this to `nil` - after you have memorized that it can be shown on demand using `C-t`.

`C-t` (`magit-popup-toggle-show-common-commands`)

Show or hide the section listing the commands shared by all popups.

`C-g` (`magit-popup-quit`)

Quit popup buffer without invoking a suffix command.

Without further action, setting arguments only affects the next suffix command. Invoking the same prefix command again resets the arguments to their default value, but the defaults can be changed directly from the popup buffer itself. For a prefix command named `NAME-popup` the default values are stored as the value of the custom option named `NAME-arguments`. While this option can be customized using the Custom interface, it is better to do so directly from the popup buffer.

`C-c C-c` (`magit-popup-set-default-arguments`)

This sets the default value for the arguments for the current popup.

Then the popup buffer is closed without invoking a suffix command; unless a prefix argument is used in which case the popup remains open.

`C-x C-s` (`magit-popup-save-default-arguments`)

This sets the default value for the arguments for the current popup and saves it for future Emacs sessions.

Then the popup buffer is closed without invoking an action; unless a prefix argument is used in which case the popup remains open.

It is also possible to add additional arguments and commands to an existing popup, but that cannot be done directly from the popup (or the Custom interface). See [Section 2.1 \[Customizing existing popups\]](#), page 4.

Documentation about a popup's arguments and commands can be shown directly from the popup.

`C-h i` (magit-popup-info)
Show this manual.

`?` (magit-popup-help)
This command reads a key sequence and then shows the documentation of the argument or command that sequence is bound to. In other words type the same keys that you would use to invoke the argument or command, but prefix the sequence with `?`.

For suffix commands this shows the doc-string. For arguments this command can only show something for popups that have an associated man-page. If the man-page is set, then this command displays it in a separate buffer and puts point on the entry about the argument in question.

The buffer which is used to display the documentation is selected. Simply press `q` to leave that buffer and restore the old window configuration.

While it isn't very useful, it is possible to move around in a popup buffer using `C-p` and `C-n`, and to invoke the argument or command at point using `RET`. But it is much more efficient to use the dedicated key bindings instead, so these commands are not listed in popup buffers along with the other common commands.

2.1 Customizing existing popups

It is possible to define additional infix arguments and suffix commands to an existing popup using the following functions.

`magit-define-popup-switch` *popup key desc switch &optional enable* [Function]
at prepend

In POPUP, define KEY as SWITCH.

POPUP is a popup command defined using `magit-define-popup`. SWITCH is a string representing an argument that takes no value. KEY is a character representing the second event in the sequence of keystrokes used to toggle the argument. (The first event, the prefix, is shared among all switches, defaults to `-`, and can be changed in `magit-popup-mode-keymap`).

DESC is a string describing the purpose of the argument, it is displayed in the popup. If optional ENABLE is non-nil then the switch is on by default.

SWITCH is inserted after all other switches already defined for POPUP, unless optional PREPEND is non-nil, in which case it is placed first. If optional AT is non-nil then it should be the KEY of another switch already defined for POPUP, the argument is then placed before or after AT, depending on PREPEND.

`magit-define-popup-option` *popup key desc option &optional reader* [Function]
value at prepend

In POPUP, define KEY as OPTION.

POPUP is a popup command defined using `magit-define-popup`. OPTION is a string representing an argument that takes a value. KEY is a character representing the second event in the sequence of keystrokes used to set the argument's value. (The first event, the prefix, is shared among all options, defaults to =, and can be changed in `magit-popup-mode-keymap`).

DESC is a string describing the purpose of the argument, it is displayed in the popup. If optional VALUE is non-nil then the option is on by default, and VALUE is its default value.

OPTION is inserted after all other options already defined for POPUP, unless optional PREPEND is non-nil, in which case it is placed first. If optional AT is non-nil then it should be the KEY of another option already defined for POPUP, the argument is then placed before or after AT, depending on PREPEND.

`magit-define-popup-action` *popup key desc command &optional at* [Function]
prepend

In POPUP, define KEY as COMMAND.

POPUP is a popup command defined using `magit-define-popup`. COMMAND can be any command but should usually consume the popup arguments in its `interactive` form. KEY is a character representing the event used to invoke the action, i.e. to interactively call the COMMAND.

DESC is a string describing the purpose of the action, it is displayed in the popup.

COMMAND is inserted after all other commands already defined for POPUP, unless optional PREPEND is non-nil, in which case it is placed first. If optional AT is non-nil then it should be the KEY of another command already defined for POPUP, the command is then placed before or after AT, depending on PREPEND.

`magit-change-popup-key` *popup type from to* [Function]

In POPUP, bind TO to what FROM was bound to. TYPE is one of `:action`, `:switch`, or `:option`. Bind TO and unbind FROM, both are characters.

`magit-remove-popup-key` *popup type key* [Function]

In POPUP, remove KEY's binding of TYPE. POPUP is a popup command defined using `magit-define-popup`. TYPE is one of `:action`, `:switch`, or `:option`. KEY is the character which is to be unbound.

2.2 Other options

`magit-popup-use-prefix-argument` [User Option]

This option controls the effect that the use of a prefix argument before entering a popup has. The **intended** default is **default**, but the **actual** default is **disabled**. This is necessary because the old popup implementation did simply forward such a pre-popup prefix argument to the suffix command invoked from the popup, and changing that without users being aware of it could lead to tears.

- **disabled**

Bring up a Custom option buffer so that the user reads this and then makes an informed choice.

- **default**
With a prefix argument directly invoke the popup's default action (an Emacs command), instead of bringing up the popup.
- **popup**
With a prefix argument bring up the popup, otherwise directly invoke the popup's default action.
- **nil**
Ignore prefix arguments.

magit-popup-manpage-package

[User Option]

The Emacs package used to display man-pages, one of **man** or **woman**.

3 Defining prefix and suffix commands

If you write an extension for Magit then you should use this library now and later when `transient` is released port to that.

If you are considering using this library to define popups for packages not related to Magit, then keep in mind that it will be superseded eventually. Once `transient` has been released I will only fix bugs in `magit-popup` but not implement any new features.

Also consider using `hydra` instead. To some extent `magit-popup` and `hydra` are similar but have a different focus. The main purpose of `magit-popup` is to pass infix arguments to suffix commands. If all you need is a command dispatcher then you are better off using `hydra`. Of course `hydra` may also be a better fit not only because of the features it lacks, but also because of the features it provides, which are in turn missing from `magit-popup`.

Here is an example of how one defines a prefix command along with its infix arguments, and then also one of its suffix commands.

```
;;;###autoload (autoload 'magit-tag-popup "magit" nil t)
(magit-define-popup magit-tag-popup
  "Show popup buffer featuring tagging commands."
  'magit-commands
  :man-page "git-tag"
  :switches '((?a "Annotate" "--annotate")
             (?s "Sign"      "--sign")
             (?f "Force"    "--force"))
  :actions '((?t "Create"   magit-tag)
            (?k "Delete"   magit-tag-delete)
            (?p "Prune"    magit-tag-prune))
  :default-action 'magit-tag)

;;;###autoload
(defun magit-tag (name rev &optional args)
  "Create a new tag with the given NAME at REV."
  (interactive (list (magit-read-tag "Tag name")
                    (magit-read-branch-or-commit "Place tag on")
                    (magit-tag-arguments)))
  (magit-run-git-with-editor "tag" args name rev))
```

3.1 Defining prefix commands

Prefix commands and their infix arguments are defined using the macro `magit-define-popup`. The key bindings and descriptions of suffix commands are also defined using that macro, but the actual interactive commands have to be defined separately using plain `defun`.

`magit-define-popup` *name doc [group [mode [option]]] :keyword value. . .* [Macro]

This macro defines a popup named NAME. The NAME should begin with the package prefix and by convention end with `-popup`, it is used as the name of the command which shows the popup and for an internal variable (whose value is used to store information about the popup and should not be accessed directly). DOC is the docstring of the popup command.

This macro also defines an option and a function both named `SHORTNAME-arguments`, where `SHORTNAME` is `NAME` with the trailing `-popup` removed. The name of this option and this function can be overwritten using the optional argument `OPTION`, but that is rarely advisable. As a special case if `OPTION` is specified but `nil`, then this option and this function are not defined at all, which is useful for popups that are used as simple dispatchers that offer no arguments.

The option `SHORTNAME-arguments` holds the value for the popup arguments. It can be customized from within the popup or using the Custom interface. It can also have a buffer local value in any non-popup buffer. The local value for the buffer from which the popup command was invoked, can be set from within the popup buffer.

The function `SHORTNAME-arguments` returns the currently effective value of the variable by the same name. See below for more information.

The optional argument `GROUP` specifies the Custom group in which the option is placed. If omitted then the option is placed in some group the same way it is done when directly using `defcustom` and omitting the group.

The optional argument `MODE` specifies the mode used by the popup buffer. If it is omitted or `nil` then `magit-popup-mode` is used.

The remaining arguments should have the form `[KEYWORD VALUE] . . .`

The following keywords are meaningful (and by convention are usually specified in that order):

- `:actions`

The actions which can be invoked from the popup. `VALUE` is a list whose members have the form `(KEY DESC COMMAND)`, see `magit-define-popup-action` for details.

How the actions are split into rows and columns currently depends on the available space and `:max-action-columns`.

WARNING: This will likely be change to use a more explicit format `((KEY DESC COMMAND). . .). . .)` before the release.

Actions are regular Emacs commands, which usually have an `interactive` form setup to consume the values of the popup `:switches` and `:options` when invoked from the corresponding popup, else when invoked as the default action or directly without using the popup, the default value of the variable `SHORTNAME-arguments`. This is usually done by calling the function `SHORTNAME-arguments`.

Members of `VALUE` may also be strings, assuming the first member is also a string. Instead of just one action section with the heading `\ "Actions\ "`, multiple sections are then inserted into the popup buffer, using these strings as headings.

Members of `VALUE` may also be `nil`. This should only be used together with `:max-action-columns` and allows having gaps in the action grid, which can help arranging actions sensibly.

- `:default-action`

The default action of the popup which is used directly instead of displaying the popup buffer, when the popup is invoked with a prefix argument. Also see `magit-popup-use-prefix-argument` and `:use-prefix`, which can be used to inverse the meaning of the prefix argument.

- **:use-prefix**
Controls when to display the popup buffer and when to invoke the default action (if any) directly. This overrides the global default set using `magit-popup-use-prefix-argument`. The value, if specified, should be one of `default` or `prefix`.
- **:switches**
The popup arguments which can be toggled on and off. `VALUE` is a list whose members have the form `(KEY DESC SWITCH)`, see `magit-define-popup-switch` for details.
- **:options**
The popup arguments which take a value, as in `"-opt~OPTVAL"`. `VALUE` is a list whose members have the form `(KEY DESC OPTION READER)`, see `magit-define-popup-option` for details.
- **:default-arguments**
The default arguments, a list of switches (which are then enabled by default) and options with their default values, as in `"-OPT~OPTVAL\"`.
- **:man-page**
The name of the manpage to be displayed when the user requests help for an argument.

When `MODE` is `magit-popup-sequence-mode`, then the following keywords are also meaningful:

- **:sequence-predicate**
When this function returns non-nil, then the popup uses `:sequence-actions` instead of `:actions`.
- **:sequence-actions**
The actions which can be invoked from the popup, when `:sequence-predicate` returns non-nil.

3.2 Defining suffix commands

Commands intended to be invoked from a particular popup should determine the currently effective arguments by calling the function `SHORTNAME-arguments` inside their `interactive` form. This function is created by the `magit-define-popup` macro. For a popup named `prefix-foo-popup` the name of this function is `prefix-foo-arguments`.

When the command was invoked as an action in the respective popup, then this function returns the arguments that were set in the popup. Otherwise when the command was invoked as the default of the popup (by calling the popup command with a prefix argument), or without using the popup command at all, then this function returns the buffer-local or global value of the variable `SHORTNAME-arguments`.

Internally arguments are handled as a list of strings. This might not be appropriate for the intended use inside commands, or it might be necessary to manipulate that list somehow, i.e. to split `"-ARG=VAL"` into `"-ARG""VAL"`. This should be done by advising or redefining the function `SHORTNAME-arguments`.

Internally `SHORTNAME-arguments` used following variables and function. Except when redefining the former, you should not use these directly.

magit-current-popup [Variable]

The popup from which this editing command was invoked.

magit-current-popup-args [Variable]

The value of the popup arguments for this editing command.

If the current command was invoked from a popup, then this is a list of strings of all the set switches and options. This includes arguments which are set by default not only those explicitly set during this invocation.

When the value is nil, then that can be because no argument is set, or because the current command wasn't invoked from a popup at all.

magit-current-popup-args &rest args [Function]

This function returns the value of the popup arguments for this editing command.

The value is the same as that of the variable by the same name, except that **FILTER** is applied. **FILTER** is a list of regexps; only arguments that match one of them are returned. The first element of **FILTER** may also be **:not** in which case only arguments that don't match any of the regexps are returned, or **:only** which doesn't change the behavior.