7.5 Git Tools - Searching

Searching

With just about any size codebase, you'll often need to find where a function is called or defined, or display the history of a method. Git provides a couple of useful tools for looking through the code and commits stored in its database quickly and easily. We'll go through a few of them.

Git Grep

Git ships with a command called `grep` that allows you to easily search through any committed tree, the working directory, or even the index for a string or regular expression. For the examples that follow, we'll search through the source code for Git itself.

By default, `git grep` will look through the files in your working directory. As a first variation, you can use either of the `-n` or `--line-number` options to print out the line numbers where Git has found matches:

```
$ git grep -n gmtime_r
compat/gmtime.c:3:#undef gmtime_r
compat/gmtime.c:8:      return git_gmtime_r(timep, &result);
compat/gmtime.c:11:struct tm *git_gmtime_r(const time_t *timep, struct tm *result)
compat/gmtime.c:16:     ret = gmtime_r(timep, result);
compat/mingw.c:826:struct tm *gmtime_r(const time_t *timep, struct tm *result)
compat/mingw.h:206:struct tm *gmtime_r(const time_t *timep, struct tm *result);
date.c:482:             if (gmtime_r(&now, &now_tm))
date.c:545:             if (gmtime_r(&time, tm)) {
date.c:758:             /* gmtime_r() in match_digit() may have clobbered it */
git-compat-util.h:1138:struct tm *git_gmtime_r(const time_t *, struct tm *);
git-compat-util.h:1140:#define gmtime_r git_gmtime_r
```

In addition to the basic search shown above, `git grep` supports a plethora of other interesting options.

For instance, instead of printing all of the matches, you can ask `git grep` to summarize the output by showing you only which files contained the search string and how many matches there were in each file with the `-c` or `--count` option:

```
$ git grep --count gmtime_r
compat/gmtime.c:4
compat/mingw.c:1
compat/mingw.h:1
date.c:3
git-compat-util.h:2
```

If you're interested in the *context* of a search string, you can display the enclosing method or function for each matching string with either of the `-p` or `--show-function` options:

```
$ git grep -p gmtime_r *.c
date.c=static int match_multi_number(timestamp_t num, char c, const char *date,
date.c:         if (gmtime_r(&now, &now_tm))
date.c=static int match_digit(const char *date, struct tm *tm, int *offset, int *tm_gmt)
date.c:         if (gmtime_r(&time, tm)) {
date.c=int parse_date_basic(const char *date, timestamp_t *timestamp, int *offset)
date.c:         /* gmtime_r() in match_digit() may have clobbered it */
```

As you can see, the `gmtime_r` routine is called from both the `match_multi_number` and `match_digit` functions in the `date.c` file (the third match displayed represents just the string appearing in a comment).

You can also search for complex combinations of strings with the `--and` flag, which ensures that multiple matches must occur in the same line of text. For instance, let's look for any lines that define a constant whose name contains *either* of the substrings "LINK" or "BUF_MAX", specifically in an older version of the Git

codebase represented by the tag `v1.8.0` (we'll throw in the `--break` and `--heading` options which help split up the output into a more readable format):

```
$ git grep --break --heading \
    -n -e '#define' --and \( -e LINK -e BUF_MAX \) v1.8.0
v1.8.0:builtin/index-pack.c
62:#define FLAG_LINK (1u<<20)

v1.8.0:cache.h
73:#define S_IFGITLINK  0160000
74:#define S_ISGITLINK(m)       (((m) & S_IFMT) == S_IFGITLINK)

v1.8.0:environment.c
54:#define OBJECT_CREATION_MODE OBJECT_CREATION_USES_HARDLINKS

v1.8.0:strbuf.c
326:#define STRBUF_MAXLINK (2*PATH_MAX)

v1.8.0:symlinks.c
53:#define FL_SYMLINK  (1 << 2)

v1.8.0:zlib.c
30:/* #define ZLIB_BUF_MAX ((uInt)-1) */
31:#define ZLIB_BUF_MAX ((uInt) 1024 * 1024 * 1024) /* 1GB */
```

The `git grep` command has a few advantages over normal searching commands like `grep` and `ack`. The first is that it's really fast, the second is that you can search through any tree in Git, not just the working directory. As we saw in the above example, we looked for terms in an older version of the Git source code, not the version that was currently checked out.

Git Log Searching

Perhaps you're looking not for *where* a term exists, but *when* it existed or was introduced. The `git log`command has a number of powerful tools for finding specific commits by the content of their messages or even the content of the diff they introduce.

If, for example, we want to find out when the `ZLIB_BUF_MAX` constant was originally introduced, we can use the `-S` option (colloquially referred to as the Git "pickaxe" option) to tell Git to show us only those commits that changed the number of occurrences of that string.

```
$ git log -S ZLIB_BUF_MAX --oneline
e01503b zlib: allow feeding more than 4GB in one go
ef49a7a zlib: zlib can only process 4GB at a time
```

If we look at the diff of those commits, we can see that in `ef49a7a` the constant was introduced and in `e01503b`it was modified.

If you need to be more specific, you can provide a regular expression to search for with the `-G` option.

Line Log Search

Another fairly advanced log search that is insanely useful is the line history search. Simply run `git log` with the `-L` option, and it will show you the history of a function or line of code in your codebase.

For example, if we wanted to see every change made to the function `git_deflate_bound` in the `zlib.c` file, we could run `git log -L :git_deflate_bound:zlib.c`. This will try to figure out what the bounds of that function are and then look through the history and show us every change that was made to the function as a series of patches back to when the function was first created.

```
$ git log -L :git_deflate_bound:zlib.c
commit ef49a7a0126d64359c974b4b3b71d7ad42ee3bca
Author: Junio C Hamano <gitster@pobox.com>
```

```
Date:   Fri Jun 10 11:52:15 2011 -0700

    zlib: zlib can only process 4GB at a time

diff --git a/zlib.c b/zlib.c
--- a/zlib.c
+++ b/zlib.c
@@ -85,5 +130,5 @@
-unsigned long git_deflate_bound(z_streamp strm, unsigned long size)
+unsigned long git_deflate_bound(git_zstream *strm, unsigned long size)
 {
-       return deflateBound(strm, size);
+       return deflateBound(&strm->z, size);
 }


commit 225a6f1068f71723a910e8565db4e252b3ca21fa
Author: Junio C Hamano <gitster@pobox.com>
Date:   Fri Jun 10 11:18:17 2011 -0700

    zlib: wrap deflateBound() too

diff --git a/zlib.c b/zlib.c
--- a/zlib.c
+++ b/zlib.c
@@ -81,0 +85,5 @@
+unsigned long git_deflate_bound(z_streamp strm, unsigned long size)
+{
+       return deflateBound(strm, size);
+}
+
```

If Git can't figure out how to match a function or method in your programming language, you can also provide it with a regular expression (or *regex*). For example, this would have done the same thing as the example above: `git log -L '/unsigned long git_deflate_bound/',/^}/:zlib.c`. You could also give it a range of lines or a single line number and you'll get the same sort of output.

prev | next