git-clone

git-clone(1) ==========

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>] [--dissociate] [--separate-git-dir <git-dir>] [--depth <depth>] [--[no-]single-branch] [--[no-]tags] [--recurse-submodules[=<pathspec>]] [--[no-]shallow-submodules] [--[no-]remote-submodules] [--jobs <n>] [--sparse] [--[no-]reject-shallow] [--filter=<filter-spec> [--also-filter-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` `--remotes`), 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 ------- `-l`:: `--local`:: 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.
If the repository’s `$GIT_DIR/objects` has symbolic links or is a symbolic link, the clone will fail. This is a security measure to prevent the unintentional copying of files by dereferencing the symbolic links.
This option does not work with repositories owned by other users for security reasons, and `--no-local` must be specified for the clone to succeed.
NOTE: this operation can race with concurrent modification to the source repository, similar to

running `cp` `-r` `<src>` `<dst>` while modifying <*src*>.

`--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.

`-s`

`--shared`
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` `maintenance` `run` `--auto`. (See linkgit:git-maintenance[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 `--local` option in a repository cloned with `--shared` will copy objects from the source repository into a pack in the cloned repository, removing the disk space savings of `clone` `--shared`. It is safe, however, to run `git` `gc`, which uses the `--local` option by default.

If you want to break the dependency of a repository cloned with `--shared` 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>`

`--reference-if-able=<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. When using the `--reference-if-able`, a non existing directory is skipped with a warning instead of aborting the clone.

> [!NOTE] see the NOTE for the `--shared` option, and also the `--dissociate` option.

`--dissociate`
Borrow the objects from reference repositories specified with the `--reference` options only to reduce network transfer, and stop borrowing from them after a clone is made by making necessary local copies of borrowed objects. This option can also be used when cloning locally from a repository that already borrows objects from another repository—the new repository will borrow objects from the same repository, and this option can be used to stop the borrowing.

`-q`

`--quiet`
Operate quietly. Progress is not reported to the standard error stream.

`-v`

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

`--progress`
Report progress status on the standard error stream by default when attached to a terminal, unless `--quiet` is specified. This flag forces progress status even if the standard error stream is not directed to a terminal.

`--server-option=<option>`
Transmit the given string to the server when communicating using protocol version 2. The given string must not contain a *NUL* or *LF* character. The server's handling of server options, including unknown ones, is server-specific. When multiple `--server-option=<option>` are given, they are all sent to the other side in the order listed on the command line. When no `--server-option=<option>` is given from the command line, the values of configuration variable `remote.<name>.serverOption` are used instead.

`-n`

`--no-checkout`
Do not checkout `HEAD` after the clone is complete.

`--no-reject-shallow`

`--reject-shallow`
Fail if the source repository is a shallow repository. The `clone.rejectShallow` configuration variable can be used to specify the default.

`--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 `--no-checkout` 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.

`--sparse`
Employ a sparse-checkout, with only files in the toplevel directory initially being present. The linkgit:git-sparse-checkout[1] command can be used to grow the working directory as needed.

`--filter=<filter-spec>`
Use the partial clone feature and request that the server sends a subset of reachable objects according to a given object filter. When using `--filter`, the supplied <*filter-spec*> is used for the partial clone filter.

If `--filter=auto` is used the filter specification is determined automatically through the 'promisor-remote' protocol (see linkgit:gitprotocol-v2[5]) by combining the filter specifications advertised by the server for the promisor remotes that the client accepts (see the `promisor.acceptFromServer` configuration option in linkgit:git-config[1]). This allows the server to suggest the optimal filter for the available promisor remotes.

As with other filter specifications, the "auto" value is persisted in the configuration. This ensures that future fetches will continue to adapt to the server's current recommendation.

For details on all other available filter specifications, see the `--filter=<filter-spec>` option in linkgit:git-rev-list[1].

For example, `--filter=blob:none` will filter out all blobs (file contents) until needed by Git. Also, `--filter=blob:limit=<size>` will filter out all blobs of size at least <*size*>.

`--also-filter-submodules`
Also apply the partial clone filter to any submodules in the repository. Requires `--filter` and `--recurse-submodules`. This can be turned on by default by setting the `clone.filterSubmodules` config option.

`--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.

`-o<name>`

`--origin=<name>`
Instead of using the remote name `origin` to keep track of the upstream repository, use <*name*>. Overrides `clone.defaultRemoteName` from the config.

`-b<name>`

`--branch=<name>`
Point the newly created `HEAD` to <*name*> branch instead of the branch pointed to by the cloned

repository's `HEAD`. 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.

`--revision=<rev>`
Create a new repository, and fetch the history leading to the given revision *<rev>* (and nothing else), without making any remote-tracking branch, and without making any local branch, and detach `HEAD` to *<rev>*. The argument can be a ref name (e.g. `refs/heads/main` or `refs/tags/v1.0`) that peels down to a commit, or a hexadecimal object name. This option is incompatible with `--branch` and `--mirror`.

`-u<upload-pack>`

`--upload-pack=<upload-pack>`
Specify a non-default path for the command run on the other end when the repository to clone from is accessed via ssh.

`--template=<template-directory>`
Specify the directory from which templates will be used; (See the "TEMPLATE DIRECTORY" section of linkgit:git-init[1].)

`-c<key>=<value>`

`--config=<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 linkgit: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.

Due to limitations of the current implementation, some configuration variables do not take effect until after the initial fetch and checkout. Configuration variables known to not take effect are: `remote.<name>.mirror` and `remote.<name>.tagOpt`. Use the corresponding `--mirror` and `--no-tags` options instead.

`--depth=<depth>`
Create a 'shallow' clone with a history truncated to the specified number of commits. Implies `--single-branch` unless `--no-single-branch` is given to fetch the histories near the tips of all branches. If you want to clone submodules shallowly, also pass `--shallow-submodules`.

`--shallow-since=<date>`
Create a shallow clone with a history after the specified time.

`--shallow-exclude=<ref>`
Create a shallow clone with a history, excluding commits reachable from a specified remote branch or tag. This option can be specified multiple times.

`--single-branch`

`--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. 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.

`--tags`

`--no-tags`
Control whether or not tags will be cloned. When `--no-tags` is given, the option will be become permanent by setting the `remote.<remote>.tagOpt=--no-tags` configuration. This ensures that future `git pull` and `git fetch` won't follow any tags. Subsequent explicit tag fetches will still work (see linkgit:git-fetch[1]).

By default, tags are cloned and passing `--tags` is thus typically a no-op, unless it cancels out a previous `--no-tags`.

Can be used in conjunction with `--single-branch` to clone and maintain a branch with no references other than a single cloned branch. This is useful e.g. to maintain minimal clones of the default branch of some repository for search indexing.

`--recurse-submodules[=<pathspec>]`
After the clone is created, initialize and clone submodules within based on the provided *<pathspec>*. If no `=<pathspec>` is provided, all submodules are initialized and cloned. This option can be given multiple times for pathspecs consisting of multiple entries. The resulting clone has `submodule.active` set to the provided pathspec, or "." (meaning all submodules) if no pathspec is provided.

Submodules are initialized and cloned using their default settings. This is equivalent to running `git submodule update --init --recursive <pathspec>` 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)

`--shallow-submodules`

`--no-shallow-submodules`
All submodules which are cloned will be shallow with a depth of 1.

`--remote-submodules`

`--no-remote-submodules`
All submodules which are cloned will use the status of the submodule's remote-tracking branch to update the submodule, rather than the superproject's recorded SHA-1. Equivalent to passing `--remote` to `git submodule update`.

`--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.

`--ref-storage-format=<format>`

Specify the given ref storage *<format>* for the repository. Backends that require additional configuration accept a payload in the form `<format>://<payload>`, for example a connection string identifying the database that shall store the references. The valid values are:
include::ref-storage-format.adoc[]

`--ref-format=<format>`
Deprecated alias of `--ref-storage-format=<format>`.

`-j<n>`

`--jobs=<n>`
The number of submodules fetched at the same time. Defaults to the `submodule.fetchJobs` option.

<repository>

The (possibly remote) *<repository>* to clone from. See the GIT 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.

`--bundle-uri=<uri>`
Before fetching from the remote, fetch a bundle from the given *<uri>* and unbundle the data into the local repository. The refs in the bundle will be stored under the hidden `refs/bundle/*` namespace. This option is incompatible with `--depth`, `--shallow-since`, and `--shallow-exclude`.

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, but this is 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>[:<port>]/<path-to-git-repo>`
- `git://<host>[:<port>]/<path-to-git-repo>`
- `http[s]://<host>[:<port>]/<path-to-git-repo>`
- `ftp[s]://<host>[:<port>]/<path-to-git-repo>`

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

- `[<user>@]<host>:/<path-to-git-repo>`

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>[:<port>]/~<user>/<path-to-git-repo>`
- `git://<host>[:<port>]/~<user>/<path-to-git-repo>`
- `[<user>@]<host>:~<user>/<path-to-git-repo>`

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

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

ifndef::git-clone[] These two syntaxes are mostly equivalent, except when cloning, when the former implies `--local` option. See linkgit:git-clone[1] for details. endif::git-clone[]

ifdef::git-clone[] These two syntaxes are mostly equivalent, except the former implies `--local` option. endif::git-clone[]

`git clone`, `git fetch` and `git pull`, but not `git push`, will also accept a suitable bundle file. See linkgit:git-bundle[1].

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 linkgit:gitremote-helpers[7] 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
```

- Clone a local repository from a different user:

```
$ git clone --no-local /home/otheruser/proj.git /pub/scm/proj.git
```

CONFIGURATION ----------

Everything below this line in this section is selectively included from the linkgit:git-config[1] documentation. The content is the same as what’s found there:

ifndef::git-init[] :see-git-init: (See the "TEMPLATE DIRECTORY" section of linkgit:git-init[1].) endif::[]

`init.templateDir`
Specify the directory from which templates will be copied.

`init.defaultBranch`
Allows overriding the default branch name e.g. when initializing a new repository.

`init.defaultObjectFormat`
Allows overriding the default object format for new repositories. See `--object-format=` in linkgit:git-init[1]. Both the command line option and the `GIT_DEFAULT_HASH` environment variable take precedence over this config.

`init.defaultRefStorageFormat`
Allows overriding the default ref storage format for new repositories. See `--ref-storage-format=` in linkgit:git-init[1]. Both the command line option and the `GIT_DEFAULT_REF_STORAGE_FORMAT` environment variable take precedence over this config.

`init.defaultRefFormat`
Deprecated alias of `init.defaultRefStorageFormat`.

init.defaultSubmodulePathConfig
A boolean that specifies if `git` `init` and `git` `clone` should automatically set `extensions.submodulePathConfig` to `true`. This allows all new repositories to automatically use the submodule path extension. Defaults to `false` when unset.

`clone.defaultRemoteName`
The name of the remote to create when cloning a repository. Defaults to `origin`. ifdef::git-clone[] It can be overridden by passing the `--origin` command-line option. endif::[] ifndef::git-clone[] It can be overridden by passing the `--origin` command-line option to linkgit:git-clone[1]. endif::[]

`clone.rejectShallow`
Reject cloning a repository if it is a shallow one; this can be overridden by passing the `--reject-shallow` option on the command line. ifndef::git-clone[] See linkgit:git-clone[1]. endif::[]

`clone.filterSubmodules`
If a partial clone filter is provided (see `--filter` in linkgit:git-rev-list[1]) and `--recurse-submodules` is used, also apply the filter to submodules.

GIT --- Part of the linkgit:git[1] suite

□□□ #1
□ □□□ □□□ 2026-10-31 20:10:13 UTC
□ □□□ □□□ 2026-10-31 20:10:13 UTC