NAME

git-update-ref − Update the object name stored in a ref safely

SYNOPSIS

git update−ref [−m <reason>] [−−no−deref] (−d <ref> [<oldvalue>] | [−−create−reflog] <ref> <newvalue> [<oldvalue>] |

DESCRIPTION

Given two arguments, stores the <newvalue> in the <ref>, possibly dereferencing the symbolic refs. E.g. **git update−ref HEAD <newvalue>** updates the current branch head to the new object.

Given three arguments, stores the <newvalue> in the <ref>, possibly dereferencing the symbolic refs, after verifying that the current value of the <ref> matches <oldvalue>. E.g. **git update−ref refs/heads/master <newvalue> <oldvalue>** updates the master branch head to <newvalue> only if its current value is <oldvalue>. You can specify 40 "0" or an empty string as <oldvalue> to make sure that the ref you are creating does not exist.

It also allows a "ref" file to be a symbolic pointer to another ref file by starting with the four−byte header sequence of "ref:".

More importantly, it allows the update of a ref file to follow these symbolic pointers, whether they are symlinks or these "regular file symbolic refs". It follows **real** symlinks only if they start with "refs/": otherwise it will just try to read them and update them as a regular file (i.e. it will allow the filesystem to follow them, but will overwrite such a symlink to somewhere else with a regular filename).

If −−no−deref is given, <ref> itself is overwritten, rather than the result of following the symbolic pointers.

In general, using

git update−ref HEAD "$head"

should be a *lot* safer than doing

echo "$head" > "$GIT_DIR/HEAD"

both from a symlink following standpoint **and** an error checking standpoint. The "refs/" rule for symlinks means that symlinks that point to "outside" the tree are safe: they’ll be followed for reading but not for writing (so we’ll never write through a ref symlink to some other tree, if you have copied a whole archive by creating a symlink tree).

With **−d** flag, it deletes the named <ref> after verifying it still contains <oldvalue>.

With **−−stdin**, update−ref reads instructions from standard input and performs all modifications together. Specify commands of the form:

update SP <ref> SP <newvalue> [SP <oldvalue>] LF
create SP <ref> SP <newvalue> LF
delete SP <ref> [SP <oldvalue>] LF
verify SP <ref> [SP <oldvalue>] LF
option SP <opt> LF
start LF
prepare LF
commit LF
abort LF

With **−−create−reflog**, update−ref will create a reflog for each ref even if one would not ordinarily be

created.

Quote fields containing whitespace as if they were strings in C source code; i.e., surrounded by double−quotes and with backslash escapes. Use 40 "0" characters or the empty string to specify a zero value. To specify a missing value, omit the value and its preceding SP entirely.

Alternatively, use **−z** to specify in NUL−terminated format, without quoting:

update SP <ref> NUL <newvalue> NUL [<oldvalue>] NUL
create SP <ref> NUL <newvalue> NUL
delete SP <ref> NUL [<oldvalue>] NUL
verify SP <ref> NUL [<oldvalue>] NUL
option SP <opt> NUL
start NUL
prepare NUL
commit NUL
abort NUL

In this format, use 40 "0" to specify a zero value, and use the empty string to specify a missing value.

In either format, values can be specified in any form that Git recognizes as an object name. Commands in any other format or a repeated <ref> produce an error. Command meanings are:

update
: Set <ref> to <newvalue> after verifying <oldvalue>, if given. Specify a zero <newvalue> to ensure the ref does not exist after the update and/or a zero <oldvalue> to make sure the ref does not exist before the update.

create
: Create <ref> with <newvalue> after verifying it does not exist. The given <newvalue> may not be zero.

delete
: Delete <ref> after verifying it exists with <oldvalue>, if given. If given, <oldvalue> may not be zero.

verify
: Verify <ref> against <oldvalue> but do not change it. If <oldvalue> is zero or missing, the ref must not exist.

option
: Modify behavior of the next command naming a <ref>. The only valid option is **no−deref** to avoid dereferencing a symbolic ref.

start
: Start a transaction. In contrast to a non−transactional session, a transaction will automatically abort if the session ends without an explicit commit. This command may create a new empty transaction when the current one has been committed or aborted already.

prepare
: Prepare to commit the transaction. This will create lock files for all queued reference updates. If one reference could not be locked, the transaction will be aborted.

commit
: Commit all reference updates queued for the transaction, ending the transaction.

abort
: Abort the transaction, releasing all locks if the transaction is in prepared state.

If all <ref>s can be locked with matching <oldvalue>s simultaneously, all modifications are performed. Otherwise, no modifications are performed. Note that while each individual <ref> is updated or deleted

atomically, a concurrent reader may still see a subset of the modifications.

LOGGING UPDATES

If config parameter "core.logAllRefUpdates" is true and the ref is one under "refs/heads/", "refs/remotes/", "refs/notes/", or a pseudoref like HEAD or ORIG_HEAD; or the file "$GIT_DIR/logs/<ref>" exists then **git update−ref** will append a line to the log file "$GIT_DIR/logs/<ref>" (dereferencing all symbolic refs before creating the log name) describing the change in ref value. Log lines are formatted as:

oldsha1 SP newsha1 SP committer LF

Where "oldsha1" is the 40 character hexadecimal value previously stored in <ref>, "newsha1" is the 40 character hexadecimal value of <newvalue> and "committer" is the committer’s name, email address and date in the standard Git committer ident format.

Optionally with −m:

oldsha1 SP newsha1 SP committer TAB message LF

Where all fields are as described above and "message" is the value supplied to the −m option.

An update will fail (without changing <ref>) if the current user is unable to create a new log file, append to the existing log file or does not have committer information available.

GIT

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