The Tig Manual

Jonas Fonseca

This is the manual for Tig, the ncurses-based text-mode interface for git. Tig allows you to browse changes in a Git repository and can additionally act as a pager for output of various Git commands. When used as a pager, it will display input from stdin and colorize it.

When browsing repositories, Tig uses the underlying Git commands to present the user with various views, such as summarized commit log and showing the commit with the log message, diffstat, and the diff.

1. Calling Conventions

1.1. Pager Mode

If stdin is a pipe, any log or diff options will be ignored and the pager view will be opened loading data from stdin. The pager mode can be used for colorizing output from various Git commands.

Example on how to colorize the output of git-show(1):

```
$ git show | tig
```

1.2. Git Command Options

All Git command options specified on the command line will be passed to the given command and all will be shell quoted before they are passed to the shell.

> **Note:** If you specify options for the main view, you should not use the `--pretty` option as this option will be set automatically to the format expected by the main view.

Example on how to view a commit and show both author and committer information:

```
$ tig show --pretty=fuller
```

See the section on specifying revisions for an introduction to revision options supported by the Git commands. For details on specific Git command options, refer to the man page of the command in question.

2. The Viewer

The display consists of a status window on the last line of the screen and one or more views. The default is to only show one view at a time but it is possible to split both the main and log view to also show the commit diff.

If you are in the log view and press *Enter* when the current line is a commit line, such as:

```
commit 4d55caff4cc89335192f3e566004b4ceef572521
```

You will split the view so that the log view is displayed in the top window and the diff view in the bottom window. You can switch between the two views by pressing *Tab*. To maximize the log view again, simply press *l*.

2.1. Views

Various *views* of a repository are presented. Each view is based on output from an external command, most often *git log*, *git diff*, or *git show*.

The main view

: Is the default view, and it shows a one line summary of each commit in the chosen list of revisions. The summary includes author date, author, and the first line of the log message. Additionally, any repository references, such as tags, will be shown.

The log view

: Presents a more rich view of the revision log showing the whole log message and the diffstat.

The reflog view

: Presents a view of the reflog allowing to navigate the repo history.

The diff view

: Shows either the diff of the current working tree, that is, what has changed since the last commit, or the commit diff complete with log message, diffstat and diff.

The tree view

: Lists directory trees associated with the current revision allowing subdirectories to be descended or ascended and file blobs to be viewed.

The blob view

: Displays the file content or "blob" of data associated with a file name.

The blame view

: Displays the file content annotated or blamed by commits.

The refs view

: Displays the branches, remotes and tags in the repository.

The status view

: Displays status of files in the working tree and allows changes to be staged/unstaged as well as adding of untracked files.

The stage view

: Displays diff changes for staged or unstaged files being tracked or file content of untracked files.

The stash view

: Displays the list of stashes in the repository.

The grep view

: Displays a list of files and all the lines that matches a search pattern.

The pager view

: Is used for displaying both input from stdin and output from Git commands entered in the internal prompt.

The help view

: Displays a quick reference of key bindings.

2.2. Browsing State and User-defined Commands

The viewer keeps track of both what head and commit ID you are currently viewing. The commit ID will follow the cursor line and change every time you highlight a different commit. Whenever you reopen the diff view it will be reloaded, if the commit ID changed. The head ID is used when opening the main and log view to indicate from what revision to show history.

Some of the commands used or provided by Tig can be configured. This goes for some of the environment variables as well as the external commands. These user-defined commands can use arguments that refer to the current browsing state by using one of the following variables.

Table 1. Browsing state variables

Example user-defined commands:

- Allow to amend the last commit:

  ```
  bind generic + !git commit --amend
  ```

- Copy commit ID to clipboard:

  ```
  bind generic 9 @sh -c "echo -n %(commit) | xclip -selection c"
  ```

- Add/edit notes for the current commit used during a review:

  ```
  bind generic T !git notes edit %(commit)
  ```

- Enter Git's interactive add for fine-grained staging of file content:

  ```
  bind generic I !git add -i %(file)
  ```

- Rebase current branch on top of the selected branch:

  ```
  bind refs 3 !git rebase -i %(branch)
  ```

2.3. Title Windows

Each view has a title window which shows the name of the view, current commit ID if available, and where the view is positioned:

```
[main] c622eefaa485995320bc743431bae0d497b1d875 - commit 1 of 61 (1%)
```

By default, the title of the current view is highlighted using bold font. For long loading views (taking over 3 seconds) the time since loading started will be appended:

```
[main] 77d9e40fbcea3238015aea403e06f61542df9a31 - commit 1 of 779 (0%) 5s
```

3. Environment Variables

Several options related to the interface with Git can be configured via environment options.

3.1. Configuration Files

Upon startup, Tig first reads the system wide configuration file (`{sysconfdir}/tigrc` by default) and then proceeds to read the user's configuration file (`~/.tigrc` or `$XDG_CONFIG_HOME/tig/config` by default). The paths to either of these files can be overridden through the following environment variables:

TIGRC_USER

Path of the user configuration file.

TIGRC_SYSTEM

Path of the system wide configuration file.

3.2. History Files

If compiled with readline support, Tig writes a persistent command and search history to `~/.tig_history` or `$XDG_DATA_HOME/tig/history`.

3.3. Repository References

Commits that are referenced by tags and branch heads will be marked by the reference name surrounded by *[* and *]*:

```
2006-03-26 19:42 Petr Baudis         | [cogito-0.17.1] Cogito 0.17.1
```

If you want to limit what branches are shown, say only show branches named `master` or those which start with the `feature/` prefix, you can do it by setting the following variable:

```
$ TIG_LS_REMOTE="git ls-remote . master feature/*" tig
```

Or set the variable permanently in your environment.

TIG_LS_REMOTE

Command for retrieving all repository references. The command should output data in the same format as git-ls-remote(1). Defaults to:

```
git ls-remote .
```

3.4. Diff options

It is possible to alter how diffs are shown by the diff view. If for example you prefer to have commit and author dates shown as relative dates, use:

```
$ TIG_DIFF_OPTS="--relative-date" tig
```

Or set the variable permanently in your environment.

4. Default Keybindings

Below the default key bindings are shown.

4.1. View Switching

4.2. View Manipulation

4.3. View Specific Actions

4.4. Cursor Navigation

4.5. Scrolling

4.6. Searching

The format for patterns is either POSIX.2 “extended” REs or PCRE / PCRE2 if Tig was compiled with PCRE / PCRE2 support (check with `tig -v`). See the manpage of re_format(7) or pcrepattern(3) / pcre2pattern(3).

Case sensitivity can be controlled with variable ignore-case.

4.7. Misc

4.8. Prompt

4.9. External Commands

For more custom needs, external commands provide a way to easily execute a script or program. They are bound to keys and use information from the current browsing state, such as the current commit ID. Tig comes with the following built-in external commands:

5. Revision Specification

This section describes various ways to specify what revisions to display or otherwise limit the view to. Tig does not itself parse the described revision options so refer to the relevant Git man pages for further information. Relevant man pages besides git-log(1) are git-diff(1) and git-rev-list(1).

You can tune the interaction with Git by making use of the options explained in this section. For example, by configuring the environment variable described in the section on diff options.

5.1. Limit by Path Name

If you are interested only in those revisions that made changes to a specific file (or even several files) list the files like this:

```
$ tig Makefile README
```

To avoid ambiguity with Tig's subcommands or repository references such as tag names, be sure to separate file names from other Git options using "`--`". So if you have a file named *status* it will clash with the *status* subcommand, and thus you will have to use:

```
$ tig -- status
```

5.2. Limit by Date or Number

To speed up interaction with Git, you can limit the amount of commits to show both for the log and main view. Either limit by date using e.g. `--since=1.month` or limit by the number of commits using `-n400`.

If you are only interested in changes that happened between two dates you can use:

```
$ tig --after="May 5th" --before="2006-05-16 15:44"
```

> **Note:** If you want to avoid having to quote dates containing spaces you can use "." instead, e.g. `--after=May.5th`.

5.3. Limiting by Commit Ranges

Alternatively, commits can be limited to a specific range, such as "all commits between *tag-1.0* and *tag-2.0*". For example:

```
$ tig tag-1.0..tag-2.0
```

This way of commit limiting makes it trivial to only browse the commits which haven't been pushed to a remote branch. Assuming *origin* is your upstream remote branch, using:

```
$ tig origin..HEAD
```

will list what will be pushed to the remote branch. Optionally, the ending *HEAD* can be left out since it is implied.

5.4. Limiting by Reachability

Git interprets the range specifier "tag-1.0..tag-2.0" as "all commits reachable from *tag-2.0* but not from *tag-1.0*". Where reachability refers to what commits are ancestors (or part of the history) of the branch or tagged revision in question.

If you prefer to specify which commit to preview in this way use the following:

```
$ tig tag-2.0 ^tag-1.0
```

You can think of ^ as a negation operator. Using this alternate syntax, it is possible to further prune commits by specifying multiple branch cut offs.

5.5. Combining Revisions Specification

Revision options can to some degree be combined, which makes it possible to say "show at most 20 commits from within the last month that changed files under the Documentation/ directory."

```
$ tig --since=1.month -n20 -- Documentation/
```

5.6. Examining All Repository References

In some cases, it can be useful to query changes across all references in a repository. An example is to ask "did any line of development in this repository change a particular file within the last week". This can be accomplished using:

```
$ tig --all --since=1.week -- Makefile
```

6. More Information

Please visit Tig's home page (https://jonas.github.io/tig) or main Git repository (https://github.com/jonas/tig) for information about new releases and how to report bugs and feature requests.

7. Copyright

8. See Also

Manpages:

- tig(1)
- tigrc(5)