NAME

gitprotocol-capabilities - Protocol v0 and v1 capabilities

SYNOPSIS

<over-the-wire-protocol>

DESCRIPTION

Note

this document describes capabilities for versions 0 and 1 of the pack protocol. For version 2, please refer to the **gitprotocol-v2**(5) doc.

Servers SHOULD support all capabilities defined in this document.

On the very first line of the initial server response of either receive-pack and upload-pack the first reference is followed by a NUL byte and then a list of space delimited server capabilities. These allow the server to declare what it can and cannot support to the client.

Client will then send a space separated list of capabilities it wants to be in effect. The client MUST NOT ask for capabilities the server did not say it supports.

Server MUST diagnose and abort if capabilities it does not understand was sent. Server MUST NOT ignore capabilities that client requested and server advertised. As a consequence of these rules, server MUST NOT advertise capabilities it does not understand.

The *atomic*, *report-status*, *report-status-v2*, *delete-refs*, *quiet*, and *push-cert* capabilities are sent and recognized by the receive-pack (push to server) process.

The *ofs-delta* and *side-band-64k* capabilities are sent and recognized by both upload-pack and receive-pack protocols. The *agent* and *session-id* capabilities may optionally be sent in both protocols.

All other capabilities are only recognized by the upload-pack (fetch from server) process.

MULTI_ACK

The *multi_ack* capability allows the server to return "ACK obj-id continue" as soon as it finds a commit that it can use as a common base, between the client’s wants and the client’s have set.

By sending this early, the server can potentially head off the client from walking any further down that particular branch of the client’s repository history. The client may still need to walk down other

branches, sending have lines for those, until the server has a complete cut across the DAG, or the client has said "done".

Without multi_ack, a client sends have lines in --date-order until the server has found a common base. That means the client will send have lines that are already known by the server to be common, because they overlap in time with another branch that the server hasn't found a common base on yet.

For example suppose the client has commits in caps that the server doesn't and the server has commits in lower case that the client doesn't, as in the following diagram:

```
  +---- u ---------------------- x
 /             +----- y
/             /
a -- b -- c -- d -- E -- F
  \
   +--- Q -- R -- S
```

If the client wants x,y and starts out by saying have F,S, the server doesn't know what F,S is. Eventually the client says "have d" and the server sends "ACK d continue" to let the client know to stop walking down that line (so don't send c-b-a), but it's not done yet, it needs a base for x. The client keeps going with S-R-Q, until a gets reached, at which point the server has a clear base and it all ends.

Without multi_ack the client would have sent that c-b-a chain anyway, interleaved with S-R-Q.

MULTI_ACK_DETAILED

This is an extension of multi_ack that permits client to better understand the server's in-memory state. See **gitprotocol-pack**(5), section "Packfile Negotiation" for more information.

NO-DONE

This capability should only be used with the smart HTTP protocol. If multi_ack_detailed and no-done are both present, then the sender is free to immediately send a pack following its first "ACK obj-id ready" message.

Without no-done in the smart HTTP protocol, the server session would end and the client has to make another trip to send "done" before the server can send the pack. no-done removes the last round and thus slightly reduces latency.

THIN-PACK

A thin pack is one with deltas which reference base objects not contained within the pack (but are known to exist at the receiving end). This can reduce the network traffic significantly, but it requires

the receiving end to know how to "thicken" these packs by adding the missing bases to the pack.

The upload-pack server advertises *thin-pack* when it can generate and send a thin pack. A client requests the *thin-pack* capability when it understands how to "thicken" it, notifying the server that it can receive such a pack. A client MUST NOT request the *thin-pack* capability if it cannot turn a thin pack into a self-contained pack.

Receive-pack, on the other hand, is assumed by default to be able to handle thin packs, but can ask the client not to use the feature by advertising the *no-thin* capability. A client MUST NOT send a thin pack if the server advertises the *no-thin* capability.

The reasons for this asymmetry are historical. The receive-pack program did not exist until after the invention of thin packs, so historically the reference implementation of receive-pack always understood thin packs. Adding *no-thin* later allowed receive-pack to disable the feature in a backwards-compatible manner.

SIDE-BAND, SIDE-BAND-64K

This capability means that server can send, and client understand multiplexed progress reports and error info interleaved with the packfile itself.

These two options are mutually exclusive. A modern client always favors *side-band-64k*.

Either mode indicates that the packfile data will be streamed broken up into packets of up to either 1000 bytes in the case of *side_band*, or 65520 bytes in the case of *side_band_64k*. Each packet is made up of a leading 4-byte pkt-line length of how much data is in the packet, followed by a 1-byte stream code, followed by the actual data.

The stream code can be one of:

1 - pack data
2 - progress messages
3 - fatal error message just before stream aborts

The "side-band-64k" capability came about as a way for newer clients that can handle much larger packets to request packets that are actually crammed nearly full, while maintaining backward compatibility for the older clients.

Further, with side-band and its up to 1000-byte messages, it’s actually 999 bytes of payload and 1 byte for the stream code. With side-band-64k, same deal, you have up to 65519 bytes of data and 1 byte for the stream code.

The client MUST send only maximum of one of "side-band" and "side- band-64k". Server MUST diagnose it as an error if client requests both.

OFS-DELTA

Server can send, and client understand PACKv2 with delta referring to its base by position in pack rather than by an obj-id. That is, they can send/read OBJ_OFS_DELTA (aka type 6) in a packfile.

AGENT

The server may optionally send a capability of the form **agent=X** to notify the client that the server is running version **X**. The client may optionally return its own agent string by responding with an **agent=Y** capability (but it MUST NOT do so if the server did not mention the agent capability). The **X** and **Y** strings may contain any printable ASCII characters except space (i.e., the byte range 32 < x < 127), and are typically of the form "package/version" (e.g., "git/1.8.3.1"). The agent strings are purely informative for statistics and debugging purposes, and MUST NOT be used to programmatically assume the presence or absence of particular features.

OBJECT-FORMAT

This capability, which takes a hash algorithm as an argument, indicates that the server supports the given hash algorithms. It may be sent multiple times; if so, the first one given is the one used in the ref advertisement.

When provided by the client, this indicates that it intends to use the given hash algorithm to communicate. The algorithm provided must be one that the server supports.

If this capability is not provided, it is assumed that the only supported algorithm is SHA-1.

SYMREF

This parameterized capability is used to inform the receiver which symbolic ref points to which ref; for example, "symref=HEAD:refs/heads/master" tells the receiver that HEAD points to master. This capability can be repeated to represent multiple symrefs.

Servers SHOULD include this capability for the HEAD symref if it is one of the refs being sent.

Clients MAY use the parameters from this capability to select the proper initial branch when cloning a repository.

SHALLOW

This capability adds "deepen", "shallow" and "unshallow" commands to the fetch-pack/upload-pack protocol so clients can request shallow clones.

DEEPEN-SINCE

This capability adds "deepen-since" command to fetch-pack/upload-pack protocol so the client can request shallow clones that are cut at a specific time, instead of depth. Internally it’s equivalent of doing "rev-list --max-age=<timestamp>" on the server side. "deepen-since" cannot be used with "deepen".

DEEPEN-NOT

This capability adds "deepen-not" command to fetch-pack/upload-pack protocol so the client can request shallow clones that are cut at a specific revision, instead of depth. Internally it’s equivalent of doing "rev-list --not <rev>" on the server side. "deepen-not" cannot be used with "deepen", but can be used with "deepen-since".

DEEPEN-RELATIVE

If this capability is requested by the client, the semantics of "deepen" command is changed. The "depth" argument is the depth from the current shallow boundary, instead of the depth from remote refs.

NO-PROGRESS

The client was started with "git clone -q" or something, and doesn’t want that side band 2. Basically the client just says "I do not wish to receive stream 2 on sideband, so do not send it to me, and if you did, I will drop it on the floor anyway". However, the sideband channel 3 is still used for error responses.

INCLUDE-TAG

The *include-tag* capability is about sending annotated tags if we are sending objects they point to. If we pack an object to the client, and a tag object points exactly at that object, we pack the tag object too. In general this allows a client to get all new annotated tags when it fetches a branch, in a single network connection.

Clients MAY always send include-tag, hardcoding it into a request when the server advertises this capability. The decision for a client to request include-tag only has to do with the client’s desires for tag data, whether or not a server had advertised objects in the refs/tags/* namespace.

Servers MUST pack the tags if their referrant is packed and the client has requested include-tags.

Clients MUST be prepared for the case where a server has ignored include-tag and has not actually sent tags in the pack. In such cases the client SHOULD issue a subsequent fetch to acquire the tags that include-tag would have otherwise given the client.

The server SHOULD send include-tag, if it supports it, regardless of whether or not there are tags available.

REPORT-STATUS

The receive-pack process can receive a *report-status* capability, which tells it that the client wants a report of what happened after a packfile upload and reference update. If the pushing client requests this capability, after unpacking and updating references the server will respond with whether the packfile unpacked successfully and if each reference was updated successfully. If any of those were not successful, it will send back an error message. See **gitprotocol-pack**(5) for example messages.

REPORT-STATUS-V2

Capability *report-status-v2* extends capability *report-status* by adding new "option" directives in order to support reference rewritten by the "proc-receive" hook. The "proc-receive" hook may handle a command for a pseudo-reference which may create or update a reference with different name, new-oid, and old-oid. While the capability *report-status* cannot report for such case. See **gitprotocol-pack**(5) for details.

DELETE-REFS

If the server sends back the *delete-refs* capability, it means that it is capable of accepting a zero-id value as the target value of a reference update. It is not sent back by the client, it simply informs the client that it can be sent zero-id values to delete references.

QUIET

If the receive-pack server advertises the *quiet* capability, it is capable of silencing human-readable progress output which otherwise may be shown when processing the received pack. A send-pack client should respond with the *quiet* capability to suppress server-side progress reporting if the local progress reporting is also being suppressed (e.g., via **push -q**, or if stderr does not go to a tty).

ATOMIC

If the server sends the *atomic* capability it is capable of accepting atomic pushes. If the pushing client requests this capability, the server will update the refs in one atomic transaction. Either all refs are updated or none.

PUSH-OPTIONS

If the server sends the *push-options* capability it is able to accept push options after the update commands have been sent, but before the packfile is streamed. If the pushing client requests this capability, the server will pass the options to the pre- and post- receive hooks that process this push request.

ALLOW-TIP-SHA1-IN-WANT

If the upload-pack server advertises this capability, fetch-pack may send "want" lines with object names that exist at the server but are not advertised by upload-pack. For historical reasons, the name of this capability contains "sha1". Object names are always given using the object format negotiated

through the *object-format* capability.

ALLOW-REACHABLE-SHA1-IN-WANT

If the upload-pack server advertises this capability, fetch-pack may send "want" lines with object names that exist at the server but are not advertised by upload-pack. For historical reasons, the name of this capability contains "sha1". Object names are always given using the object format negotiated through the *object-format* capability.

PUSH-CERT=<NONCE>

The receive-pack server that advertises this capability is willing to accept a signed push certificate, and asks the <nonce> to be included in the push certificate. A send-pack client MUST NOT send a push-cert packet unless the receive-pack server advertises this capability.

FILTER

If the upload-pack server advertises the *filter* capability, fetch-pack may send "filter" commands to request a partial clone or partial fetch and request that the server omit various objects from the packfile.

SESSION-ID=<SESSION ID>

The server may advertise a session ID that can be used to identify this process across multiple requests. The client may advertise its own session ID back to the server as well.

Session IDs should be unique to a given process. They must fit within a packet-line, and must not contain non-printable or whitespace characters. The current implementation uses trace2 session IDs (see **api-trace2**[1] for details), but this may change and users of the session ID should not rely on this fact.

GIT

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

NOTES

1. api-trace2
   git-htmldocs/technical/api-trace2.html