
This book is available in English.

Full translation available in

azərbaycan dili,
български език,
Deutsch,
Español,
Français,
Ελληνικά,
日本語,
한국어,
Nederlands,

Русский,
Slovenščina,
Tagalog,
Українська
简体中文,

Partial translations available in

Čeština,
Македонски,
Polski,
Српски,
Ўзбекча,
繁體中文,

Translations started for

Беларуская,
فارسی,
Indonesian,
Italiano,
Bahasa Melayu,
Português (Brasil),
Português (Portugal),
Svenska,
Türkçe.

The source of this book is hosted on GitHub.
Patches, suggestions and comments are welcome.
Chapters ▾

2nd Edition

1.6 Getting Started - First-Time Git Setup

First-Time Git Setup

Now that you have Git on your system, you'll want to do a few things to customize your Git environment. You should have to do these things only once on any given computer; they'll stick around between upgrades. You can also change them at any time by running through the commands again.

Git comes with a tool called `git config` that lets you get and set configuration variables that control all aspects of how Git looks and operates. These variables can be stored in three different places:

1. `[path]/etc/gitconfig` file: Contains values applied to every user on the system and all their repositories. If you pass the option `--system` to `git config`, it reads and writes from this file specifically. Because this is a system configuration file, you would need administrative or superuser privilege to make changes to it.

2. `~/.gitconfig` or `~/.config/git/config` file: Values specific personally to you, the user. You can make Git read and write to this file specifically by passing the `--global` option, and this affects *all* of the repositories you work with on your system.

3. `config` file in the Git directory (that is, `.git/config`) of whatever repository you're currently using: Specific to that single repository. You can force Git to read from and write to this file with the `--local` option, but that is in fact the default. Unsurprisingly, you need to be located somewhere in a Git repository for this option to work properly.

Each level overrides values in the previous level, so values in `.git/config` trump those in `[path]/etc/gitconfig`.

On Windows systems, Git looks for the `.gitconfig` file in the `$HOME` directory (`C:\Users\$USER` for most people). It also still looks for `[path]/etc/gitconfig`, although it's relative to the MSys root, which is wherever you decide to install Git on your Windows system when you run the installer. If you are using version 2.x or later of Git for Windows, there is also a system-level config file 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. This config file can only be changed by `git config -f <file>` as an admin.

You can view all of your settings and where they are coming from using:

```
$ git config --list --show-origin
```

Your Identity

The first thing you should do when you install Git is to set your user name and email address. This is important because every Git commit uses this information, and it's immutably baked into the commits you start creating:

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

Again, you need to do this only once if you pass the `--global` option, because then Git will always use that information for anything you do on that system. If you want to override this with a different name or email address for specific projects, you can run the command without the `--global` option when you're in that project.

Many of the GUI tools will help you do this when you first run them.

Your Editor

Now that your identity is set up, you can configure the default text editor that will be used when Git needs you to type in a message. If not configured, Git uses your system's default editor.

If you want to use a different text editor, such as Emacs, you can do the following:

```
$ git config --global core.editor emacs
```

On a Windows system, if you want to use a different text editor, you must specify the full path to its executable file. This can be different depending on how your editor is packaged.

In the case of Notepad++, a popular programming editor, you are likely to want to use the 32-bit version, since at the time of writing the 64-bit version doesn't support all plug-ins. If you are on a 32-bit Windows system, or you have a 64-bit editor on a 64-bit system, you'll type something like this:

```
$ git config --global core.editor "'C:/Program Files/Notepad++/notepad++.exe' -multiInst -notabbar -nosession -noPlugin"
```

Note: Vim, Emacs and Notepad++ are popular text editors often used by developers on Unix-based systems like Linux and macOS or a Windows system. If you are using another editor, or a 32-bit version, please find specific instructions for how to set up your favorite editor with Git in git config core.editor commands.

Warning: You may find, if you don't setup your editor like this, you get into a really confusing state when Git attempts to launch it. An example on a Windows system may include a prematurely terminated Git operation during a Git initiated edit.

Your default branch name

By default Git will create a branch called *master* when you create a new repository with `git init`. From Git version 2.28 onwards, you can set a different name for the initial branch.

To set *main* as the default branch name do:

```
$ git config --global init.defaultBranch main
```

Checking Your Settings

If you want to check your configuration settings, you can use the `git config --list` command to list all the settings Git can find at that point:

```
$ git config --list
user.name=John Doe
user.email=johndoe@example.com
color.status=auto
color.branch=auto
color.interactive=auto
color.diff=auto
...
```

You may see keys more than once, because Git reads the same key from different files (`[path]/etc/gitconfig` and `~/.gitconfig`, for example). In this case, Git uses the last value for each unique key it sees.

You can also check what Git thinks a specific key's value is by typing `git config <key>`:

```
$ git config user.name
John Doe
```

Note

Since Git might read the same configuration variable value from more than one file, it’s possible that you have an unexpected value for one of these values and you don’t know why. In cases like that, you can query Git as to the *origin* for that value, and it will tell you which configuration file had the final say in setting that value:

```
$ git config --show-origin rerere.autoUpdate
file:/home/johndoe/.gitconfig   false
```

prev | next
About this site
Patches, suggestions, and comments are welcome.
Git is a member of Software Freedom Conservancy