NAME

gitrepository-layout - Git Repository Layout

SYNOPSIS

$GIT_DIR/*

DESCRIPTION

A Git repository comes in two different flavours:

- a .git directory at the root of the working tree;
- a <project>.git directory that is a *bare* repository (i.e. without its own working tree), that is typically used for exchanging histories with others by pushing into it and fetching from it.

Note: Also you can have a plain text file .git at the root of your working tree, containing gitdir: <path> to point at the real directory that has the repository. This mechanism is often used for a working tree of a submodule checkout, to allow you in the containing superproject to git checkout a branch that does not have the submodule. The checkout has to remove the entire submodule working tree, without losing the submodule repository.

These things may exist in a Git repository.

objects
: Object store associated with this repository. Usually an object store is self sufficient (i.e. all the objects that are referred to by an object found in it are also found in it), but there are a few ways to violate it.
  1. You could have an incomplete but locally usable repository by creating a shallow clone. See **git-clone(1)**.
  2. You could be using the objects/info/alternates or $GIT_ALTERNATE_OBJECT_DIRECTORIES mechanisms to *borrow* objects from other object stores. A repository with this kind of incomplete object store is not suitable to be published for use with dumb transports but otherwise is OK as long as objects/info/alternates points at the object stores it borrows from.

objects/[0-9a-f][0-9a-f]
: A newly created object is stored in its own file. The objects are splayed over 256 subdirectories using the first two characters of the sha1 object name to keep the number of directory entries in objects itself to a manageable number. Objects found here are often called *unpacked* (or *loose*) objects.

objects/pack
: Packs (files that store many object in compressed form, along with index files to allow them to be randomly accessed) are found in this directory.

objects/info
: Additional information about the object store is recorded in this directory.

objects/info/packs
: This file is to help dumb transports discover what packs are available in this object store. Whenever a pack is added or removed, git update-server-info should be run to keep this file up-to-date if the repository is published for dumb transports. *git repack* does this by default.

objects/info/alternates
: This file records paths to alternate object stores that this object store borrows objects from, one pathname per line. Note that not only native Git tools use it locally, but the HTTP fetcher also tries to use it remotely; this will usually work if you have relative paths (relative to the object database, not to the repository!) in your alternates file, but it will not work if you use absolute paths unless the absolute path in filesystem and web URL is the same. See also *objects/info/http-alternates*.

objects/info/http-alternates
: This file records URLs to alternate object stores that this object store borrows objects from,

to be used when the repository is fetched over HTTP.

refs
: References are stored in subdirectories of this directory. The *git prune* command knows to preserve objects reachable from refs found in this directory and its subdirectories.

refs/heads/name
: records tip-of-the-tree commit objects of branch name

refs/tags/name
: records any object name (not necessarily a commit object, or a tag object that points at a commit object).

refs/remotes/name
: records tip-of-the-tree commit objects of branches copied from a remote repository.

refs/replace/<obj-sha1>
: records the SHA-1 of the object that replaces <obj-sha1>. This is similar to info/grafts and is internally used and maintained by **git-replace(1)**. Such refs can be exchanged between repositories while grafts are not.

packed-refs
: records the same information as refs/heads/, refs/tags/, and friends record in a more efficient way. See **git-pack-refs(1)**.

HEAD
: A symref (see glossary) to the refs/heads/ namespace describing the currently active branch. It does not mean much if the repository is not associated with any working tree (i.e. a *bare* repository), but a valid Git repository **must** have the HEAD file; some porcelains may use it to guess the designated default branch of the repository (usually *master*). It is legal if the named branch *name* does not (yet) exist. In some legacy setups, it is a symbolic link instead of a symref that points at the current branch.

  HEAD can also record a specific commit directly, instead of being a symref to point at the current branch. Such a state is often called *detached HEAD*. See **git-checkout(1)** for details.

branches
: A slightly deprecated way to store shorthands to be used to specify a URL to *git fetch*, *git pull* and *git push*. A file can be stored as branches/<name> and then *name* can be given to these commands in place of *repository* argument. See the REMOTES section in **git-fetch(1)** for details. This mechanism is legacy and not likely to be found in modern repositories.

hooks
: Hooks are customization scripts used by various Git commands. A handful of sample hooks are installed when *git init* is run, but all of them are disabled by default. To enable, the .sample suffix has to be removed from the filename by renaming. Read **githooks(5)** for more details about each hook.

index
: The current index file for the repository. It is usually not found in a bare repository.

sharedindex.<SHA-1>
: The shared index part, to be referenced by $GIT_DIR/index and other temporary index files. Only valid in split index mode.

info
: Additional information about the repository is recorded in this directory.

info/refs
: This file helps dumb transports discover what refs are available in this repository. If the repository is published for dumb transports, this file should be regenerated by *git update-server-info* every time a tag or branch is created or modified. This is normally done from the

hooks/update hook, which is run by the *git-receive-pack* command when you *git push* into the repository.

info/grafts
: This file records fake commit ancestry information, to pretend the set of parents a commit has is different from how the commit was actually created. One record per line describes a commit and its fake parents by listing their 40-byte hexadecimal object names separated by a space and terminated by a newline.

  Note that the grafts mechanism is outdated and can lead to problems transferring objects between repositories; see **git-replace(1)** for a more flexible and robust system to do the same thing.

info/exclude
: This file, by convention among Porcelains, stores the exclude pattern list. .gitignore is the per-directory ignore file. *git status*, *git add*, *git rm* and *git clean* look at it but the core Git commands do not look at it. See also: **gitignore(5)**.

info/sparse-checkout
: This file stores sparse checkout patterns. See also: **git-read-tree(1)**.

remotes
: Stores shorthands for URL and default refnames for use when interacting with remote repositories via *git fetch*, *git pull* and *git push* commands. See the REMOTES section in **git-fetch(1)** for details. This mechanism is legacy and not likely to be found in modern repositories.

logs
: Records of changes made to refs are stored in this directory. See **git-update-ref(1)** for more information.

logs/refs/heads/name
: Records all changes made to the branch tip named name.

logs/refs/tags/name
: Records all changes made to the tag named name.

shallow
: This is similar to info/grafts but is internally used and maintained by shallow clone mechanism. See --depth option to **git-clone(1)** and **git-fetch(1)**.

modules
: Contains the git-repositories of the submodules.

SEE ALSO

git-init(1), **git-clone(1)**, **git-fetch(1)**, **git-pack-refs(1)**, **git-gc(1)**, **git-checkout(1)**, **gitglossary(7)**, **The Git User's Manual**[1]

GIT

Part of the **git(1)** suite.

NOTES

1. The Git User's Manual
   file:///usr/share/doc/git/html/user-manual.html