



Linux Ubuntu 26.04 Manual Pages on command 'git-credential.1'

\$ man git-credential.1

GIT-CREDENTIAL(1)

Git Manual

GIT-CREDENTIAL(1)

NAME

git-credential - Retrieve and store user credentials

SYNOPSIS

'git credential' (fill|approve|reject|capability)

DESCRIPTION

Git has an internal interface for storing and retrieving credentials from system-specific helpers, as well as prompting the user for usernames and passwords. The git-credential command exposes this interface to scripts which may want to retrieve, store, or prompt for credentials in the same manner as Git. The design of this scriptable interface models the internal C API; see credential.h for more background on the concepts.

git-credential takes an "action" option on the command-line (one of fill, approve, or reject) and reads a credential description on stdin (see INPUT/OUTPUT FORMAT).

If the action is fill, git-credential will attempt to add "username" and "password" attributes to the description by reading config files, by contacting any configured credential helpers, or by prompting the user. The username and password attributes of the credential description are then printed to stdout together with the attributes already provided.

If the action is approve, git-credential will send the description to any configured credential helpers, which may store the credential for later use.

If the action is reject, git-credential will send the description to any configured credential helpers, which may erase any stored credentials matching the description.

If the action is capability, git-credential will announce any capabilities it supports to

standard output.

If the action is approve or reject, no output should be emitted.

TYPICAL USE OF GIT CREDENTIAL

An application using git-credential will typically use git credential following these steps:

1. Generate a credential description based on the context.

For example, if we want a password for `https://example.com/foo.git`, we might generate the following credential description (don't forget the blank line at the end; it tells git credential that the application finished feeding all the information it has):

```
protocol=https  
  
host=example.com  
  
path=foo.git
```

2. Ask git-credential to give us a username and password for this description. This is done by running `git credential fill`, feeding the description from step (1) to its standard input. The complete credential description (including the credential per se, i.e. the login and password) will be produced on standard output, like:

```
protocol=https  
  
host=example.com  
  
username=bob  
  
password=secr3t
```

In most cases, this means the attributes given in the input will be repeated in the output, but Git may also modify the credential description, for example by removing the path attribute when the protocol is HTTP(s) and `credential.useHttpPath` is false.

If the git credential knew about the password, this step may not have involved the user actually typing this password (the user may have typed a password to unlock the keychain instead, or no user interaction was done if the keychain was already unlocked) before it returned `password=secr3t`.

3. Use the credential (e.g., access the URL with the username and password from step (2)), and see if it's accepted.
4. Report on the success or failure of the password. If the credential allowed the operation to complete successfully, then it can be marked with an "approve" action to tell git credential to reuse it in its next invocation. If the credential was rejected during the operation, use the "reject" action so that git credential will ask for a new password in its next invocation. In either case, git credential should be fed with the

credential description obtained from step (2) (which also contains the fields provided in step (1)).

INPUT/OUTPUT FORMAT

git credential reads and/or writes (depending on the action used) credential information in its standard input/output. This information can correspond either to keys for which git credential will obtain the login information (e.g. host, protocol, path), or to the actual credential data to be obtained (username/password).

The credential is split into a set of named attributes, with one attribute per line. Each attribute is specified by a key-value pair, separated by an = (equals) sign, followed by a newline.

The key may contain any bytes except =, newline, or NUL. The value may contain any bytes except newline or NUL. A line, including the trailing newline, may not exceed 65535 bytes in order to allow implementations to parse efficiently.

Attributes with keys that end with C-style array brackets [] can have multiple values. Each instance of a multi-valued attribute forms an ordered list of values - the order of the repeated attributes defines the order of the values. An empty multi-valued attribute (key[]=) acts to clear any previous entries and reset the list.

In all cases, all bytes are treated as-is (i.e., there is no quoting, and one cannot transmit a value with newline or NUL in it). The list of attributes is terminated by a blank line or end-of-file.

Git understands the following attributes:

protocol

The protocol over which the credential will be used (e.g., https).

host

The remote hostname for a network credential. This includes the port number if one was specified (e.g., "example.com:8088").

path

The path with which the credential will be used. E.g., for accessing a remote https repository, this will be the repository's path on the server.

username

The credential's username, if we already have one (e.g., from a URL, the configuration, the user, or from a previously run helper).

password

The credential's password, if we are asking it to be stored.

password_expiry_utc

Generated passwords such as an OAuth access token may have an expiry date. When reading credentials from helpers, git credential fill ignores expired passwords. Represented as Unix time UTC, seconds since 1970.

oauth_refresh_token

An OAuth refresh token may accompany a password that is an OAuth access token. Helpers must treat this attribute as confidential like the password attribute. Git itself has no special behaviour for this attribute.

url

When this special attribute is read by git credential, the value is parsed as a URL and treated as if its constituent parts were read (e.g., url=https://example.com would behave as if protocol=https and host=example.com had been provided). This can help callers avoid parsing URLs themselves.

Note that specifying a protocol is mandatory and if the URL doesn't specify a hostname (e.g., "cert:///path/to/file") the credential will contain a hostname attribute whose value is an empty string.

Components which are missing from the URL (e.g., there is no username in the example above) will be left unset.

authtype

This indicates that the authentication scheme in question should be used. Common values for HTTP and HTTPS include basic, bearer, and digest, although the latter is insecure and should not be used. If credential is used, this may be set to an arbitrary string suitable for the protocol in question (usually HTTP).

This value should not be sent unless the appropriate capability (see below) is provided on input.

credential

The pre-encoded credential, suitable for the protocol in question (usually HTTP). If this key is sent, authtype is mandatory, and username and password are not used. For HTTP, Git concatenates the authtype value and this value with a single space to determine the Authorization header.

This value should not be sent unless the appropriate capability (see below) is provided on input.

ephemeral

This boolean value indicates, if true, that the value in the credential field should not be saved by the credential helper because its usefulness is limited in time. For example, an HTTP Digest credential value is computed using a nonce and reusing it will not result in successful authentication. This may also be used for situations with short duration (e.g., 24-hour) credentials. The default value is false.

The credential helper will still be invoked with store or erase so that it can determine whether the operation was successful.

This value should not be sent unless the appropriate capability (see below) is provided on input.

state[]

This value provides an opaque state that will be passed back to this helper if it is called again. Each different credential helper may specify this once. The value should include a prefix unique to the credential helper and should ignore values that don't match its prefix.

This value should not be sent unless the appropriate capability (see below) is provided on input.

continue

This is a boolean value, which, if enabled, indicates that this authentication is a non-final part of a multistage authentication step. This is common in protocols such as NTLM and Kerberos, where two rounds of client authentication are required, and setting this flag allows the credential helper to implement the multistage authentication step.

This flag should only be sent if a further stage is required; that is, if another round of authentication is expected.

This value should not be sent unless the appropriate capability (see below) is provided on input. This attribute is one-way from a credential helper to pass information to Git (or other programs invoking git credential).

wwwauth[]

When an HTTP response is received by Git that includes one or more WWW-Authenticate authentication headers, these will be passed by Git to credential helpers.

Each WWW-Authenticate header value is passed as a multi-valued attribute `wwwauth[]`, where the order of the attributes is the same as they appear in the HTTP response. This attribute is one-way from Git to pass additional information to credential helpers.

capability[]

This signals that Git, or the helper, as appropriate, supports the capability in question. This can be used to provide better, more specific data as part of the protocol. A capability[] directive must precede any value depending on it and these directives should be the first item announced in the protocol.

There are two currently supported capabilities. The first is authtype, which indicates that the authtype, credential, and ephemeral values are understood. The second is state, which indicates that the state[] and continue values are understood.

It is not obligatory to use the additional features just because the capability is supported, but they should not be provided without the capability.

Unrecognised attributes and capabilities are silently discarded.

CAPABILITY INPUT/OUTPUT FORMAT

For git credential capability, the format is slightly different. First, a version 0 announcement is made to indicate the current version of the protocol, and then each capability is announced with a line like capability authtype. Credential helpers may also implement this format, again with the capability argument. Additional lines may be added in the future; callers should ignore lines which they don't understand.

Because this is a new part of the credential helper protocol, older versions of Git, as well as some credential helpers, may not support it. If a non-zero exit status is received, or if the first line doesn't start with the word version and a space, callers should assume that no capabilities are supported.

The intention of this format is to differentiate it from the credential output in an unambiguous way. It is possible to use very simple credential helpers (e.g., inline shell scripts) which always produce identical output. Using a distinct format allows users to continue to use this syntax without having to worry about correctly implementing capability advertisements or accidentally confusing callers querying for capabilities.

GIT

Part of the git(1) suite

Git 2.53.0

03/02/2026

GIT-CREDENTIAL(1)