

;login:

THE MAGAZINE OF USENIX & SAGE

December 2001 • Volume 26 • Number 8

inside:

PROGRAMMING

Searching Through Your Files with
Glimpse

By Bob Gray

USENIX & SAGE

The Advanced Computing Systems Association &
The System Administrators Guild

searching through your files with glimpse

by Bob Gray

Bob Gray is co-founder of Boulder Labs, a digital video company. Designing architectures for performance has been his focus ever since he built an image processor system on UNIX in the late 1970s. He has a Ph.D. in computer science from the University of Colorado.



bob@cs.colorado.edu

You have hundreds of megabytes of emails, FAQs, documents, and source code. You need to find something that you only vaguely remember. What are you going to do? You could start looking with an editor, you could try `grep`, but there is a better way.

Recently someone asked me about resisting poison ivy while hiking. I knew I had an email or FAQ about the topic, but it had been years since I had saved the information. In two seconds, I located the article with this command:

```
% glimpse -W 'poison;ivy'
```

In contrast, a recursive version of `egrep` required 300 seconds to search my 11,000 files totaling 250MB. Further, the command

```
% find . -print | xargs egrep 'poison|ivy'
```

yields dozens of inappropriate matches (including binary files) because it matches lines containing either “poison” or “ivy”, whereas the `glimpse -W` option requires that both words be present in the same file.

Glimpse is an indexing and query system that allows you to search through files very quickly. Glimpse has a lot of overlapping capabilities with `grep`, but they each have their own sweet spots. In this article, I’ll start with a few examples, then I’ll provide some background. We’ll look at the features of this tool and show its performance. By the end, you’ll have enough information for deciding whether to add `glimpse` to your repertoire.

Glimpse: Practical Examples

My email folders are reasonably tidy – I delete unneeded messages, yet my email still consumes more than 25MB in over 4000 messages. My email client, `exmh`, presorts new mail into a hierarchy of files rooted in the directory `$HOME/Mail`. Often, I need to retrieve old messages that I only vaguely remember. Using `glimpse`, it’s easy to find the desired message in the `$HOME/mail` tree:

```
% glimpse -F Mail 'master;boot;record'
```

If necessary, the search can be improved with the case insensitive option, `-i`, the complete word option, `-w`, and/or the file as a record option, `-W` (more details below). The authors of `glimpse` even suggest that you alias `glimpse` with `'glimpse -i -w'` because it’s generally most useful.

At the University of Colorado (see “Teaching Operating Systems with Source Code UNIX”)¹, I insist that the students load `glimpse` to aid in working with a large body of code. Tracing through function calls or variables is easy once an index exists. One of my goals for this class is demonstrating how to get comfortable with a large, unfamiliar code base. Tools such as `glimpse` and `editor tags` are essential (`man etags`).

Glimpse History

Glimpse was developed by Udi Manber and Burra Gopal of the University of Arizona and Sun Wu of the National Chung-Cheng University, Taiwan. They published “GLIMPSE: A Tool to Search through Entire File Systems” in the 1994 Winter USENIX Proceedings. The paper is on the USENIX Web site.² Much of *glimpse* is based on their earlier work with *agrep* (see “AGREP - A Fast Approximate Pattern-Matching Tool,” published in the 1992 USENIX Proceedings).³

Glimpse has continued to evolve over the years, and there now is a cooperative development organization (see <http://webglimpse.org/>) to advance this software and its derivatives. To support the effort, they collect a license fee when *glimpse* is used commercially.

Although not a part of standard UNIX⁴ distributions, *glimpse* is freely available. There are Linux RPMs,⁵ precompiled binaries, and an entry in the FreeBSD ports tree. *Glimpse* 3.6, available from,⁶ can be used without licenses. *Glimpse* 4.12.6⁷ is free for noncommercial use, but commercial use requires a license. *Glimpse* source code is available from many locations. An “archie” search will enumerate source code sites. Try: <http://elfikom.physik.uni-oldenburg.de/Docs/net-serv/archie-gate.html> (use the keyword “*glimpse-3*”).

Glimpse Features

Glimpse, like *grep*, is a UNIX searching tool that helps you find content in files. Whereas *grep* finds patterns in one or more files by on-the-spot examination, *glimpse* instead consults a pre-built index to perform the query. The advantage is speed – files comprising hundreds of megabytes can be searched in seconds. The disadvantage is the extra space and time required to compute the index. Assuming a hierarchy of ASCII files, the index requires an additional 2–3% disk space, or for maximum performance, 20–30%. The time to compute the index is on the order of the time it takes to *grep* through the same files. But I keep a fresh index ready for searching with a *crontab* entry.

```
53 4 * * * /usr/local/bin/glimpseindex . >/dev/null
```

builds my home directory (“”) *glimpse* index every morning at 4:53 a.m. and stores it under \$HOME. Alternate indexes can be built for any hierarchy and stored in an arbitrary directory using the *-H* option. Let’s run through a few examples to show various *glimpse* features:

```
% glimpse windsurfing
```

will match lines that contain the target word. A ‘-i’ will make the search case insensitive.

```
% glimpse 'Arizona desert:windsurfing'
```

will find all lines that contain both “Arizona desert” and “windsurfing”.

```
% glimpse -W 'license;hash;expired;features'
```

requires that all four words exist somewhere in the file. For those files, *glimpse* will output the lines that contain any of the words.

```
% glimpse -w ivy
```

requires complete word match; “divy” and “bivy” won’t match.

Glimpse, like *grep*, is a UNIX searching tool that helps you find content in files.

Glimpse has flexibility on the definition of a record.

```
% glimpse -F '\.c$' union
```

searches for the word “union” in “C” files. The -F option limits the search to those files whose name matches the given parameter: in this case, files ending with the C file suffix .c (for example, kern/vfs_bio.c and vm/vnode_pager.c). The -F option allows a case-insensitive flag, so -F '-i faq' would look in file names containing “faq”, “Faq”, etc. And -F '-v \.c\$' would conduct the search in anything BUT C files.

```
% glimpse -2 pneumatic
```

will find all occurrences of “pneumatic” allowing two spelling errors. That would include “mnemonic”, “pneumonia”, and “newmonics”. This feature is part of agrep, where an integer between 1 and 8 specifies the maximum number of errors permitted in finding the approximate match (the default is zero). Generally, each insertion, deletion, or substitution counts as one error. Also from agrep is the Boolean matching concept illustrated in the next two examples.

```
% glimpse '{political,computer};science'
```

will match lines with any of these strings: “political science”, “computer science”, or “science of computers”.

```
% glimpse -W 'fame;~glory'
```

will output all lines containing “fame” in all files that contain “fame” but do not contain “glory”.

```
glimpse -i -F 'faq$' -d '$$' 'master;boot;record'
```

Glimpse has flexibility on the definition of a record. The -d option allows you to override the default record delimiter, '\$', that is, a line is a record. In the example, -d '\$\$' defines paragraphs as records, so in any file name ending with 'faq', it will find occurrences of 'master;boot;record' all in the same paragraph. For searching in files containing email the option -d '^From ' defines records as entire email messages.

I’ve highlighted the features of glimpse that I’ve found most useful over the years. Read the manual page to see how glimpse can best help you.

Glimpse Performance

This section gives some time and space requirements of glimpse. I’ll measure performance on the freely available FreeBSD 4.4 kernel sources of September 2001 so that my experiments can be repeated by the readers. This is a rather small sample to index, but it is still useful and realistic for those needing to deal with kernel source code. I’ll use a modest 200 MHz, 32MB PC with a SCSI disk that’s a few years old. Let’s characterize the body of source code.

```
# cd /usr/src/sys
# du .
574 ./alpha/alpha
...
248 ./ufs/ufs
646 ./ufs
536 ./vm
----
48083 .
```

Clearly, glimpse enables much faster searching.

```
# find . -type f -print | xargs wc
...
  1018      3552      25067 ./vm/vnode_pager.c
    61       408      2777 ./vm/vnode_pager.h
512603 1869237 14093776 total

# find . -type f -print | wc
 3472   3472   80303
```

There are 3,472 source code files taking up 48MB of disk space. In total, the kernel consists of 512,603 lines, 1,869,237 words, and 14,093,776 characters. We'll measure how long `grep` takes to make a typical search in this code base and then look at the elapsed time for `glimpse`, assuming the index exists. (For measuring time, I'll use the built-in shell command `time` and report only the elapsed-time component).

```
# cd /usr/src/sys
# find . -type f -print | xargs grep vm_pageout_deficit
./kern/vfs_bio.c: vm_pageout_deficit += ...
...
./vm/vm_pageout.h:extern int vm_pageout_deficit
elapsed time: 39 seconds

# glimpse -H . vm_pageout_deficit
kern/vfs_bio.c: vm_pageout_deficit += ...
...
vm/vm_pageout.h: extern int vm_pageout_deficit;
elapsed time: 0.5 seconds
```

Clearly, `glimpse` enables much faster searching. (The `-H` option tells `glimpse` to consult the index in the current directory.) If you own a 1 GHz PC, don't assume you could search five times faster than with a 200 MHz PC. Realize that `grep` is mostly an I/O-bound process because you have to read 3,472 files to conduct the search. Let's look at the cost of building the index. As with `grep`, `glimpseindex` is also I/O bound.

```
# glimpseindex -H . .
Indexing "/usr/src/sys" ...

Size of files being indexed = 45988952 B, Total #of files = 3457 ...

-rw----- 1 root    117885   Sep 28 08:55 .glimpse_filenames
-rw----- 1 root     13828   Sep 28 08:55 .glimpse_filenames_index
-rw----- 1 root   2563925   Sep 28 08:55 .glimpse_index
-rw----- 1 root     417     Sep 28 08:55 .glimpse_messages
-rw----- 1 root     880     Sep 28 08:55 .glimpse_partitions
-rw----- 1 root    12341   Sep 28 08:55 .glimpse_statistics
elapsed time: 94 seconds
```

`Glimpseindex` builds an index of the tree rooted at `."` and, with `"-H ."`, stores it in the current directory. Remember, you don't need to run `glimpseindex` very often, so the 94 seconds can support a lot of cheap `glimpse` searches. The index size is almost 3MB to index 48MB of data, or 6%. The `glimpse` authors recommend that most casual users create the smallest index by specifying the `-o` option. And where searching speed is paramount, build a larger index with the `-b` option.

The astute reader may notice that `glimpseindex` reports: "Total #of files = 3457" but `"find . -type f -print | wc"` reports 3,472. That's because there are a couple of binary files in the hierarchy that `glimpseindex` skips. It also makes an effort to identify and skip

REFERENCES

1. <http://boulderlabs.com/12.teaching>.
2. http://www.usenix.org/publications/library/proceedings/sf94/full_papers/manber.glimpse.
3. <http://www.usenix.org/publications/library/proceedings/wu.pdf>.
4. As always, I loosely use the term “UNIX” to mean UNIX-like systems, including Linux, {Free, Open, Net}BSD, Solaris, etc.
5. <ftp://linux1.fnal.gov/linux/611/SRPMS//glimpse-4.1-4.src.rpm>.
6. <ftp://ftp.cs.tu-berlin.de/pub/linux/Mirrors/sunsite.unc.edu/utis/text/glimpse-3.6.src.tgz>.
7. <ftp://ftp.polito.it/pub/tools/unix/harvest/glimpse-4.12.6.tar.gz>.
8. <http://swexpert.com/> (click on UNIX Basics).
9. <http://webglimpse.org/>.
10. <http://www.tardis.ed.ac.uk/harvest/> and <http://www.si.uniroma1.it/mirror/harvest-net/>.

other non-ASCII files such as compressed, uuencoded, and postscript files. You can customize the skipped files with a `.glimpse_exclude` file.

On the other hand, sometimes you want to index what is kept in compressed files. Using a `.glimpse_include` file, you can arrange for `glimpseindex` to examine otherwise ignored files. The `.glimpse_filters` file allows you to specify a program to explode the coded files so that `glimpseindex` has something to work with. For example, if `.glimpse_filters` includes the line

```
*.Z uncompress <
```

then any file ending in `.Z` is uncompressed before `glimpseindex` sees it. The file itself is not changed (i.e., it stays compressed).

Miscellaneous

Three quick side notes: first, my colleague Peter Collinson just wrote an excellent tutorial, “Grep Is Fundamental,” in the September 2001 *Server/Workstation Expert*.⁸

Second, `Webglimpse` is a by-product of `glimpse` for indexing Web sites.⁹ Its predecessor, `Harvest`, provides some interesting history.¹⁰

Third, sometimes it’s not the content you want to search but just the file names. I’ve mentioned that I automatically create a daily `glimpse` index. I also create a `FIND` file:

```
find $HOME -print > $HOME/.DOT/.FIND
```

I have a shell script, `ef`, that consults this list to help me locate file names:

```
% ef backpacking
/usr/people/bob/BACKPACKING
/usr/people/bob/BACKPACKING/REPAIRS
/usr/people/bob/BACKPACKING/food
/usr/people/bob/BACKPACKING/BP-LIST
```

The first argument to the script is the string I’m looking for. Successive arguments are filters to eliminate noise. Here is the essence of the `ef` script:

```
Ist=$HOME/.DOT/.FIND
case $# in
1) egrep -i $1 $Ist ;;
2) egrep -i $1 $Ist | egrep -v $2 ;;
3) egrep -i $1 $Ist | egrep -v $2 | egrep -v $3 ;;
4) egrep -i $1 $Ist | egrep -v $2 | egrep -v $3 | egrep -v $4 ;;
```

To find any file name, give it as the first argument. If you “hit-the-jackpot,” start adding filter arguments until the list shows just what you want. Try it and it will become clear.

Thanks to reviewers Dave Clements, Tom Poindexter, and Steve Gaede.