NAME

git-clone - Clone a repository into a new directory

SYNOPSIS

git clone [--template=<template_directory>]
[-l] [-s] [--no-hardlinks] [-q] [-n] [--bare] [--mirror]
[-o <name>] [-b <name>] [-u <upload-pack>] [--reference <repository>]
[--separate-git-dir <git dir>]
[--depth <depth>] [--[no-]single-branch]
[--recursive | --recurse-submodules] [--] <repository>
[<directory>]

DESCRIPTION

Clones a repository into a newly created directory, creates remote-tracking branches for each branch in the cloned repository (visible using git branch -r), and creates and checks out an initial branch that is forked from the cloned repository's currently active branch.

After the clone, a plain git fetch without arguments will update all the remote-tracking branches, and a git pull without arguments will in addition merge the remote master branch into the current master branch, if any (this is untrue when --single-branch is given; see below).

This default configuration is achieved by creating references to the remote branch heads under refs/remotes/origin and by initializing remote.origin.url and remote.origin.fetch configuration variables.

OPTIONS

--local, -l

When the repository to clone from is on a local machine, this flag bypasses the normal Git aware transport mechanism and clones the repository by making a copy of HEAD and everything under objects and refs directories. The files under .git/objects/ directory are hardlinked to save space when possible.

If the repository is specified as a local path (e.g., /path/to/repo), this is the default, and --local is essentially a no-op. If the repository is specified as a URL, then this flag is ignored (and we never use the local optimizations). Specifying --no-local will override the default when /path/to/repo is given, using the regular Git transport instead.

--no-hardlinks

Force the cloning process from a repository on a local filesystem to copy the files under the .git/objects directory instead of using hardlinks. This may be desirable if you are trying to make a back-up of your repository.

--shared, -s

When the repository to clone is on the local machine, instead of using hard links, automatically setup .git/objects/info/alternates to share the objects with the source repository. The resulting repository starts out without any object of its own.

NOTE: this is a possibly dangerous operation; do **not** use it unless you understand what it does. If you clone your repository using this option and then delete branches (or use any other Git command that makes any existing commit unreferenced) in the source repository, some objects may become unreferenced (or dangling). These objects may be removed by normal Git operations (such as git commit) which automatically call git gc --auto. (See **git-gc(1)**.) If these objects are removed and were referenced by the cloned repository, then the cloned repository will become corrupt.

Note that running git repack without the -l option in a repository cloned with -s will copy objects from the source repository into a pack in the cloned repository, removing the disk space savings of clone -s. It is safe, however, to run git gc, which uses the -l option by default.

If you want to break the dependency of a repository cloned with -s on its source repository, you can simply run git repack -a to copy all objects from the source repository into a pack in the cloned repository.

--reference <repository>
: If the reference repository is on the local machine, automatically setup .git/objects/info/alternates to obtain objects from the reference repository. Using an already existing repository as an alternate will require fewer objects to be copied from the repository being cloned, reducing network and local storage costs.

  NOTE: see the NOTE for the --shared option.

--quiet, -q
: Operate quietly. Progress is not reported to the standard error stream. This flag is also passed to the ‘rsync’ command when given.

--verbose, -v
: Run verbosely. Does not affect the reporting of progress status to the standard error stream.

--progress
: Progress status is reported on the standard error stream by default when it is attached to a terminal, unless -q is specified. This flag forces progress status even if the standard error stream is not directed to a terminal.

--no-checkout, -n
: No checkout of HEAD is performed after the clone is complete.

--bare
: Make a *bare* Git repository. That is, instead of creating <directory> and placing the administrative files in <directory>/.git, make the <directory> itself the $GIT_DIR. This obviously implies the -n because there is nowhere to check out the working tree. Also the branch heads at the remote are copied directly to corresponding local branch heads, without mapping them to refs/remotes/origin/. When this option is used, neither remote-tracking branches nor the related configuration variables are created.

--mirror
: Set up a mirror of the source repository. This implies --bare. Compared to --bare, --mirror not only maps local branches of the source to local branches of the target, it maps all refs (including remote-tracking branches, notes etc.) and sets up a refspec configuration such that all these refs are overwritten by a git remote update in the target repository.

--origin <name>, -o <name>
: Instead of using the remote name origin to keep track of the upstream repository, use <name>.

--branch <name>, -b <name>
: Instead of pointing the newly created HEAD to the branch pointed to by the cloned repository’s HEAD, point to <name> branch instead. In a non-bare repository, this is the branch that will be checked out. --branch can also take tags and detaches the HEAD at that commit in the resulting repository.

--upload-pack <upload-pack>, -u <upload-pack>
: When given, and the repository to clone from is accessed via ssh, this specifies a non-default path for the command run on the other end.

--template=<template_directory>
: Specify the directory from which templates will be used; (See the TEMPLATE DIRECTORY section of **git-init(1)**.)

--config <key>=<value>, -c <key>=<value>
: Set a configuration variable in the newly-created repository; this takes effect immediately after the repository is initialized, but before the remote history is fetched or any files checked out. The key is in the same format as expected by **git-config(1)** (e.g., core.eol=true). If

multiple values are given for the same key, each value will be written to the config file. This makes it safe, for example, to add additional fetch refspecs to the origin remote.

--depth <depth>
: Create a *shallow* clone with a history truncated to the specified number of revisions.

--[no-]single-branch
: Clone only the history leading to the tip of a single branch, either specified by the --branch option or the primary branch remote's HEAD points at. When creating a shallow clone with the --depth option, this is the default, unless --no-single-branch is given to fetch the histories near the tips of all branches. Further fetches into the resulting repository will only update the remote-tracking branch for the branch this option was used for the initial cloning. If the HEAD at the remote did not point at any branch when --single-branch clone was made, no remote-tracking branch is created.

--recursive, --recurse-submodules
: After the clone is created, initialize all submodules within, using their default settings. This is equivalent to running git submodule update --init --recursive immediately after the clone is finished. This option is ignored if the cloned repository does not have a worktree/checkout (i.e. if any of --no-checkout/-n, --bare, or --mirror is given)

--separate-git-dir=<git dir>
: Instead of placing the cloned repository where it is supposed to be, place the cloned repository at the specified directory, then make a filesystem-agnostic Git symbolic link to there. The result is Git repository can be separated from working tree.

<repository>
: The (possibly remote) repository to clone from. See the URLS section below for more information on specifying repositories.

<directory>
: The name of a new directory to clone into. The humanish part of the source repository is used if no directory is explicitly given (repo for /path/to/repo.git and foo for host.xz:foo/.git). Cloning into an existing directory is only allowed if the directory is empty.

GIT URLS

In general, URLs contain information about the transport protocol, the address of the remote server, and the path to the repository. Depending on the transport protocol, some of this information may be absent.

Git supports ssh, git, http, and https protocols (in addition, ftp, and ftps can be used for fetching and rsync can be used for fetching and pushing, but these are inefficient and deprecated; do not use them).

The native transport (i.e. git:// URL) does no authentication and should be used with caution on unsecured networks.

The following syntaxes may be used with them:

- ssh://[user@]host.xz[:port]/path/to/repo.git/
- git://host.xz[:port]/path/to/repo.git/
- http[s]://host.xz[:port]/path/to/repo.git/
- ftp[s]://host.xz[:port]/path/to/repo.git/
- rsync://host.xz/path/to/repo.git/

An alternative scp-like syntax may also be used with the ssh protocol:

- [user@]host.xz:path/to/repo.git/

This syntax is only recognized if there are no slashes before the first colon. This helps differentiate a local path that contains a colon. For example the local path foo:bar could be specified as an absolute path or ./foo:bar to avoid being misinterpreted as an ssh url.

The ssh and git protocols additionally support ~username expansion:

- ssh://[user@]host.xz[:port]/~[user]/path/to/repo.git/
- git://host.xz[:port]/~[user]/path/to/repo.git/
- [user@]host.xz:/~[user]/path/to/repo.git/

For local repositories, also supported by Git natively, the following syntaxes may be used:

- /path/to/repo.git/
- file:///path/to/repo.git/

These two syntaxes are mostly equivalent, except the former implies --local option.

When Git doesn't know how to handle a certain transport protocol, it attempts to use the *remote-<transport>* remote helper, if one exists. To explicitly request a remote helper, the following syntax may be used:

- <transport>::<address>

where <address> may be a path, a server and path, or an arbitrary URL-like string recognized by the specific remote helper being invoked. See **gitremote-helpers(1)** for details.

If there are a large number of similarly-named remote repositories and you want to use a different format for them (such that the URLs you use will be rewritten into URLs that work), you can create a configuration section of the form:

```
[url <actual url base>]
insteadOf = <other url base>
```

For example, with this:

```
[url git://git.host.xz/]
insteadOf = host.xz:/path/to/
insteadOf = work:
```

a URL like work:repo.git or like host.xz:/path/to/repo.git will be rewritten in any context that takes a URL to be git://git.host.xz/repo.git.

If you want to rewrite URLs for push only, you can create a configuration section of the form:

```
[url <actual url base>]
pushInsteadOf = <other url base>
```

For example, with this:

```
[url ssh://example.org/]
pushInsteadOf = git://example.org/
```

a URL like git://example.org/path/to/repo.git will be rewritten to ssh://example.org/path/to/repo.git for pushes, but pulls will still use the original URL.

EXAMPLES

- Clone from upstream:

  ```
  $ git clone git://git.kernel.org/pub/scm/.../linux.git my-linux
  $ cd my-linux
  $ make
  ```

- Make a local clone that borrows from the current directory, without checking things out:

  ```
  $ git clone -l -s -n . ../copy
  $ cd ../copy
  $ git show-branch
  ```

- Clone from upstream while borrowing from an existing local directory:

  ```
  $ git clone --reference /git/linux.git
  git://git.kernel.org/pub/scm/.../linux.git
  my-linux
  ```

```
$ cd my-linux
```

- Create a bare repository to publish your changes to the public:

```
$ git clone --bare -l /home/proj/.git /pub/scm/proj.git
```

GIT

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