## NAME

git-push - Update remote refs along with associated objects

## SYNOPSIS

*git push* [--all | --mirror | --tags] [--follow-tags] [-n | --dry-run] [--receive-pack=<git-receive-pack>]
[--repo=<repository>] [-f | --force] [--prune] [-v | --verbose] [-u | --set-upstream]
[--force-with-lease[=<refname>[:<expect>]]]
[--no-verify] [<repository> [<refspec>...]]

## DESCRIPTION

Updates remote refs using local refs, while sending objects necessary to complete the given refs.

You can make interesting things happen to a repository every time you push into it, by setting up *hooks* there. See documentation for **git-receive-pack(1)**.

When the command line does not specify where to push with the <repository> argument, branch.*.remote configuration for the current branch is consulted to determine where to push. If the configuration is missing, it defaults to *origin*.

When the command line does not specify what to push with <refspec>... arguments or --all, --mirror, --tags options, the command finds the default <refspec> by consulting remote.*.push configuration, and if it is not found, honors push.default configuration to decide what to push (See **git-config(1)** for the meaning of push.default).

## OPTIONS

<repository>

The remote repository that is destination of a push 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).

<refspec>...

Specify what destination ref to update with what source object. The format of a <refspec> parameter is an optional plus +, followed by the source object <src>, followed by a colon :, followed by the destination ref <dst>.

The <src> is often the name of the branch you would want to push, but it can be any arbitrary SHA-1 expression, such as master˜4 or HEAD (see **gitrevisions(7)**).

The <dst> tells which ref on the remote side is updated with this push. Arbitrary expressions cannot be used here, an actual ref must be named. If git push [<repository>] without any <refspec> argument is set to update some ref at the destination with <src> with remote.<repository>.push configuration variable, :<dst> part can be omitted---such a push will update a ref that <src> normally updates without any <refspec> on the command line. Otherwise, missing :<dst> means to update the same ref as the <src>.

The object referenced by <src> is used to update the <dst> reference on the remote side. By default this is only allowed if <dst> is not a tag (annotated or lightweight), and then only if it can fast-forward <dst>. By having the optional leading +, you can tell Git to update the <dst> ref even if it is not allowed by default (e.g., it is not a fast-forward.) This does **not** attempt to merge <src> into <dst>. See EXAMPLES below for details.

tag <tag> means the same as refs/tags/<tag>:refs/tags/<tag>.

Pushing an empty <src> allows you to delete the <dst> ref from the remote repository.

The special refspec : (or +: to allow non-fast-forward updates) directs Git to push matching branches: for every branch that exists on the local side, the remote side is updated if a branch of the same name already exists on the remote side.

--all

Push all branches (i.e. refs under refs/heads/); cannot be used with other <refspec>.

--prune
: Remove remote branches that don't have a local counterpart. For example a remote branch tmp will be removed if a local branch with the same name doesn't exist any more. This also respects refspecs, e.g. git push --prune remote refs/heads/*:refs/tmp/* would make sure that remote refs/tmp/foo will be removed if refs/heads/foo doesn't exist.

--mirror
: Instead of naming each ref to push, specifies that all refs under refs/ (which includes but is not limited to refs/heads/, refs/remotes/, and refs/tags/) be mirrored to the remote repository. Newly created local refs will be pushed to the remote end, locally updated refs will be force updated on the remote end, and deleted refs will be removed from the remote end. This is the default if the configuration option remote.<remote>.mirror is set.

-n, --dry-run
: Do everything except actually send the updates.

--porcelain
: Produce machine-readable output. The output status line for each ref will be tab-separated and sent to stdout instead of stderr. The full symbolic names of the refs will be given.

--delete
: All listed refs are deleted from the remote repository. This is the same as prefixing all refs with a colon.

--tags
: All refs under refs/tags are pushed, in addition to refspecs explicitly listed on the command line.

--follow-tags
: Push all the refs that would be pushed without this option, and also push annotated tags in refs/tags that are missing from the remote but are pointing at commit-ish that are reachable from the refs being pushed.

--receive-pack=<git-receive-pack>, --exec=<git-receive-pack>
: Path to the *git-receive-pack* program on the remote end. Sometimes useful when pushing to a remote repository over ssh, and you do not have the program in a directory on the default $PATH.

--[no-]force-with-lease, --force-with-lease=<refname>, --force-with-lease=<refname>:<expect>
: Usually, git push refuses to update a remote ref that is not an ancestor of the local ref used to overwrite it.

  This option bypasses the check, but instead requires that the current value of the ref to be the expected value. git push fails otherwise.

  Imagine that you have to rebase what you have already published. You will have to bypass the must fast-forward rule in order to replace the history you originally published with the rebased history. If somebody else built on top of your original history while you are rebasing, the tip of the branch at the remote may advance with her commit, and blindly pushing with --force will lose her work.

  This option allows you to say that you expect the history you are updating is what you rebased and want to replace. If the remote ref still points at the commit you specified, you can be sure that no other people did anything to the ref (it is like taking a lease on the ref without explicitly locking it, and you update the ref while making sure that your earlier lease is still valid).

  --force-with-lease alone, without specifying the details, will protect all remote refs that are going to be updated by requiring their current value to be the same as the remote-tracking branch we have for them, unless specified with a --force-with-lease=<refname>:<expect> option that explicitly states what the expected value is.

--force-with-lease=<refname>, without specifying the expected value, will protect the named ref (alone), if it is going to be updated, by requiring its current value to be the same as the remote-tracking branch we have for it.

--force-with-lease=<refname>:<expect> will protect the named ref (alone), if it is going to be updated, by requiring its current value to be the same as the specified value <expect> (which is allowed to be different from the remote-tracking branch we have for the refname, or we do not even have to have such a remote-tracking branch when this form is used).

Note that all forms other than --force-with-lease=<refname>:<expect> that specifies the expected current value of the ref explicitly are still experimental and their semantics may change as we gain experience with this feature.

--no-force-with-lease will cancel all the previous --force-with-lease on the command line.

-f, --force
: Usually, the command refuses to update a remote ref that is not an ancestor of the local ref used to overwrite it. Also, when --force-with-lease option is used, the command refuses to update a remote ref whose current value does not match what is expected.

This flag disables these checks, and can cause the remote repository to lose commits; use it with care.

Note that --force applies to all the refs that are pushed, hence using it with push.default set to matching or with multiple push destinations configured with remote.*.push may overwrite refs other than the current branch (including local refs that are strictly behind their remote counterpart). To force a push to only one branch, use a + in front of the refspec to push (e.g git push origin +master to force a push to the master branch). See the <refspec>... section above for details.

--repo=<repository>
: This option is only relevant if no <repository> argument is passed in the invocation. In this case, *git push* derives the remote name from the current branch: If it tracks a remote branch, then that remote repository is pushed to. Otherwise, the name origin is used. For this latter case, this option can be used to override the name origin. In other words, the difference between these two commands

```
git push public #1
git push --repo=public #2
```

is that #1 always pushes to public whereas #2 pushes to public only if the current branch does not track a remote branch. This is useful if you write an alias or script around *git push*.

-u, --set-upstream
: For every branch that is up to date or successfully pushed, add upstream (tracking) reference, used by argument-less **git-pull(1)** and other commands. For more information, see *branch.<name>.merge* in **git-config(1)**.

--[no-]thin
: These options are passed to **git-send-pack(1)**. A thin transfer significantly reduces the amount of sent data when the sender and receiver share many of the same objects in common. The default is --thin.

-q, --quiet
: Suppress all output, including the listing of updated refs, unless an error occurs. Progress is not reported to the standard error stream.

-v, --verbose
: Run verbosely.

--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.

--recurse-submodules=check|on-demand
: Make sure all submodule commits used by the revisions to be pushed are available on a remote-tracking branch. If *check* is used Git will verify that all submodule commits that changed in the revisions to be pushed are available on at least one remote of the submodule. If any commits are missing the push will be aborted and exit with non-zero status. If *on-demand* is used all submodules that changed in the revisions to be pushed will be pushed. If on-demand was not able to push all necessary revisions it will also be aborted and exit with non-zero status.

--[no-]verify
: Toggle the pre-push hook (see **githooks(5)**). The default is --verify, giving the hook a chance to prevent the push. With --no-verify, the hook is bypassed completely.

## 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>

## OUTPUT

The output of git push depends on the transport method used; this section describes the output when pushing over the Git protocol (either locally or via ssh).

The status of the push is output in tabular form, with each line representing the status of a single ref. Each line is of the form:

<flag> <summary> <from> -> <to> (<reason>)

If --porcelain is used, then each line of the output is of the form:

<flag> t <from>:<to> t <summary> (<reason>)

The status of up-to-date refs is shown only if --porcelain or --verbose option is used.

flag

A single character indicating the status of the ref:

(space)

for a successfully pushed fast-forward;

+

for a successful forced update;

-

for a successfully deleted ref;

*

for a successfully pushed new ref;

!

for a ref that was rejected or failed to push; and

=

for a ref that was up to date and did not need pushing.

summary

For a successfully pushed ref, the summary shows the old and new values of the ref in a form suitable for using as an argument to git log (this is <old>..<new> in most cases, and <old>...<new> for forced non-fast-forward updates).

For a failed update, more details are given:

rejected

Git did not try to send the ref at all, typically because it is not a fast-forward and you did not force the update.

remote rejected
: The remote end refused the update. Usually caused by a hook on the remote side, or because the remote repository has one of the following safety options in effect: receive.denyCurrentBranch (for pushes to the checked out branch), receive.denyNonFastForwards (for forced non-fast-forward updates), receive.denyDeletes or receive.denyDeleteCurrent. See **git-config(1)**.

remote failure
: The remote end did not report the successful update of the ref, perhaps because of a temporary error on the remote side, a break in the network connection, or other transient error.

from
: The name of the local ref being pushed, minus its refs/<type>/ prefix. In the case of deletion, the name of the local ref is omitted.

to
: The name of the remote ref being updated, minus its refs/<type>/ prefix.

reason
: A human-readable explanation. In the case of successfully pushed refs, no explanation is needed. For a failed ref, the reason for failure is described.

## NOTE ABOUT FAST-FORWARDS

When an update changes a branch (or more in general, a ref) that used to point at commit A to point at another commit B, it is called a fast-forward update if and only if B is a descendant of A.

In a fast-forward update from A to B, the set of commits that the original commit A built on top of is a subset of the commits the new commit B builds on top of. Hence, it does not lose any history.

In contrast, a non-fast-forward update will lose history. For example, suppose you and somebody else started at the same commit X, and you built a history leading to commit B while the other person built a history leading to commit A. The history looks like this:

```
B
/
---X---A
```

Further suppose that the other person already pushed changes leading to A back to the original repository from which you two obtained the original commit X.

The push done by the other person updated the branch that used to point at commit X to point at commit A. It is a fast-forward.

But if you try to push, you will attempt to update the branch (that now points at A) with commit B. This does *not* fast-forward. If you did so, the changes introduced by commit A will be lost, because everybody will now start building on top of B.

The command by default does not allow an update that is not a fast-forward to prevent such loss of history.

If you do not want to lose your work (history from X to B) or the work by the other person (history from X to A), you would need to first fetch the history from the repository, create a history that contains changes done by both parties, and push the result back.

You can perform git pull, resolve potential conflicts, and git push the result. A git pull will create a merge commit C between commits A and B.

```
B---C
/ /
---X---A
```

Updating A with the resulting merge commit will fast-forward and your push will be accepted.

Alternatively, you can rebase your change between X and B on top of A, with git pull --rebase, and push the result back. The rebase will create a new commit D that builds the change between X and B on top of A.

```
B D
/ /
---X---A
```

Again, updating A with this commit will fast-forward and your push will be accepted.

There is another common situation where you may encounter non-fast-forward rejection when you try to push, and it is possible even when you are pushing into a repository nobody else pushes into. After you push commit A yourself (in the first picture in this section), replace it with git commit --amend to produce commit B, and you try to push it out, because forgot that you have pushed A out already. In such a case, and only if you are certain that nobody in the meantime fetched your earlier commit A (and started building on top of it), you can run git push --force to overwrite it. In other words, git push --force is a method reserved for a case where you do mean to lose history.

## EXAMPLES

git push
: Works like git push <remote>, where <remote> is the current branch's remote (or origin, if no remote is configured for the current branch).

git push origin
: Without additional configuration, pushes the current branch to the configured upstream (remote.origin.merge configuration variable) if it has the same name as the current branch, and errors out without pushing otherwise.

  The default behavior of this command when no <refspec> is given can be configured by setting the push option of the remote, or the push.default configuration variable.

  For example, to default to pushing only the current branch to origin use git config remote.origin.push HEAD. Any valid <refspec> (like the ones in the examples below) can be configured as the default for git push origin.

git push origin :
: Push matching branches to origin. See <refspec> in the OPTIONS section above for a description of matching branches.

git push origin master
: Find a ref that matches master in the source repository (most likely, it would find refs/heads/master), and update the same ref (e.g. refs/heads/master) in origin repository with it. If master did not exist remotely, it would be created.

git push origin HEAD
: A handy way to push the current branch to the same name on the remote.

git push mothership master:satellite/master dev:satellite/dev
: Use the source ref that matches master (e.g. refs/heads/master) to update the ref that matches satellite/master (most probably refs/remotes/satellite/master) in the mothership repository; do the same for dev and satellite/dev.

  This is to emulate git fetch run on the mothership using git push that is run in the opposite direction in order to integrate the work done on satellite, and is often necessary when you can only make connection in one way (i.e. satellite can ssh into mothership but mothership cannot initiate connection to satellite because the latter is behind a firewall or does not run sshd).

  After running this git push on the satellite machine, you would ssh into the mothership and run git merge there to complete the emulation of git pull that were run on mothership to pull changes made on satellite.

git push origin HEAD:master
: Push the current branch to the remote ref matching master in the origin repository. This form is convenient to push the current branch without thinking about its local name.

git push origin master:refs/heads/experimental
: Create the branch experimental in the origin repository by copying the current master branch. This form is only needed to create a new branch or tag in the remote repository when the local name and the remote name are different; otherwise, the ref name on its own will work.

git push origin :experimental
: Find a ref that matches experimental in the origin repository (e.g. refs/heads/experimental), and delete it.

git push origin +dev:master
: Update the origin repository's master branch with the dev branch, allowing non-fast-forward updates. **This can leave unreferenced commits dangling in the origin repository.** Consider the following situation, where a fast-forward is not possible:

```
o---o---o---A---B origin/master

X---Y---Z dev
```

The above command would change the origin repository to

```
A---B (unnamed branch)
/
o---o---o---X---Y---Z master
```

Commits A and B would no longer belong to a branch with a symbolic name, and so would be unreachable. As such, these commits would be removed by a git gc command on the origin repository.

## GIT

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