NAME

gitcredentials - Providing usernames and passwords to Git

SYNOPSIS

```
git config credential.https://example.com.username myusername
git config credential.helper "$helper $options"
```

DESCRIPTION

Git will sometimes need credentials from the user in order to perform operations; for example, it may need to ask for a username and password in order to access a remote repository over HTTP. Some remotes accept a personal access token or OAuth access token as a password. This manual describes the mechanisms Git uses to request these credentials, as well as some features to avoid inputting these credentials repeatedly.

REQUESTING CREDENTIALS

Without any credential helpers defined, Git will try the following strategies to ask the user for usernames and passwords:

1. the **GIT_ASKPASS** environment variable is set, the program specified by the variable is invoked. A suitable prompt is provided to the program on the command line, and the user’s input is read from its standard output.

2. if the **core.askPass** configuration variable is set, its value is used as above.

3. if the **SSH_ASKPASS** environment variable is set, its value is used as above.

4. the user is prompted on the terminal.

AVOIDING REPETITION

It can be cumbersome to input the same credentials over and over. Git provides two methods to reduce this annoyance:

1. configuration of usernames for a given authentication context.

2.
helpers to cache or store passwords, or to interact with a system password wallet or keychain.

The first is simple and appropriate if you do not have secure storage available for a password. It is generally configured by adding this to your config:

```
[credential "https://example.com"]
    username = me
```

Credential helpers, on the other hand, are external programs from which Git can request both usernames and passwords; they typically interface with secure storage provided by the OS or other programs. Alternatively, a credential-generating helper might generate credentials for certain servers via some API.

To use a helper, you must first select one to use. Git currently includes the following helpers:

cache
Cache credentials in memory for a short period of time. See **git-credential-cache**(1) for details.

store
Store credentials indefinitely on disk. See **git-credential-store**(1) for details.

You may also have third-party helpers installed; search for **credential-*** in the output of **git help -a**, and consult the documentation of individual helpers. Once you have selected a helper, you can tell Git to use it by putting its name into the credential.helper variable.

1.
a helper.

```
$ git help -a | grep credential-
credential-foo
```

2.
its description.

```
$ git help credential-foo
```

3.
Git to use it.

```
$ git config --global credential.helper foo
```

Available helpers

The community maintains a comprehensive list of Git credential helpers at **https://git-scm.com/doc/credential-helpers**.

OAuth

An alternative to inputting passwords or personal access tokens is to use an OAuth credential helper. Initial authentication opens a browser window to the host. Subsequent authentication happens in the background. Many popular Git hosts support OAuth.

CREDENTIAL CONTEXTS

Git considers each credential to have a context defined by a URL. This context is used to look up context-specific configuration, and is passed to any helpers, which may use it as an index into secure storage.

For instance, imagine we are accessing **https://example.com/foo.git**. When Git looks into a config file to see if a section matches this context, it will consider the two a match if the context is a more-specific subset of the pattern in the config file. For example, if you have this in your config file:

```
[credential "https://example.com"]
    username = foo
```

then we will match: both protocols are the same, both hosts are the same, and the "pattern" URL does not care about the path component at all. However, this context would not match:

```
[credential "https://kernel.org"]
    username = foo
```

because the hostnames differ. Nor would it match **foo.example.com**; Git compares hostnames exactly, without considering whether two hosts are part of the same domain. Likewise, a config entry for **http://example.com** would not match: Git compares the protocols exactly. However, you may use wildcards in the domain name and other pattern matching techniques as with the **http.<URL>.*** options.

If the "pattern" URL does include a path component, then this too must match exactly: the context **https://example.com/bar/baz.git** will match a config entry for **https://example.com/bar/baz.git** (in addition to matching the config entry for **https://example.com**) but will not match a config entry for **https://example.com/bar**.

CONFIGURATION OPTIONS

Options for a credential context can be configured either in **credential.*** (which applies to all credentials), or **credential.<URL>.***, where <URL> matches the context as described above.

The following options are available in either location:

helper

The name of an external credential helper, and any associated options. If the helper name is not an absolute path, then the string **git credential-** is prepended. The resulting string is executed by the shell (so, for example, setting this to **foo --option=bar** will execute **git credential-foo --option=bar** via the shell. See the manual of specific helpers for examples of their use.

If there are multiple instances of the **credential.helper** configuration variable, each helper will be tried in turn, and may provide a username, password, or nothing. Once Git has acquired both a username and a non-expired password, no more helpers will be tried.

If **credential.helper** is configured to the empty string, this resets the helper list to empty (so you may override a helper set by a lower-priority config file by configuring the empty-string helper, followed by whatever set of helpers you would like).

username

A default username, if one is not provided in the URL.

useHttpPath

By default, Git does not consider the "path" component of an http URL to be worth matching via external helpers. This means that a credential stored for **https://example.com/foo.git** will also be used for **https://example.com/bar.git**. If you do want to distinguish these cases, set this option to **true**.

CUSTOM HELPERS

You can write your own custom helpers to interface with any system in which you keep credentials.

Credential helpers are programs executed by Git to fetch or save credentials from and to long-term storage (where "long-term" is simply longer than a single Git process; e.g., credentials may be stored in-memory for a few minutes, or indefinitely on disk).

Each helper is specified by a single string in the configuration variable **credential.helper** (and others, see **git-config**(1)). The string is transformed by Git into a command to be executed using these rules:

1. the helper string begins with "!", it is considered a shell snippet, and everything after the "!" becomes the command.

2. if the helper string begins with an absolute path, the verbatim helper string becomes the command.

3. the string "git credential-" is prepended to the helper string, and the result becomes the command.

The resulting command then has an "operation" argument appended to it (see below for details), and the result is executed by the shell.

Here are some example specifications:

```
# run "git credential-foo"
[credential]
	helper = foo

# same as above, but pass an argument to the helper
[credential]
	helper = "foo --bar=baz"

# the arguments are parsed by the shell, so use shell
# quoting if necessary
[credential]
	helper = "foo --bar=’whitespace arg’"

# you can also use an absolute path, which will not use the git wrapper
[credential]
	helper = "/path/to/my/helper --with-arguments"

# or you can specify your own shell snippet
[credential "https://example.com"]
	username = your_user
	helper = "!f() { test \"$1\" = get && echo \"password=$(cat $HOME/.secret)\"; }; f"
```

Generally speaking, rule (3) above is the simplest for users to specify. Authors of credential helpers should make an effort to assist their users by naming their program "git-credential-$NAME", and putting it in the **$PATH** or **$GIT_EXEC_PATH** during installation, which will allow a user to enable it with **git config credential.helper $NAME**.

When a helper is executed, it will have one "operation" argument appended to its command line, which is one of:

get
Return a matching credential, if any exists.

store
Store the credential, if applicable to the helper.

erase
Remove matching credentials, if any, from the helper's storage.

The details of the credential will be provided on the helper's stdin stream. The exact format is the same as the input/output format of the **git credential** plumbing command (see the section **INPUT/OUTPUT FORMAT** in **git-credential**(1) for a detailed specification).

For a **get** operation, the helper should produce a list of attributes on stdout in the same format (see **git-credential**(1) for common attributes). A helper is free to produce a subset, or even no values at all if it has nothing useful to provide. Any provided attributes will overwrite those already known about by Git's credential subsystem. Unrecognised attributes are silently discarded.

While it is possible to override all attributes, well behaving helpers should refrain from doing so for any attribute other than username and password.

If a helper outputs a **quit** attribute with a value of **true** or **1**, no further helpers will be consulted, nor will the user be prompted (if no credential has been provided, the operation will then fail).

Similarly, no more helpers will be consulted once both username and password had been provided.

For a **store** or **erase** operation, the helper's output is ignored.

If a helper fails to perform the requested operation or needs to notify the user of a potential issue, it may write to stderr.

If it does not support the requested operation (e.g., a read-only store or generator), it should silently ignore the request.

If a helper receives any other operation, it should silently ignore the request. This leaves room for future operations to be added (older helpers will just ignore the new requests).

GIT

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