# Git config

In this document, we'll take an in-depth look at the `GIT CONFIG` command. The `GIT CONFIG` command is a convenience function that is used to set Git configuration values on a global or local project level. These configuration levels correspond to `.gitconfig` text files. Executing git config will modify a configuration text file. We'll be covering common configuration settings like email, username, and editor. Becoming familiar with git config and the various Git configuration settings will help you create a powerful, customized Git workflow.

## Usage

The most basic use of git config is to invoke it with a configuration name, which will display the set value at that name. Configuration names are dot delimited strings composed of a 'section' and a 'key' basic on their hierarchy. For example - user.email

```
git config user.email
```

In this example, email is a child property of the user configuration block. This will return the configured email address if any, that Git will associate with locally created commits.

## git config levels and files

Before we further discuss git config usage, let's take a moment to cover configuration levels. The git config command can accept arguments to specify which configuration level to operate on. The following configuration levels are available:

- **--local**

By default, git config will write to a local level if no configuration option is passed. Local-level configuration is applied to the context repository git config gets invoked in. Local configuration values are stored in a file that can be found in the repo's .git director: `.git/config`

- **--global**

Global level configuration is user-specific, meaning it is applied to an operating system user. Global configuration values are stored in a file that is located in a user's home directory. `~/.gitconfig` on Unix systems and `C:\Users\\.gitconfig` on windows.

- **--system**

System-level configuration is applied across an entire machine. This covers all users on an operating system and all repos. The system-level configuration file lives in a gitconfig file off the system root path `${prefix}/etc/gitconfig` on Unix systems. On windows this file can be found at `C: \Documents` and `Settings\All Users\Application Data\Git\config` on Windows XP, and in `C: \ProgramData\Git\config` on Windows Vista and newer.

Thus the order of priority for configuration levels is - local, global, and system. This means when looking for a configuration value, Git will start at the local level and bubble up to the system level.

## Writing a value

Expanding on what we already know about git config, let's look at an example in which we write a value:

```
git config --global user.email "your_email@example.com"
```

This example writes the value your_email@example.com to the configuration name user.email. It uses the --global flag so this value is set for the current operating system user.

git config editor - core.editor

Many Git commands will launch a text editor to prompt for further input. One of the most common use cases for git config is configuring which editor Git should use. Listed below is a table of the popular editor and matching git config commands:

| Editor | config command |
|---|---|
| Atom | `~ git config --global core.editor "atom --wait"~` |
| emacs | `~ git config --global core.editor "emacs"~` |
| nano | `~ git config --global core.editor "nano -w"~` |
| vim | `~ git config --global core.editor "vim"~` |
| Sublime Text (Mac) | `~ git config --global core.editor "subl -n -w"~` |

| | |
|---|---|
| Sublime Text (Win, 32-bit install) | `~ git config --global core.editor "'c:/program files (x86)/sublime text 3 /sublimetext.exe' -w"~` |
| Sublime Text (Win, 64-bit install) | `~ git config --global core.editor "'c:/program files/sublime text 3 /sublimetext.exe' -w"~` |
| Textmate | `~ git config --global core.editor "mate -w"~` |

## Merge tools

In the event of a merge conflict, Git will launch a “merge tool”. By default, Git uses an internal implementation of the common Unix diff program. The internal Git diff is a minimal merge conflict viewer. There are many external third-party merge conflict resolutions that can be used instead.

```
git config --global merge.tool kidff3
```

## Colored outputs

Git supports colored terminal output which helps with rapidly reading Git output. You can customize your Git output to use a personalized color theme. The git config command is used to set these color values.

#### color.ui

This is the master variable for Git colors. Setting it to false will disable all Git-colored terminal output.

```
git config --global color.ui false
```

By default, color.ui is set to auto which will apply colors to the immediate terminal output stream. The auto setting will omit color code output if the output stream is redirected to a file or piped to another process.

You can set the color.ui value to always which will also apply color code output when redirecting the output stream to files or pipes. This can unintentionally cause problems since the receiving pipe may not be expecting color-coded input.

## Git color values

In addition to color.ui, there are many other granular color settings. Like color.ui, these color settings can all be set to false, auto, or always. These color settings can also have a specific color value set. Some examples of supported color values are:

- normal
- **black**
- **red**
- **green**
- **yellow**
- **blue**
- **magenta**
- **cyan**
- white

Colors may also be specified as hexadecimal color codes like `##c80a50`, or ANSI 256 color values if your terminal supports it.

## Git color configuration settings

1. `color.branch`

   - Configures the output color of the Git branch output command

2. `color.branch. <slot>`

   1. current: the current branch
   2. local: a local branch
   3. remote: a remote branch ref in refs/remotes
   4. upstream: an upstream tracking branch
   5. plain: any other ref

3. `color.diff`

- Applies colors to git diff, git long, and git show output

4. `color.diff.<slot>`

- Configuring a <slot> value under color.diff tells git which part of the patch to use a specific color on.

1. context: The context text of the diff. Git context is the lines of text content shown in a diff or patch that highlights changes.
2. plain: a synonym for context
3. meta: applies color to the meta-information of the diff
4. frag: applies a color to the removed lines in the diff
5. new: colors the added lines of the diff
6. commit: colors commit headers within the diff
7. whitespace: sets a color for any whitespace errors in a diff

5. `color.decorate.<slot>`

- Customize the color for git log --decorate output. The supported <slot> values are- branch, remoteBranch, tag, stash, or HEAD. They are respectively applicable to local branches, remote-tracking branches, tags, stashed changes and HEAD.

6. `color.grep`

- Applies color to the output of git grep.

7. `color.grep.<slot>`

- Also applicable to git grep. The <slot> variable specifies which part of the grep output to apply color.

1. context: non-matching text in context lines

2. filename: filename prefix

3. function: function name lines

4. linenumber: line number prefix

5. match: matching text

6. matchContext: matching text in context lines

7. matchedSelected: matching text in selected lines

8. selected: non-matching text in selected lines

9. separator: separators between fields on a line (:, -, and =) and between hunks (--)

8. `color.interactive`

- This variable applies color for interactive prompts and displays. Examples are git and --interactive and git clean --interactive.

9. `color.interactive.<slot>`

- The <slot> variable can be specified to target more specific “interactive output”. The available <slot> values are: prompt, header, help, error; and each act on the corresponding interactive output.

10. `color.pager`

- Enables or disables the color output when the pager is in use

11. `color.showBranch`

- Enables or disables color output for the git show branch command

12. `color.status`

- A boolean value that enables or disables color output for Git status

13. `color.status.<slot>`

Used to specify custom color for specified git status elements. <slot> supports the following values:

- header

- Targets the header text of the status area

- added or updated

- Both target files which are added but not committed

- changed

- Targets files that are modified but not added to the git index

- untracked

- Targets files that are not tracked by Git

- branch

- Applies the color of the current branch

- nobranch

- The color of the “no branch” warning is shown in

- unmerged

- Colors files that have unmerged changes

## Aliases

You may be familiar with the concept of aliases from your operating system command-line; if not, they’re custom shortcuts that define which command will expand to longer or combined commands. Aliases save you the time and energy cost of typing frequently used commands. Git provides its own alias system. A common use case for Git aliases is shortening the commit command. Git aliases are stored in Git configuration files. This means you can use the `GIT CONFIG` command to configure aliases.

```
git config --global alias.ci commit
```

This example creates a ci alias for the git commit command. You can then invoke git commit by executing git ci. Aliases can also reference other aliases to create powerful combos.

```
git config --global alias.amend ci --amend
```

This example creates an alias amend which composes the ci alias into a new alias that uses --amend flag.

## Formatting & whitespace

Git has several “whitespace” features that can be configured to highlight whitespace issues when using git diff. The whitespace issues will be highlighted using the configured color.diff.whitespace.

The following features are enabled by default:

- blank-et-eol highlights orphan whitespaces at the line endings
- space-before-tab highlights a space character that appears before a tab character when indenting a line
- bland-at-eof highlights bland lines inserted at the end of a file

The following features are disabled by default

- indent-with-non-tab highlights a line that is indented with spaces instead of tabs
- tab-in-indent highlights an initial tab indent as an error
- trailing-space is shorthand for both-et-eol and bland-at-eof
- cr-at-eol highlights a carriage return at the line endings
- tabwidth= defines how many character positions a tab occupies. The default value is 8. Allowed values are 1-63

## How to do a git config global edit?

The global git config is simply a text file, so it can be edited with whatever text editor you choose. Open, edit global git config, save and close, and the changes will take effect the next time you issue a git command. It’s that easy.

From within the BASH shell or terminal window, you can invoke the default Git editor through the following command:

```
git config --global --edit
```

## List and show global git config

To see all of the properties configured globally in Git, you can use the --list switch on the git config command. Adding the -show-origin switch will also tell you the global .gitconfig file's location.

```
global@git:~/$ git config --global --list --show-origin
file:/home/gme/.gitconfig user.email=cameronmcnz@example.com
file:/home/gme/.gitconfig user.name=cameronmcnz
file:/home/gme/.gitconfig core.editor=vim
file:/home/gme/.gitconfig http.sslverify=false
file:/home/gme/.gitconfig credential.helper=store
file:/home/gme/.gitconfig http.proxy=193.168.0.11
file:/home/gme/.gitconfig http.postbuffer=193.168.0.12
file:/home/gme/.gitconfig http.sslcainfo=193.168.0.10
```

## Remove global git config settings

To delete git config settings simply use the unset command:

```
git config --global --unset core.editor
```

Occasionally, a property gets set twice and the -unset switch fails. In this case, simply use the global git confg's -unset-all switch.

```
git config --global --unset-all core.editor
```

Try these commands to remove the user name and emails

```
git config --global --unset user.name
git config --global --unset user.email
```

Try these commands to remove all user's usernames and emails

```
git config --global --unset-all user.name
git config --global --unset-all user.email
```

The global git config is an important file for customizing your version control experience. Knowing how to show Git config settings is important, as is being able to edit, update and remove settings. Knowing how will certainly make your experience with the global Git config tool more pleasurable.

Keep Thinking!