10.8 Git Internals - Environment Variables

Environment Variables

Git always runs inside a `bash` shell, and uses a number of shell environment variables to determine how it behaves. Occasionally, it comes in handy to know what these are, and how they can be used to make Git behave the way you want it to. This isn't an exhaustive list of all the environment variables Git pays attention to, but we'll cover the most useful.

Global Behavior

Some of Git's general behavior as a computer program depends on environment variables.

`GIT_EXEC_PATH` determines where Git looks for its sub-programs (like `git-commit`, `git-diff`, and others). You can check the current setting by running `git --exec-path`.

`HOME` isn't usually considered customizable (too many other things depend on it), but it's where Git looks for the global configuration file. If you want a truly portable Git installation, complete with global configuration, you can override `HOME` in the portable Git's shell profile.

`PREFIX` is similar, but for the system-wide configuration. Git looks for this file at `$PREFIX/etc/gitconfig`.

`GIT_CONFIG_NOSYSTEM`, if set, disables the use of the system-wide configuration file. This is useful if your system config is interfering with your commands, but you don't have access to change or remove it.

`GIT_PAGER` controls the program used to display multi-page output on the command line. If this is unset, `PAGER` will be used as a fallback.

`GIT_EDITOR` is the editor Git will launch when the user needs to edit some text (a commit message, for example). If unset, `EDITOR` will be used.

Repository Locations

Git uses several environment variables to determine how it interfaces with the current repository.

`GIT_DIR` is the location of the `.git` folder. If this isn't specified, Git walks up the directory tree until it gets to ~ or `/`, looking for a `.git` directory at every step.

`GIT_CEILING_DIRECTORIES` controls the behavior of searching for a `.git` directory. If you access directories that are slow to load (such as those on a tape drive, or across a slow network connection), you may want to have Git stop trying earlier than it might otherwise, especially if Git is invoked when building your shell prompt.

`GIT_WORK_TREE` is the location of the root of the working directory for a non-bare repository. If `--git-dir` or `GIT_DIR` is specified but none of `--work-tree`, `GIT_WORK_TREE` or `core.worktree` is specified, the current working directory is regarded as the top level of your working tree.

`GIT_INDEX_FILE` is the path to the index file (non-bare repositories only).

`GIT_OBJECT_DIRECTORY` can be used to specify the location of the directory that usually resides at `.git/objects`.

`GIT_ALTERNATE_OBJECT_DIRECTORIES` is a colon-separated list (formatted like `/dir/one:/dir/two:…`) which tells Git where to check for objects if they aren't in `GIT_OBJECT_DIRECTORY`. If you happen to have a lot of projects with large files that have the exact same contents, this can be used to avoid storing too many copies of them.

Pathspecs

A "pathspec" refers to how you specify paths to things in Git, including the use of wildcards. These are used in the `.gitignore` file, but also on the command-line (`git add *.c`).

`GIT_GLOB_PATHSPECS` and **`GIT_NOGLOB_PATHSPECS`** control the default behavior of wildcards in pathspecs. If `GIT_GLOB_PATHSPECS` is set to 1, wildcard characters act as wildcards (which is the default); if `GIT_NOGLOB_PATHSPECS` is set to 1, wildcard characters only match themselves, meaning something like `*.c` would only match a file *named* "*.c", rather than any file whose name ends with `.c`. You can override this in individual cases by starting the pathspec with `:(glob)` or `:(literal)`, as in `:(glob)\*.c`.

`GIT_LITERAL_PATHSPECS` disables both of the above behaviors; no wildcard characters will work, and the override prefixes are disabled as well.

`GIT_ICASE_PATHSPECS` sets all pathspecs to work in a case-insensitive manner.

Committing

The final creation of a Git commit object is usually done by `git-commit-tree`, which uses these environment variables as its primary source of information, falling back to configuration values only if these aren't present.

`GIT_AUTHOR_NAME` is the human-readable name in the "author" field.

`GIT_AUTHOR_EMAIL` is the email for the "author" field.

`GIT_AUTHOR_DATE` is the timestamp used for the "author" field.

`GIT_COMMITTER_NAME` sets the human name for the "committer" field.

`GIT_COMMITTER_EMAIL` is the email address for the "committer" field.

`GIT_COMMITTER_DATE` is used for the timestamp in the "committer" field.

`EMAIL` is the fallback email address in case the `user.email` configuration value isn't set. If *this* isn't set, Git falls back to the system user and host names.

Networking

Git uses the `curl` library to do network operations over HTTP, so **`GIT_CURL_VERBOSE`** tells Git to emit all the messages generated by that library. This is similar to doing `curl -v` on the command line.

GIT_SSL_NO_VERIFY tells Git not to verify SSL certificates. This can sometimes be necessary if you're using a self-signed certificate to serve Git repositories over HTTPS, or you're in the middle of setting up a Git server but haven't installed a full certificate yet.

If the data rate of an HTTP operation is lower than **GIT_HTTP_LOW_SPEED_LIMIT** bytes per second for longer than **GIT_HTTP_LOW_SPEED_TIME** seconds, Git will abort that operation. These values override the `http.lowSpeedLimit` and `http.lowSpeedTime` configuration values.

GIT_HTTP_USER_AGENT sets the user-agent string used by Git when communicating over HTTP. The default is a value like `git/2.0.0`.

Diffing and Merging

GIT_DIFF_OPTS is a bit of a misnomer. The only valid values are `-u<n>` or `--unified=<n>`, which controls the number of context lines shown in a `git diff` command.

GIT_EXTERNAL_DIFF is used as an override for the `diff.external` configuration value. If it's set, Git will invoke this program when `git diff` is invoked.

GIT_DIFF_PATH_COUNTER and **GIT_DIFF_PATH_TOTAL** are useful from inside the program specified by `GIT_EXTERNAL_DIFF` or `diff.external`. The former represents which file in a series is being diffed (starting with 1), and the latter is the total number of files in the batch.

GIT_MERGE_VERBOSITY controls the output for the recursive merge strategy. The allowed values are as follows:

- 0 outputs nothing, except possibly a single error message.
- 1 shows only conflicts.
- 2 also shows file changes.
- 3 shows when files are skipped because they haven't changed.
- 4 shows all paths as they are processed.
- 5 and above show detailed debugging information.

The default value is 2.

Debugging

Want to *really* know what Git is up to? Git has a fairly complete set of traces embedded, and all you need to do is turn them on. The possible values of these variables are as follows:

- "true", "1", or "2" – the trace category is written to stderr.
- An absolute path starting with `/` – the trace output will be written to that file.

GIT_TRACE controls general traces, which don't fit into any specific category. This includes the expansion of aliases, and delegation to other sub-programs.

```
$ GIT_TRACE=true git lga
20:12:49.877982 git.c:554               trace: exec: 'git-lga'
20:12:49.878369 run-command.c:341       trace: run_command: 'git-lga'
20:12:49.879529 git.c:282               trace: alias expansion: lga => 'log' '--graph' '--pretty=oneline' '--abbrev-commit' '--decorate' '--all'
20:12:49.879885 git.c:349               trace: built-in: git 'log' '--graph' '--pretty=oneline' '--abbrev-commit' '--decorate' '--all'
20:12:49.899217 run-command.c:341       trace: run_command: 'less'
20:12:49.899675 run-command.c:192       trace: exec: 'less'
```

GIT_TRACE_PACK_ACCESS controls tracing of packfile access. The first field is the packfile being accessed, the second is the offset within that file:

```
$ GIT_TRACE_PACK_ACCESS=true git status
20:10:12.081397 sha1_file.c:2088        .git/objects/pack/pack-c3fa...291e.pack 12
20:10:12.081886 sha1_file.c:2088        .git/objects/pack/pack-c3fa...291e.pack 34662
20:10:12.082115 sha1_file.c:2088        .git/objects/pack/pack-c3fa...291e.pack 35175
# […]
20:10:12.087398 sha1_file.c:2088        .git/objects/pack/pack-e80e...e3d2.pack 56914983
20:10:12.087419 sha1_file.c:2088        .git/objects/pack/pack-e80e...e3d2.pack 14303666
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
```

GIT_TRACE_PACKET enables packet-level tracing for network operations.

```
$ GIT_TRACE_PACKET=true git ls-remote origin
20:15:14.867043 pkt-line.c:46           packet:          git< # service=git-upload-pack
20:15:14.867071 pkt-line.c:46           packet:          git< 0000
20:15:14.867079 pkt-line.c:46           packet:          git< 97b8860c071898d9e162678ea1035a8ced2f8b1f HEAD\0multi_ack thin-pack side-band side-band-6
20:15:14.867088 pkt-line.c:46           packet:          git< 0f20ae29889d61f2e93ae00fd34f1cdb53285702 refs/heads/ab/add-interactive-show-diff-func-na
20:15:14.867094 pkt-line.c:46           packet:          git< 36dc827bc9d17f80ed4f326de21247a5d1341fbc refs/heads/ah/doc-gitk-config
# […]
```

GIT_TRACE_PERFORMANCE controls logging of performance data. The output shows how long each particular `git` invocation takes.

```
$ GIT_TRACE_PERFORMANCE=true git gc
20:18:19.499676 trace.c:414             performance: 0.374835000 s: git command: 'git' 'pack-refs' '--all' '--prune'
20:18:19.845585 trace.c:414             performance: 0.343020000 s: git command: 'git' 'reflog' 'expire' '--all'
Counting objects: 170994, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (43413/43413), done.
Writing objects: 100% (170994/170994), done.
Total 170994 (delta 126176), reused 170524 (delta 125706)
20:18:23.567927 trace.c:414             performance: 3.715349000 s: git command: 'git' 'pack-objects' '--keep-true-parents' '--honor-pack-keep' '--nor
20:18:23.584728 trace.c:414             performance: 0.000910000 s: git command: 'git' 'prune-packed'
20:18:23.605218 trace.c:414             performance: 0.017972000 s: git command: 'git' 'update-server-info'
20:18:23.606342 trace.c:414             performance: 3.756312000 s: git command: 'git' 'repack' '-d' '-l' '-A' '--unpack-unreachable=2.weeks.ago'
Checking connectivity: 170994, done.
20:18:25.225424 trace.c:414             performance: 1.616423000 s: git command: 'git' 'prune' '--expire' '2.weeks.ago'
20:18:25.232403 trace.c:414             performance: 0.001051000 s: git command: 'git' 'rerere' 'gc'
20:18:25.233159 trace.c:414             performance: 6.112217000 s: git command: 'git' 'gc'
```

GIT_TRACE_SETUP shows information about what Git is discovering about the repository and environment it's interacting with.

```
$ GIT_TRACE_SETUP=true git status
20:19:47.086765 trace.c:315             setup: git_dir: .git
20:19:47.087184 trace.c:316             setup: worktree: /Users/ben/src/git
20:19:47.087191 trace.c:317             setup: cwd: /Users/ben/src/git
20:19:47.087194 trace.c:318             setup: prefix: (null)
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
```

Miscellaneous

`GIT_SSH`, if specified, is a program that is invoked instead of `ssh` when Git tries to connect to an SSH host. It is invoked like `$GIT_SSH [username@]host [-p <port>] <command>`. Note that this isn't the easiest way to customize how `ssh` is invoked; it won't support extra command-line parameters, so you'd have to write a wrapper script and set `GIT_SSH` to point to it. It's probably easier just to use the `~/.ssh/config` file for that.

`GIT_ASKPASS` is an override for the `core.askpass` configuration value. This is the program invoked whenever Git needs to ask the user for credentials, which can expect a text prompt as a command-line argument, and should return the answer on `stdout` (see Credential Storage for more on this subsystem).

`GIT_NAMESPACE` controls access to namespaced refs, and is equivalent to the `--namespace` flag. This is mostly useful on the server side, where you may want to store multiple forks of a single repository in one repository, only keeping the refs separate.

`GIT_FLUSH` can be used to force Git to use non-buffered I/O when writing incrementally to stdout. A value of 1 causes Git to flush more often, a value of 0 causes all output to be buffered. The default value (if this variable is not set) is to choose an appropriate buffering scheme depending on the activity and the output mode.

`GIT_REFLOG_ACTION` lets you specify the descriptive text written to the reflog. Here's an example:

```
$ GIT_REFLOG_ACTION="my action" git commit --allow-empty -m 'My message'
[master 9e3d55a] My message
$ git reflog -1
9e3d55a HEAD@{0}: my action: My message
```

prev | next