NAME

gitremote-helpers - Helper programs to interact with remote repositories

SYNOPSIS

git remote-<transport> <repository> [<URL>]

DESCRIPTION

Remote helper programs are normally not used directly by end users, but they are invoked by Git when it needs to interact with remote repositories Git does not support natively. A given helper will implement a subset of the capabilities documented here. When Git needs to interact with a repository using a remote helper, it spawns the helper as an independent process, sends commands to the helper's standard input, and expects results from the helper's standard output. Because a remote helper runs as an independent process from Git, there is no need to re-link Git to add a new helper, nor any need to link the helper with the implementation of Git.

Every helper must support the "capabilities" command, which Git uses to determine what other commands the helper will accept. Those other commands can be used to discover and update remote refs, transport objects between the object database and the remote repository, and update the local object store.

Git comes with a "curl" family of remote helpers, that handle various transport protocols, such as *git-remote-http*, *git-remote-https*, *git-remote-ftp* and *git-remote-ftps*. They implement the capabilities *fetch*, *option*, and *push*.

INVOCATION

Remote helper programs are invoked with one or (optionally) two arguments. The first argument specifies a remote repository as in Git; it is either the name of a configured remote or a URL. The second argument specifies a URL; it is usually of the form *<transport>://<address>*, but any arbitrary string is possible. The **GIT_DIR** environment variable is set up for the remote helper and can be used to determine where to store additional data or from which directory to invoke auxiliary Git commands.

When Git encounters a URL of the form *<transport>://<address>*, where *<transport>* is a protocol that it cannot handle natively, it automatically invokes *git remote-<transport>* with the full URL as the second argument. If such a URL is encountered directly on the command line, the first argument is the same as the second, and if it is encountered in a configured remote, the first argument is the name of that remote.

A URL of the form *<transport>::<address>* explicitly instructs Git to invoke *git remote-<transport>* with *<address>* as the second argument. If such a URL is encountered directly on the command line,

the first argument is <*address*>, and if it is encountered in a configured remote, the first argument is the name of that remote.

Additionally, when a configured remote has **remote.<name>.vcs** set to <*transport*>, Git explicitly invokes *git remote-*<*transport*> with <*name*> as the first argument. If set, the second argument is **remote.<name>.url**; otherwise, the second argument is omitted.

INPUT FORMAT

Git sends the remote helper a list of commands on standard input, one per line. The first command is always the *capabilities* command, in response to which the remote helper must print a list of the capabilities it supports (see below) followed by a blank line. The response to the capabilities command determines what commands Git uses in the remainder of the command stream.

The command stream is terminated by a blank line. In some cases (indicated in the documentation of the relevant commands), this blank line is followed by a payload in some other protocol (e.g., the pack protocol), while in others it indicates the end of input.

Capabilities

Each remote helper is expected to support only a subset of commands. The operations a helper supports are declared to Git in the response to the **capabilities** command (see COMMANDS, below).

In the following, we list all defined capabilities and for each we list which commands a helper with that capability must provide.

Capabilities for Pushing

connect

Can attempt to connect to *git receive-pack* (for pushing), *git upload-pack*, etc for communication using git's native packfile protocol. This requires a bidirectional, full-duplex connection.

Supported commands: *connect*.

stateless-connect

Experimental; for internal use only. Can attempt to connect to a remote server for communication using git's wire-protocol version 2. See the documentation for the stateless-connect command for more information.

Supported commands: *stateless-connect*.

push

Can discover remote refs and push local commits and the history leading up to them to new or existing remote refs.

Supported commands: *list for-push*, *push*.

export

Can discover remote refs and push specified objects from a fast-import stream to remote refs.

Supported commands: *list for-push*, *export*.

If a helper advertises *connect*, Git will use it if possible and fall back to another capability if the helper requests so when connecting (see the *connect* command under COMMANDS). When choosing between *push* and *export*, Git prefers *push*. Other frontends may have some other order of preference.

no-private-update

When using the *refspec* capability, git normally updates the private ref on successful push. This update is disabled when the remote-helper declares the capability *no-private-update*.

Capabilities for Fetching

connect

Can try to connect to *git upload-pack* (for fetching), *git receive-pack*, etc for communication using the Git's native packfile protocol. This requires a bidirectional, full-duplex connection.

Supported commands: *connect*.

stateless-connect

Experimental; for internal use only. Can attempt to connect to a remote server for communication using git's wire-protocol version 2. See the documentation for the stateless-connect command for more information.

Supported commands: *stateless-connect*.

fetch

Can discover remote refs and transfer objects reachable from them to the local object store.

Supported commands: *list*, *fetch*.

import
: Can discover remote refs and output objects reachable from them as a stream in fast-import format.

Supported commands: *list*, *import*.

check-connectivity
: Can guarantee that when a clone is requested, the received pack is self contained and is connected.

get
: Can use the *get* command to download a file from a given URI.

If a helper advertises *connect*, Git will use it if possible and fall back to another capability if the helper requests so when connecting (see the *connect* command under COMMANDS). When choosing between *fetch* and *import*, Git prefers *fetch*. Other frontends may have some other order of preference.

Miscellaneous capabilities

option
: For specifying settings like **verbosity** (how much output to write to stderr) and **depth** (how much history is wanted in the case of a shallow clone) that affect how other commands are carried out.

refspec <refspec>
: For remote helpers that implement *import* or *export*, this capability allows the refs to be constrained to a private namespace, instead of writing to refs/heads or refs/remotes directly. It is recommended that all importers providing the *import* capability use this. It's mandatory for *export*.

A helper advertising the capability **refspec refs/heads/*:refs/svn/origin/branches/*** is saying that, when it is asked to **import refs/heads/topic**, the stream it outputs will update the **refs/svn/origin/branches/topic** ref.

This capability can be advertised multiple times. The first applicable refspec takes precedence. The left-hand of refspecs advertised with this capability must cover all refs reported by the list command. If no *refspec* capability is advertised, there is an implied **refspec *:***.

When writing remote-helpers for decentralized version control systems, it is advised to keep a local copy of the repository to interact with, and to let the private namespace refs point to this local repository, while the refs/remotes namespace is used to track the remote repository.

bidi-import
This modifies the *import* capability. The fast-import commands *cat-blob* and *ls* can be used by remote-helpers to retrieve information about blobs and trees that already exist in fast-import's memory. This requires a channel from fast-import to the remote-helper. If it is advertised in addition to "import", Git establishes a pipe from fast-import to the remote-helper's stdin. It follows that Git and fast-import are both connected to the remote-helper's stdin. Because Git can send multiple commands to the remote-helper it is required that helpers that use *bidi-import* buffer all *import* commands of a batch before sending data to fast-import. This is to prevent mixing commands and fast-import responses on the helper's stdin.

export-marks <file>
This modifies the *export* capability, instructing Git to dump the internal marks table to <file> when complete. For details, read up on **--export-marks=<file>** in **git-fast-export**(1).

import-marks <file>
This modifies the *export* capability, instructing Git to load the marks specified in <file> before processing any input. For details, read up on **--import-marks=<file>** in **git-fast-export**(1).

signed-tags
This modifies the *export* capability, instructing Git to pass **--signed-tags=verbatim** to **git-fast-export**(1). In the absence of this capability, Git will use **--signed-tags=warn-strip**.

object-format
This indicates that the helper is able to interact with the remote side using an explicit hash algorithm extension.

COMMANDS

Commands are given by the caller on the helper's standard input, one per line.

capabilities
Lists the capabilities of the helper, one per line, ending with a blank line. Each capability may be preceded with *, which marks them mandatory for Git versions using the remote helper to understand. Any unknown mandatory capability is a fatal error.

Support for this command is mandatory.

list

Lists the refs, one per line, in the format "<value> <name> [<attr> ...]". The value may be a hex sha1 hash, "@<dest>" for a symref, ":<keyword> <value>" for a key-value pair, or "?" to indicate that the helper could not get the value of the ref. A space-separated list of attributes follows the name; unrecognized attributes are ignored. The list ends with a blank line.

See REF LIST ATTRIBUTES for a list of currently defined attributes. See REF LIST KEYWORDS for a list of currently defined keywords.

Supported if the helper has the "fetch" or "import" capability.

list for-push

Similar to *list*, except that it is used if and only if the caller wants to the resulting ref list to prepare push commands. A helper supporting both push and fetch can use this to distinguish for which operation the output of *list* is going to be used, possibly reducing the amount of work that needs to be performed.

Supported if the helper has the "push" or "export" capability.

option <name> <value>

Sets the transport helper option <name> to <value>. Outputs a single line containing one of *ok* (option successfully set), *unsupported* (option not recognized) or *error* <*msg*> (option <name> is supported but <value> is not valid for it). Options should be set before other commands, and may influence the behavior of those commands.

See OPTIONS for a list of currently defined options.

Supported if the helper has the "option" capability.

fetch <sha1> <name>

Fetches the given object, writing the necessary objects to the database. Fetch commands are sent in a batch, one per line, terminated with a blank line. Outputs a single blank line when all fetch commands in the same batch are complete. Only objects which were reported in the output of *list* with a sha1 may be fetched this way.

Optionally may output a *lock* <*file*> line indicating the full path of a file under **$GIT_DIR/objects/pack** which is keeping a pack until refs can be suitably updated. The path must end with **.keep**. This is a mechanism to name a <pack,idx,keep> tuple by giving only the keep

component. The kept pack will not be deleted by a concurrent repack, even though its objects may not be referenced until the fetch completes. The **.keep** file will be deleted at the conclusion of the fetch.

If option *check-connectivity* is requested, the helper must output *connectivity-ok* if the clone is self-contained and connected.

Supported if the helper has the "fetch" capability.

push +<src>:<dst>

Pushes the given local <src> commit or branch to the remote branch described by <dst>. A batch sequence of one or more *push* commands is terminated with a blank line (if there is only one reference to push, a single *push* command is followed by a blank line). For example, the following would be two batches of *push*, the first asking the remote-helper to push the local ref *master* to the remote ref *master* and the local **HEAD** to the remote *branch*, and the second asking to push ref *foo* to ref *bar* (forced update requested by the +).

```
push refs/heads/master:refs/heads/master
push HEAD:refs/heads/branch
\n
push +refs/heads/foo:refs/heads/bar
\n
```

Zero or more protocol options may be entered after the last *push* command, before the batch's terminating blank line.

When the push is complete, outputs one or more *ok <dst>* or *error <dst> <why>?* lines to indicate success or failure of each pushed ref. The status report output is terminated by a blank line. The option field <why> may be quoted in a C style string if it contains an LF.

Supported if the helper has the "push" capability.

import <name>

Produces a fast-import stream which imports the current value of the named ref. It may additionally import other refs as needed to construct the history efficiently. The script writes to a helper-specific private namespace. The value of the named ref should be written to a location in this namespace derived by applying the refspecs from the "refspec" capability to the name of the ref.

Especially useful for interoperability with a foreign versioning system.

Just like *push*, a batch sequence of one or more *import* is terminated with a blank line. For each batch of *import*, the remote helper should produce a fast-import stream terminated by a *done* command.

Note that if the *bidi-import* capability is used the complete batch sequence has to be buffered before starting to send data to fast-import to prevent mixing of commands and fast-import responses on the helper's stdin.

Supported if the helper has the "import" capability.

export

Instructs the remote helper that any subsequent input is part of a fast-import stream (generated by *git fast-export*) containing objects which should be pushed to the remote.

Especially useful for interoperability with a foreign versioning system.

The *export-marks* and *import-marks* capabilities, if specified, affect this command in so far as they are passed on to *git fast-export*, which then will load/store a table of marks for local objects. This can be used to implement for incremental operations.

Supported if the helper has the "export" capability.

connect <service>

Connects to given service. Standard input and standard output of helper are connected to specified service (git prefix is included in service name so e.g. fetching uses *git-upload-pack* as service) on remote side. Valid replies to this command are empty line (connection established), *fallback* (no smart transport support, fall back to dumb transports) and just exiting with error message printed (can't connect, don't bother trying to fall back). After line feed terminating the positive (empty) response, the output of service starts. After the connection ends, the remote helper exits.

Supported if the helper has the "connect" capability.

stateless-connect <service>

Experimental; for internal use only. Connects to the given remote service for communication using git's wire-protocol version 2. Valid replies to this command are empty line (connection established), *fallback* (no smart transport support, fall back to dumb transports) and just exiting with error message printed (can't connect, don't bother trying to fall back). After line feed terminating the positive (empty) response, the output of the service starts. Messages (both request and response) must consist of zero or more PKT-LINEs, terminating in a flush packet. Response messages will then have a response end packet after the flush packet to indicate the end of a

response. The client must not expect the server to store any state in between request-response pairs. After the connection ends, the remote helper exits.

Supported if the helper has the "stateless-connect" capability.

get <uri> <path>

Downloads the file from the given **<uri>** to the given **<path>**. If **<path>.temp** exists, then Git assumes that the **.temp** file is a partial download from a previous attempt and will resume the download from that position.

If a fatal error occurs, the program writes the error message to stderr and exits. The caller should expect that a suitable error message has been printed if the child closes the connection without completing a valid response for the current command.

Additional commands may be supported, as may be determined from capabilities reported by the helper.

REF LIST ATTRIBUTES

The *list* command produces a list of refs in which each ref may be followed by a list of attributes. The following ref list attributes are defined.

unchanged

This ref is unchanged since the last import or fetch, although the helper cannot necessarily determine what value that produced.

REF LIST KEYWORDS

The *list* command may produce a list of key-value pairs. The following keys are defined.

object-format

The refs are using the given hash algorithm. This keyword is only used if the server and client both support the object-format extension.

OPTIONS

The following options are defined and (under suitable circumstances) set by Git if the remote helper has the *option* capability.

option verbosity <n>

Changes the verbosity of messages displayed by the helper. A value of 0 for <n> means that processes operate quietly, and the helper produces only error output. 1 is the default level of verbosity, and higher values of <n> correspond to the number of -v flags passed on the command

line.

option progress {*true*|*false*}
Enables (or disables) progress messages displayed by the transport helper during a command.

option depth <depth>
Deepens the history of a shallow repository.

’option deepen-since <timestamp>
Deepens the history of a shallow repository based on time.

’option deepen-not <ref>
Deepens the history of a shallow repository excluding ref. Multiple options add up.

option deepen-relative {’true|*false*}
Deepens the history of a shallow repository relative to current boundary. Only valid when used with "option depth".

option followtags {*true*|*false*}
If enabled the helper should automatically fetch annotated tag objects if the object the tag points at was transferred during the fetch command. If the tag is not fetched by the helper a second fetch command will usually be sent to ask for the tag specifically. Some helpers may be able to use this option to avoid a second network connection.

option dry-run {*true*|*false*}: If true, pretend the operation completed successfully, but don’t actually change any repository data. For most helpers this only applies to the *push*, if supported.

option servpath <c-style-quoted-path>
Sets service path (--upload-pack, --receive-pack etc.) for next connect. Remote helper may support this option, but must not rely on this option being set before connect request occurs.

option check-connectivity {*true*|*false*}
Request the helper to check connectivity of a clone.

option force {*true*|*false*}
Request the helper to perform a force update. Defaults to *false*.

option cloning {*true*|*false*}
Notify the helper this is a clone request (i.e. the current repository is guaranteed empty).

option update-shallow {*true*|*false*}
Allow to extend .git/shallow if the new refs require it.

option pushcert {*true*|*false*}
GPG sign pushes.

’option push-option <string>
Transmit <string> as a push option. As the push option must not contain LF or NUL characters, the string is not encoded.

option from-promisor {*true*|*false*}
Indicate that these objects are being fetched from a promisor.

option no-dependents {*true*|*false*}
Indicate that only the objects wanted need to be fetched, not their dependents.

option atomic {*true*|*false*}
When pushing, request the remote server to update refs in a single atomic transaction. If successful, all refs will be updated, or none will. If the remote side does not support this capability, the push will fail.

option object-format {*true*|algorithm}
If *true*, indicate that the caller wants hash algorithm information to be passed back from the remote. This mode is used when fetching refs.

If set to an algorithm, indicate that the caller wants to interact with the remote side using that algorithm.

SEE ALSO

git-remote(1)

git-remote-ext(1)

git-remote-fd(1)

git-fast-import(1)

GIT

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