NAME

git-fetch - Download objects and refs from another repository

SYNOPSIS

git fetch [<options>] [<repository> [<refspec>...]]
git fetch[<options>] <group>
git fetch--m ultiple [<options>] [(<repository> | <group>)...]
git fetch--all [<options>]

DESCRIPTION

Fetch branches and/or tags (collectively, refs) from one or more other repositories, along with the objects necessary to complete their histories. Remote-tracking branches are updated (see the description of <refspec> below for ways to control this behavior).

By default, any tag that points into the histories being fetched is also fetched; the effect is to fetch tags that point at branches that you are interested in. This default behavior can be changed by using the --tags or --no-tags options or by configuring remote.<name>.tagopt. By using a refspec that fetches tags explicitly, you can fetch tags that do not point into branches you are interested in as well.

*git fetch*can fetc h from either a single named repository or URL, or from several repositories at once if <group> is given and there is a remotes.<group> entry in the configuration file. (See **git-config(1)**).

When no remote is specified, by default the origin remote will be used, unless there's an upstream branch configured for the current branch.

The names of refs that are fetched, together with the object names they point at, are written to .git/FETCH_HEAD. This information may be used by scripts or other git commands, such as **git-pull(1)**.

OPTIONS

--all
: Fetch all remotes.

-a, --append
: Append ref names and object names of fetched refs to the existing contents of .git/FETCH_HEAD. Without this option old data in .git/FETCH_HEAD will be overwritten.

--depth=<depth>
: Deepen or shorten the history of a *shallow* repository created by git clone with --depth=<depth> option (see **git-clone(1)**) to the specified number of commits from the tip of each remote branch history. Tags for the deepened commits are not fetched.

--unshallow
: If the source repository is complete, convert a shallow repository to a complete one, removing all the limitations imposed by shallow repositories.

  If the source repository is shallow, fetch as much as possible so that the current repository has the same history as the source repository.

--update-shallow
: By default when fetching from a shallow repository, git fetch refuses refs that require updating .git/shallow. This option updates .git/shallow and accept such refs.

--dry-run
: Show what would be done, without making any changes.

-f, --force
: When *git fetch* is used with <rbranch>:<lbranch> refspec, it refuses to update the local branch <lbranch> unless the remote branch <rbranch> it fetches is a descendant of <lbranch>.

This option overrides that check.

-k, --keep
: Keep downloaded pack.

--multiple
: Allow several <repository> and <group> arguments to be specified. No <refspec>s may be specified.

-p, --prune
: After fetching, remove any remote-tracking references that no longer exist on the remote. Tags are not subject to pruning if they are fetched only because of the default tag auto-following or due to a --tags option. However, if tags are fetched due to an explicit refspec (either on the command line or in the remote configuration, for example if the remote was cloned with the --mirror option), then they are also subject to pruning.

-n, --no-tags
: By default, tags that point at objects that are downloaded from the remote repository are fetched and stored locally. This option disables this automatic tag following. The default behavior for a remote may be specified with the remote.<name>.tagopt setting. See **git-config(1)**.

--refmap=<refspec>
: When fetching refs listed on the command line, use the specified refspec (can be given more than once) to map the refs to remote-tracking branches, instead of the values of remote.*.fetch configuration variables for the remote repository. See section on Configured Remote-tracking Branches for details.

-t, --tags
: Fetch all tags from the remote (i.e., fetch remote tags refs/tags/* into local tags with the same name), in addition to whatever else would otherwise be fetched. Using this option alone does not subject tags to pruning, even if --prune is used (though tags may be pruned anyway if they are also the destination of an explicit refspec; see *--prune*).

--recurse-submodules[=yes|on-demand|no]
: This option controls if and under what conditions new commits of populated submodules should be fetched too. It can be used as a boolean option to completely disable recursion when set to *no* or to unconditionally recurse into all populated submodules when set to *yes*, which is the default when this option is used without any value. Use *on-demand* to only recurse into a populated submodule when the superproject retrieves a commit that updates the submodule's reference to a commit that isn't already in the local submodule clone.

--no-recurse-submodules
: Disable recursive fetching of submodules (this has the same effect as using the *--recurse-submodules=no* option).

--submodule-prefix=<path>
: Prepend <path> to paths printed in informative messages such as Fetching submodule foo. This option is used internally when recursing over submodules.

--recurse-submodules-default=[yes|on-demand]
: This option is used internally to temporarily provide a non-negative default value for the --recurse-submodules option. All other methods of configuring fetch's submodule recursion (such as settings in **gitmodules(5)** and **git-config(1)**) override this option, as does specifying --[no-]recurse-submodules directly.

-u, --update-head-ok
: By default *git fetch* refuses to update the head which corresponds to the current branch. This flag disables the check. This is purely for the internal use for *git pull* to communicate with *git fetch*, and unless you are implementing your own Porcelain you are not supposed to use it.

--upload-pack <upload-pack>

When given, and the repository to fetch from is handled by *git fetch-pack*, *--exec=<upload-pack>* is passed to the command to specify non-default path for the command run on the other end.

-q, --quiet
: Pass --quiet to git-fetch-pack and silence any other internally used git commands. Progress is not reported to the standard error stream.

-v, --verbose
: Be verbose.

--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.

<repository>
: The remote repository that is the source of a fetch or pull operation. This parameter can be either a URL (see the section GIT URLS below) or the name of a remote (see the section REMOTES below).

<group>
: A name referring to a list of repositories as the value of remotes.<group> in the configuration file. (See **git-config(1)**).

<refspec>
: Specifies which refs to fetch and which local refs to update. When no <refspec>s appear on the command line, the refs to fetch are read from remote.<repository>.fetch variables instead (see CONFIGURED REMOTE-TRACKING BRANCHES below).

The format of a <refspec> parameter is an optional plus +, followed by the source ref <src>, followed by a colon :, followed by the destination ref <dst>. The colon can be omitted when <dst> is empty.

tag <tag> means the same as refs/tags/<tag>:refs/tags/<tag>; it requests fetching everything up to the given tag.

The remote ref that matches <src> is fetched, and if <dst> is not empty string, the local ref that matches it is fast-forwarded using <src>. If the optional plus + is used, the local ref is updated even if it does not result in a fast-forward update.

Note

When the remote branch you want to fetch is known to be rewound and rebased regularly, it is expected that its new tip will not be descendant of its previous tip (as stored in your remote-tracking branch the last time you fetched). You would want to use the + sign to indicate non-fast-forward updates will be needed for such branches. There is no way to determine or declare that a branch will be made available in a repository with this behavior; the pulling user simply must know this is the expected usage pattern for a branch.

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 when cloning, when the former implies --local option. See **git-clone(1)** for details.

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.

REMOTES

The name of one of the following can be used instead of a URL as <repository> argument:

- a remote in the Git configuration file: $GIT_DIR/config,
- a file in the $GIT_DIR/remotes directory, or
- a file in the $GIT_DIR/branches directory.

All of these also allow you to omit the refspec from the command line because they each contain a refspec which git will use by default.

Named remote in configuration file

You can choose to provide the name of a remote which you had previously configured using **git-remote(1)**, **git-config(1)** or even by a manual edit to the $GIT_DIR/config file. The URL of this remote will be used to access the repository. The refspec of this remote will be used by default when you do not provide a refspec on the command line. The entry in the config file would appear like this:

```
[remote <name>]
url = <url>
pushurl = <pushurl>
push = <refspec>
fetch = <refspec>
```

The <pushurl> is used for pushes only. It is optional and defaults to <url>.

Named file in $GIT_DIR/remotes

You can choose to provide the name of a file in $GIT_DIR/remotes. The URL in this file will be used to access the repository. The refspec in this file will be used as default when you do not provide a refspec on the command line. This file should have the following format:

```
URL: one of the above URL format
Push: <refspec>
Pull: <refspec>
```

Push: lines are used by *git push* and Pull: lines are used by *git pull* and *git fetch*. Multiple Push: and Pull: lines may be specified for additional branch mappings.

Named file in $GIT_DIR/branches

You can choose to provide the name of a file in $GIT_DIR/branches. The URL in this file will be used to access the repository. This file should have the following format:

```
<url>#<head>
```

<url> is required; #<head> is optional.

Depending on the operation, git will use one of the following refspecs, if you don't provide one on the command line. <branch> is the name of this file in $GIT_DIR/branches and <head> defaults to master.

git fetch uses:

```
refs/heads/<head>:refs/heads/<branch>
```

git push uses:

```
HEAD:refs/heads/<head>
```

CONFIGURED REMOTE-TRACKING BRANCHES

You often interact with the same remote repository by regularly and repeatedly fetching from it. In order to keep track of the progress of such a remote repository, git fetch allows you to configure remote.<repository>.fetch configuration variables.

Typically such a variable may look like this:

```
[remote origin]
fetch = +refs/heads/*:refs/remotes/origin/*
```

This configuration is used in two ways:

- When git fetch is run without specifying what branches and/or tags to fetch on the command line, e.g. git fetch origin or git fetch, remote.<repository>.fetch values are used as the refspecs---they specify which refs to fetch and which local refs to update. The example above will fetch all branches that exist in the origin (i.e. any ref that matches the left-hand side of the value, refs/heads/*) and update the corresponding remote-tracking branches in the refs/remotes/origin/* hierarchy.
- When git fetch is run with explicit branches and/or tags to fetch on the command line, e.g. git fetch origin master, the <refspec>s given on the command line determine what are to be fetched (e.g. master in the example, which is a short-hand for master:, which in turn means fetch the *master* branch but I do not explicitly say what remote-tracking branch to update with it from the command line), and the example command will fetch *only* the *master* branch. The remote.<repository>.fetch values determine which remote-tracking branch, if any, is updated. When used in this way, the remote.<repository>.fetch values do not have any effect in deciding *what* gets fetched (i.e. the values are not used as refspecs when the command-line lists refspecs); they are only used to decide *where* the refs that are fetched are stored by acting as a mapping.

The latter use of the remote.<repository>.fetch values can be overridden by giving the --refmap=<refspec> parameter(s) on the command line.

EXAMPLES

- Update the remote-tracking branches:

  ```
  $ git fetch origin
  ```

  The above command copies all branches from the remote refs/heads/ namespace and stores them to the local refs/remotes/origin/ namespace, unless the branch.<name>.fetch option is used to specify a non-default refspec.
- Using refspecs explicitly:

  ```
  $ git fetch origin +pu:pu maint:tmp
  ```

  This updates (or creates, as necessary) branches pu and tmp in the local repository by fetching from the branches (respectively) pu and maint from the remote repository.

  The pu branch will be updated even if it is does not fast-forward, because it is prefixed with a plus sign; tmp will not be.
- Peek at a remote's branch, without configuring the remote in your local repository:

  ```
  $ git fetch git://git.kernel.org/pub/scm/git/git.git maint
  $ git log FETCH_HEAD
  ```

  The first command fetches the maint branch from the repository at git://git.kernel.org/pub/scm/git/git.git and the second command uses FETCH_HEAD to examine the branch with **git-log(1)**. The fetched objects will eventually be removed by git's built-in housekeeping (see **git-gc(1)**).

BUGS

Using --recurse-submodules can only fetch new commits in already checked out submodules right now. When e.g. upstream added a new submodule in the just fetched commits of the superproject the submodule itself can not be fetched, making it impossible to check out that submodule later without having to do a fetch again. This is expected to be fixed in a future Git version.

SEE ALSO

git-pull(1)

GIT

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