git init

This page will explore the `git init` command in depth. By the end of this page you will be informed on the core functionality and extended feature set of `git init`. This exploration includes:

- `git init` options and usage
- `.git` directory overview
- custom `git init` directory environment values
- `git init` vs. `git clone`
- `git init` bare repositories
- `git init` templates

The `git init` command creates a new Git repository. It can be used to convert an existing, unversioned project to a Git repository or initialize a new, empty repository. Most other Git commands are not available outside of an initialized repository, so this is usually the first command you'll run in a new project.

Executing `git init` creates a `.git` subdirectory in the current working directory, which contains all of the necessary Git metadata for the new repository. This metadata includes subdirectories for objects, refs, and template files. A `HEAD` file is also created which points to the currently checked out commit.

Aside from the `.git` directory, in the root directory of the project, an existing project remains unaltered (unlike SVN, Git doesn't require a `.git` subdirectory in every subdirectory).

By default, `git init` will initialize the Git configuration to the `.git` subdirectory path. The subdirectory path can be modified and customized if you would like it to live elsewhere. You can set the `$GIT_DIR` environment variable to a custom path and `git init` will initialize the Git configuration files there. Additionally you can pass the `--separate-git-dir` argument for the same result. A common use case for a separate `.git` subdirectory is to keep your system configuration "dotfiles" (`.bashrc`, `.vimrc`, etc.) in the home directory while keeping the `.git` folder elsewhere.

Usage

Compared to SVN, the `git init` command is an incredibly easy way to create new version-controlled projects. Git doesn't require you to create a repository, import files, and check out a working copy. Additionally, Git does not require any pre-existing server or admin privileges. All you have to do is cd into your project subdirectory and run `git init`, and you'll have a fully functional Git repository.

```
git init
```

Transform the current directory into a Git repository. This adds a `.git` subdirectory to the current directory and makes it possible to start recording revisions of the project.

```
git init <directory>
```

Create an empty Git repository in the specified directory. Running this command will create a new subdirectory called containing nothing but the `.git` subdirectory.

If you've already run `git init` on a project directory and it contains a `.git` subdirectory, you can safely run `git init` again on the same project directory. It will not override an existing `.git` configuration.

git init vs. git clone

A quick note: `git init` and `git clone` can be easily confused. At a high level, they can both be used to "initialize a new git repository." However, `git clone` is dependent on `git init`. `git clone` is used to create a copy of an existing repository. Internally, `git clone` first calls `git init` to create a new repository. It then copies the data from the existing repository, and checks out a new set of working files. Learn more on the git clone page.

Bare repositories --- git init --bare

```
git init --bare <directory>
```

Initialize an empty Git repository, but omit the working directory. Shared repositories should always be created with the `--bare` flag (see discussion below). Conventionally, repositories initialized with the `--bare` flag end in `.git`. For example, the bare version of a repository called `my-project` should be stored in a directory called `my-project.git`.

The `--bare` flag creates a repository that doesn't have a working directory, making it impossible to edit files and commit changes in that repository. You would create a bare repository to git push and git pull from, but never directly commit to it. Central repositories should always be created as bare repositories because pushing branches to a non-bare repository has the potential to overwrite changes. Think of `--bare` as a way to mark a repository as a storage facility, as opposed to a development environment. This means that for virtually all Git workflows, the central repository is bare, and developers local repositories are non-bare.

The most common use case for `git init --bare` is to create a remote central repository:

```
ssh <user>@<host> cd path/above/repo git init --bare my-project.git
```

First, you SSH into the server that will contain your central repository. Then, you navigate to wherever you'd like to store the project. Finally, you use the `--bare` flag to create a central storage repository. Developers would then clone `my-project.git` to create a local copy on their development machine.

git init templates

```
git init <directory> --template=<template_directory>
```

Initializes a new Git repository and copies files from the into the repository.

Templates allow you to initialize a new repository with a predefined `.git` subdirectory. You can configure a template to have default directories and files that will get copied to a new repository's `.git` subdirectory. The default Git templates usually reside in a `` `/usr/share/git-core/templates` `` directory but may be a different path on your machine.

The default templates are a good reference and example of how to utilize template features. A powerful feature of templates that's exhibited in the default templates is Git Hook configuration. You can create a template with predefined Git hooks and initialize your new git repositories with common hooks ready to go. Learn more about Git Hooks at the Git Hook page.

Configuration

All configurations of `git init` take a argument. If you provide the , the command is run inside it. If this directory does not exist, it will be created. In addition to the options and configuration already discussed, `Git init` has a few other command line options. A full list of them follows:

`-Q`

`--QUIET`

Only prints "critical level" messages, Errors, and Warnings. All other output is silenced.

`--BARE`

Creates a bare repository. (See the "Bare Repositories" section above.)

`--TEMPLATE=`

Specifies the directory from which templates will be used. (See the "Git Init Templates" section above.)

`--SEPARATE-GIT-DIR=`

Creates a text file containing the path to . This file acts as a link to the `.git` directory. This is useful if you would like to store your `.git` directory on a separate location or drive from your project's working files. Some common use cases for `--separate-git-dir` are:

- To keep your system configuration "dotfiles" (`.bashrc`, `.vimrc`, etc.) in the home directory while keeping the `.git` folder elsewhere
- Your Git history has grown very large in disk size and you need to move it elsewhere to a separate high-capacity drive
- You want to have a Git project in a publicly accessible directory like `` `www:root` ``

You can call `git init --separate-git-dir` on an existing repository and the `.git dir` will be moved to the specified path.

`--SHARED[=(FALSE|TRUE|UMASK|GROUP|ALL|WORLD|EVERYBODY|0XXX)]`

Set access permissions for the new repository. This specifies which users and groups using Unix-level permissions are allowed to push/pull to the repository.

Examples

Create a new git repository for an existing code base

```
cd /path/to/code \
git init \
git add . \
git commit
```

Create a new bare repository

```
git init --bare /path/to/repo.git
```

Create a git init template and initialize a new git repository from the template

```
mkdir -p /path/to/template \
echo "Hello World" >> /absolute/path/to/template/README \
git init /new/repo/path --template=/absolute/path/to/template \
cd /new/repo/path \
cat /new/repo/path/README
```