git --distributed-even-if-your-workflow-isnt

Search entire site...

- About
- Documentation
  - Reference
  - Book
  - Videos
  - External Links
- Blog
- Downloads
  - GUI Clients
  - Logos
- Community

This book is available in English.

Full translation available in Deutsch, 简体中文, 正體中文, Français, 日本語, Nederlands, Русский, 한국어, Português (Brasil) and Čeština.

Partial translations available in Arabic, Español, Indonesian, Italiano, Suomi, Македонски, Ελληνικά, Polski and Türkçe.

Translations started for Azərbaycan dili, Беларуская, Català, Esperanto, Español (Nicaragua), فارسی, हिन्दी, Magyar, Norwegian Bokmål, Română, Српски, ภาษาไทย, Tiếng Viêt, Українська and Ўзбекча.

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

Related Material

- **git-remote** in Reference
- **git-pull** in Reference
- **git-push** in Reference
- **git-fetch** in Reference
- **git-clone** in Reference

Chapters ▾

1. **1. Getting Started**
   1. 1.1 About Version Control
   2. 1.2 A Short History of Git
   3. 1.3 Git Basics
   4. 1.4 The Command Line
   5. 1.5 Installing Git
   6. 1.6 First-Time Git Setup
   7. 1.7 Getting Help
   8. 1.8 Summary

2. **2. Git Basics**

1. 2.1 Getting a Git Repository
2. 2.2 Recording Changes to the Repository
3. 2.3 Viewing the Commit History
4. 2.4 Undoing Things
5. 2.5 Working with Remotes
6. 2.6 Tagging
7. 2.7 Git Aliases
8. 2.8 Summary

3. **3. Git Branching**

1. 3.1 Branches in a Nutshell
2. 3.2 Basic Branching and Merging
3. 3.3 Branch Management
4. 3.4 Branching Workflows
5. 3.5 Remote Branches
6. 3.6 Rebasing
7. 3.7 Summary

4. **4. Git on the Server**

1. 4.1 The Protocols
2. 4.2 Getting Git on a Server
3. 4.3 Generating Your SSH Public Key
4. 4.4 Setting Up the Server
5. 4.5 Git Daemon
6. 4.6 Smart HTTP
7. 4.7 GitWeb
8. 4.8 GitLab
9. 4.9 Third Party Hosted Options
10. 4.10 Summary

5. **5. Distributed Git**

1. 5.1 Distributed Workflows
2. 5.2 Contributing to a Project
3. 5.3 Maintaining a Project
4. 5.4 Summary

1. **6. GitHub**

1. 6.1 Account Setup and Configuration
2. 6.2 Contributing to a Project
3. 6.3 Maintaining a Project
4. 6.4 Managing an organization
5. 6.5 Scripting GitHub
6. 6.6 Summary

2. **7. Git Tools**

    1. 7.1 Revision Selection
    2. 7.2 Interactive Staging
    3. 7.3 Stashing and Cleaning
    4. 7.4 Signing Your Work
    5. 7.5 Searching
    6. 7.6 Rewriting History
    7. 7.7 Reset Demystified
    8. 7.8 Advanced Merging
    9. 7.9 Rerere
    10. 7.10 Debugging with Git
    11. 7.11 Submodules
    12. 7.12 Bundling
    13. 7.13 Replace
    14. 7.14 Credential Storage
    15. 7.15 Summary

3. **8. Customizing Git**

    1. 8.1 Git Configuration
    2. 8.2 Git Attributes
    3. 8.3 Git Hooks
    4. 8.4 An Example Git-Enforced Policy
    5. 8.5 Summary

4. **9. Git and Other Systems**

    1. 9.1 Git as a Client
    2. 9.2 Migrating to Git
    3. 9.3 Summary

5. **10. Git Internals**

    1. 10.1 Plumbing and Porcelain
    2. 10.2 Git Objects
    3. 10.3 Git References
    4. 10.4 Packfiles
    5. 10.5 The Refspec
    6. 10.6 Transfer Protocols
    7. 10.7 Maintenance and Data Recovery
    8. 10.8 Environment Variables
    9. 10.9 Summary

1. **A1. Appendix A: Git in Other Environments**

    1. A1.1 Graphical Interfaces
    2. A1.2 Git in Visual Studio
    3. A1.3 Git in Eclipse
    4. A1.4 Git in Bash

5. A1.5 Git in Zsh
6. A1.6 Git in Powershell
7. A1.7 Summary

2. **A2. Appendix B: Embedding Git in your Applications**

    1. A2.1 Command-line Git
    2. A2.2 Libgit2
    3. A2.3 JGit

3. **A3. Appendix C: Git Commands**

    1. A3.1 Setup and Config
    2. A3.2 Getting and Creating Projects
    3. A3.3 Basic Snapshotting
    4. A3.4 Branching and Merging
    5. A3.5 Sharing and Updating Projects
    6. A3.6 Inspection and Comparison
    7. A3.7 Debugging
    8. A3.8 Patching
    9. A3.9 Email
    10. A3.10 External Systems
    11. A3.11 Administration
    12. A3.12 Plumbing Commands

2nd Edition

2.5 Git Basics - Working with Remotes

Working with Remotes

To be able to collaborate on any Git project, you need to know how to manage your remote repositories. Remote repositories are versions of your project that are hosted on the Internet or network somewhere. You can have several of them, each of which generally is either read-only or read/write for you. Collaborating with others involves managing these remote repositories and pushing and pulling data to and from them when you need to share work. Managing remote repositories includes knowing how to add remote repositories, remove remotes that are no longer valid, manage various remote branches and define them as being tracked or not, and more. In this section, we'll cover some of these remote-management skills.

Showing Your Remotes

To see which remote servers you have configured, you can run the `git remote` command. It lists the shortnames of each remote handle you've specified. If you've cloned your repository, you should at least see `origin` – that is the default name Git gives to the server you cloned from:

```
$ git clone https://github.com/schacon/ticgit
Cloning into 'ticgit'...
remote: Reusing existing pack: 1857, done.
remote: Total 1857 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (1857/1857), 374.35 KiB | 268.00 KiB/s, done.
Resolving deltas: 100% (772/772), done.
Checking connectivity... done.
```

```
$ cd ticgit
$ git remote
origin
```

You can also specify `-v`, which shows you the URLs that Git has stored for the shortname to be used when reading and writing to that remote:

```
$ git remote -v
origin  https://github.com/schacon/ticgit (fetch)
origin  https://github.com/schacon/ticgit (push)
```

If you have more than one remote, the command lists them all. For example, a repository with multiple remotes for working with several collaborators might look something like this.

```
$ cd grit
$ git remote -v
bakkdoor  https://github.com/bakkdoor/grit (fetch)
bakkdoor  https://github.com/bakkdoor/grit (push)
cho45     https://github.com/cho45/grit (fetch)
cho45     https://github.com/cho45/grit (push)
defunkt   https://github.com/defunkt/grit (fetch)
defunkt   https://github.com/defunkt/grit (push)
koke      git://github.com/koke/grit.git (fetch)
koke      git://github.com/koke/grit.git (push)
origin    git@github.com:mojombo/grit.git (fetch)
origin    git@github.com:mojombo/grit.git (push)
```

This means we can pull contributions from any of these users pretty easily. We may additionally have permission to push to one or more of these, though we can't tell that here.

Notice that these remotes use a variety of protocols; we'll cover more about this in Git on the Server.

Adding Remote Repositories

We've mentioned and given some demonstrations of how the `git clone` command implicitly adds the `origin` remote for you. Here's how to add a new remote explicitly. To add a new remote Git repository as a shortname you can reference easily, run `git remote add <shortname> <url>`:

```
$ git remote
origin
$ git remote add pb https://github.com/paulboone/ticgit
$ git remote -v
origin  https://github.com/schacon/ticgit (fetch)
origin  https://github.com/schacon/ticgit (push)
pb      https://github.com/paulboone/ticgit (fetch)
pb      https://github.com/paulboone/ticgit (push)
```

Now you can use the string `pb` on the command line in lieu of the whole URL. For example, if you want to fetch all the information that Paul has but that you don't yet have in your repository, you can run `git fetch pb`:

```
$ git fetch pb
remote: Counting objects: 43, done.
remote: Compressing objects: 100% (36/36), done.
remote: Total 43 (delta 10), reused 31 (delta 5)
Unpacking objects: 100% (43/43), done.
From https://github.com/paulboone/ticgit
 * [new branch]      master     -> pb/master
 * [new branch]      ticgit     -> pb/ticgit
```

Paul's master branch is now accessible locally as `pb/master` – you can merge it into one of your branches, or you can check out a local branch at that point if you want to inspect it. (We'll go over what branches are and how to use them in much more detail in Git Branching.)

Fetching and Pulling from Your Remotes

As you just saw, to get data from your remote projects, you can run:

```
$ git fetch <remote>
```

The command goes out to that remote project and pulls down all the data from that remote project that you don't have yet. After you do this, you should have references to all the branches from that remote, which you can merge in or inspect at any time.

If you clone a repository, the command automatically adds that remote repository under the name "origin". So, `git fetch origin` fetches any new work that has been pushed to that server since you cloned (or last fetched from) it. It's important to note that the `git fetch` command only downloads the data to your local repository – it doesn't automatically merge it with any of your work or modify what you're currently working on. You have to merge it manually into your work when you're ready.

If your current branch is set up to track a remote branch (see the next section and Git Branching for more information), you can use the `git pull` command to automatically fetch and then merge that remote branch into your current branch. This may be an easier or more comfortable workflow for you; and by default, the `git clone` command automatically sets up your local master branch to track the remote master branch (or whatever the default branch is called) on the server you cloned from. Running `git pull` generally fetches data from the server you originally cloned from and automatically tries to merge it into the code you're currently working on.

Pushing to Your Remotes

When you have your project at a point that you want to share, you have to push it upstream. The command for this is simple: `git push <remote> <branch>`. If you want to push your master branch to your `origin` server (again, cloning generally sets up both of those names for you automatically), then you can run this to push any commits you've done back up to the server:

```
$ git push origin master
```

This command works only if you cloned from a server to which you have write access and if nobody has pushed in the meantime. If you and someone else clone at the same time and they push upstream and then you push upstream, your push will rightly be rejected. You'll have to fetch their work first and incorporate it into yours before you'll be allowed to push. See Git Branching for more detailed information on how to push to remote servers.

Inspecting a Remote

If you want to see more information about a particular remote, you can use the `git remote show <remote>` command. If you run this command with a particular shortname, such as `origin`, you get something like this:

```
$ git remote show origin
* remote origin
  Fetch URL: https://github.com/schacon/ticgit
  Push  URL: https://github.com/schacon/ticgit
  HEAD branch: master
  Remote branches:
    master                               tracked
    dev-branch                           tracked
  Local branch configured for 'git pull':
```

```
    master merges with remote master
  Local ref configured for 'git push':
    master pushes to master (up to date)
```

It lists the URL for the remote repository as well as the tracking branch information. The command helpfully tells you that if you're on the master branch and you run `git pull`, it will automatically merge in the master branch on the remote after it fetches all the remote references. It also lists all the remote references it has pulled down.

That is a simple example you're likely to encounter. When you're using Git more heavily, however, you may see much more information from `git remote show`:

```
$ git remote show origin
* remote origin
  URL: https://github.com/my-org/complex-project
  Fetch URL: https://github.com/my-org/complex-project
  Push  URL: https://github.com/my-org/complex-project
  HEAD branch: master
  Remote branches:
    master                           tracked
    dev-branch                       tracked
    markdown-strip                   tracked
    issue-43                         new (next fetch will store in remotes/origin)
    issue-45                         new (next fetch will store in remotes/origin)
    refs/remotes/origin/issue-11     stale (use 'git remote prune' to remove)
  Local branches configured for 'git pull':
    dev-branch merges with remote dev-branch
    master     merges with remote master
  Local refs configured for 'git push':
    dev-branch                     pushes to dev-branch                     (up to date)
    markdown-strip                 pushes to markdown-strip                 (up to date)
    master                         pushes to master                         (up to date)
```

This command shows which branch is automatically pushed to when you run `git push` while on certain branches. It also shows you which remote branches on the server you don't yet have, which remote branches you have that have been removed from the server, and multiple local branches that are able to merge automatically with their remote-tracking branch when you run `git pull`.

Renaming and Removing Remotes

You can run `git remote rename` to change a remote's shortname. For instance, if you want to rename `pb` to `paul`, you can do so with `git remote rename`:

```
$ git remote rename pb paul
$ git remote
origin
paul
```

It's worth mentioning that this changes all your remote-tracking branch names, too. What used to be referenced at `pb/master` is now at `paul/master`.

If you want to remove a remote for some reason – you've moved the server or are no longer using a particular mirror, or perhaps a contributor isn't contributing anymore – you can either use `git remote remove` or `git remote rm`:

```
$ git remote remove paul
$ git remote
origin
```

Once you delete the reference to a remote this way, all remote-tracking branches and configuration settings associated with that remote are also deleted.

prev | next
This open sourced site is hosted on GitHub.
Patches, suggestions and comments are welcome.
Git is a member of Software Freedom Conservancy