

TriFind

Advanced Harvest Search Facility

User Guide Version 1.0

Trinem Consulting Limited
Lochside House
3 Lochside Way
Edinburgh Park
Edinburgh
EH12 9DT

www.trinem.co.uk



Introduction

TriFind is a server-side tool which allows a Harvest repository to be quickly searched for items which meet specified criteria. Its most useful feature is that it allows checked-in versions to be located on the basis of text that they contain.

Unlike other search facilities which have to check the code tree out in order to perform the search, *TriFind* performs the entire search at the database level, bypassing the Harvest API. As such the performance is exceptional.

TriFind is an open-source product written entirely in 'C' and Oracle Pro*C. It does not require Harvest to be online in order for it to operate.

TriFind operates at the repository rather than the project level. This means that any project with items in the specified view path can be included in the search results. However, you can limit the search to specific projects if you wish.



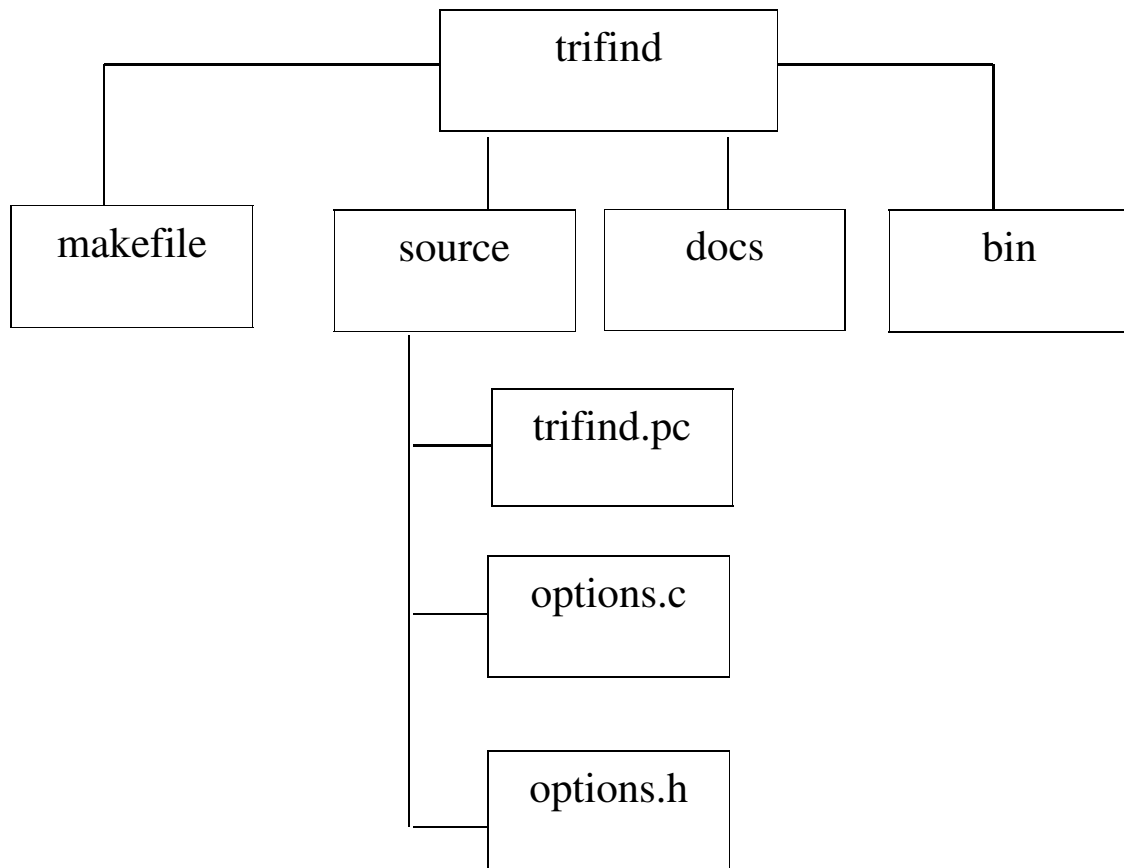
Pre-Requisites

Before you can build and install *TriFind* you need to make sure that you have the following:

1. A user account on the machine where the Harvest Database is located.
2. Access to the Oracle Pro*C pre-compiler. Note, that this is not installed with Oracle by default so you may need help from your site's DBA at this point.
3. Access to a 'C' compiler. If the database server machine does not have a 'C' compiler available then you can install and use the GNU compiler (<http://gcc.gnu.org/>)
4. Access to the *zlib* compression library. This is the library that Harvest uses to compress (and decompress) the contents of the repository. You can download and compile the library from the *zlib* website (<http://www.gzip.org/zlib/>).

Installation

TriFind can be located in any server directory. The initial code tree is shown below:

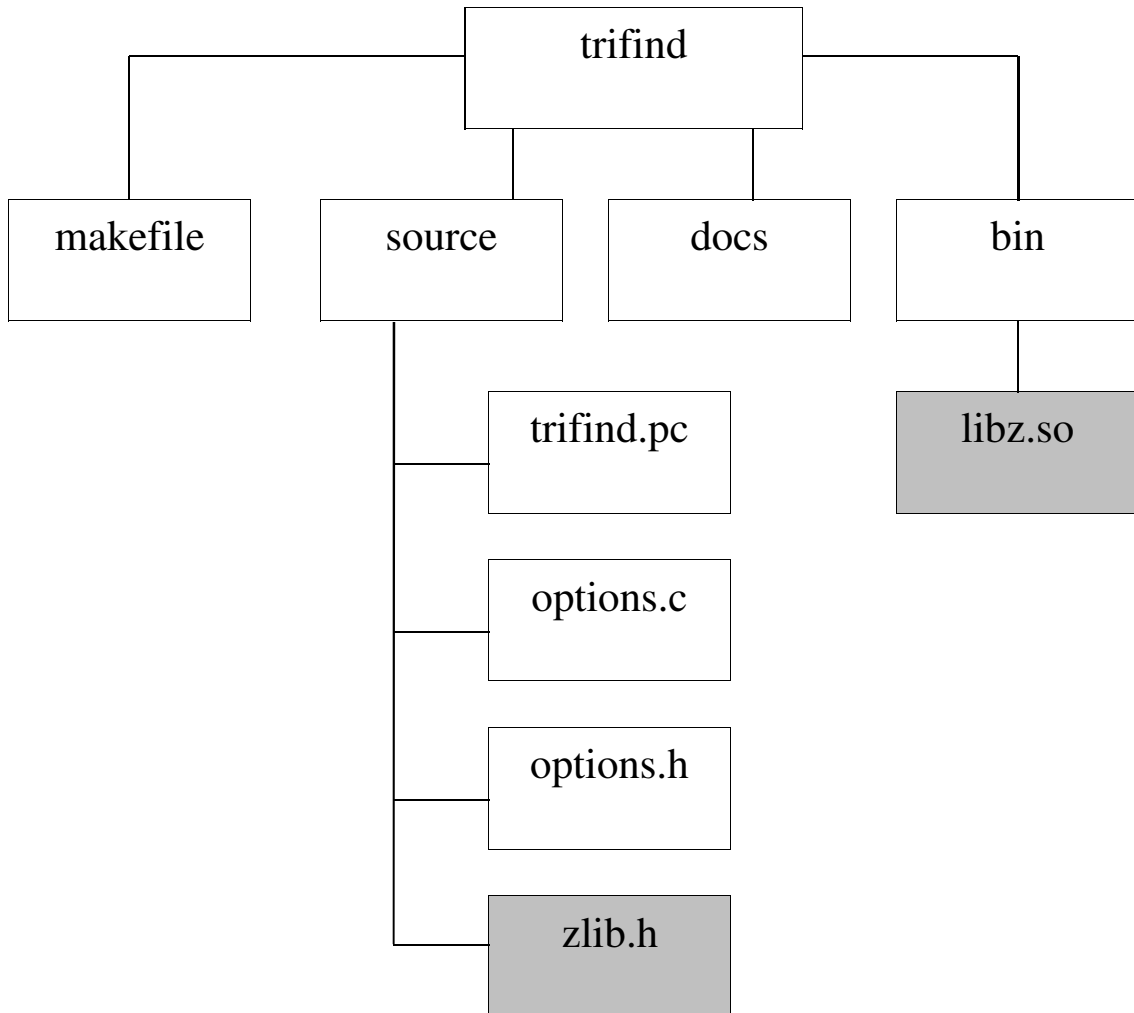


You can extract the tar file as any user. Note that the “bin” directory will be empty initially.

1) Build the *zlib* library.

Full instructions for the compilation and build of the *zlib* compression library is available on the *zlib* website (<http://www.gzip.org/zlib/>). You should create the library as a shared object (`libz.so`). Once you have created this shared object you should copy it into the `trifind/bin` directory. The include file (`zlib.h`) should be copied to the `trifind/source` directory.

After the compression library has been built and the files copied, the TriFind directory tree should look like this:



 = Files copied from zlib

2) Create the “trifind” executable.

You should edit `makefile` in order to identify the location and name of the compiler you will be using to build *TriFind*. Change the line that begins `CC=` to identify the path to your compiler location. You may also need to amend the path to the Pro*C compiler (`proc`).

Once the `makefile` has been amended you should invoke the compiler(s):

- Change your working directory to the `trifind` base directory (the directory containing the `makefile`)
- Run the `make` command.

Upon successful completion of the `make` command, the *TriFind* executable (`trifind`) will be found in the `trifind/bin` directory.

Running *TriFind*



The `-h` option causes *TriFind* to output help text.

There are three ways to pass parameters to *TriFind*:

- 1) By command line switches
- 2) By setting appropriate environment variables
- 3) By including options in a `trienv` configuration file

TriFind will look for options in the order specified above. In other words, command line switches take precedence over environment variables and environment variables take precedence over variables specified in the `trienv` configuration file.



This mechanism allows the command line to be simplified but it is not mandatory.

Each command line switch has an associated environment variable. Environment variables can be specified either in the calling shell or in the `trienv` configuration file(s).

The `trienv` configuration file(s) have the following syntax:

```
<envname>=<value>
<envname>=<value>
.
.
```

Comments can be included and are identified with `#` characters. Anything after the `#` character is ignored.

Here is a sample `trienv` configuration file:

```
#
# Sample config file
#
TRIUSER=harvest          # username
TRIPASS=harvest          # password
TRIRECSEARCH=Y           # recursive search
TRIVIEWPATH=/sdkrep      # viewpath
```



You can use these files to hide the Oracle Username and Password and prevent it appearing on the command line.

If the command line switch takes an argument (e.g.: `-usr` or `-en`) then the value of the corresponding environment variable is the value of the argument. If the command line switch does not take an argument (e.g.: `-s` or `-t`) then the corresponding environment variable should simply be set to a non-null value.

When run, *TriFind* will look for a **trienv** configuration file in **\$HOME/.trienv**. Should this file exist, then any environment variables specified in this file are used accordingly.

Regardless of whether **\$HOME/.trienv** exists, *TriFind* will then look in one of the following locations for a **trienv** file and will take the switches specified in the first file it finds. Once the file is located, further searching stops. Note, however, that variables specified in **\$HOME/.trienv** always take precedence.

- 1) **\$HARVESTHOME/trienv**
- 2) **/etc/trienv**

This mechanism can be used to set up common options so that it is not necessary to specify them on the command line on each invocation of *TriFind*.

Command Line Switches

This table shows each option and its associated environment variable:

<i>command line option</i>	<i>environment variable</i>	<i>description</i>
-usr	TRIUSER	Oracle User Name
-pw	TRIPASS	Oracle Password
-en	TRIENV	Project Name. Only versions present in the specified project will be returned.
-datefmt	TRIDATEFORMAT	The format in which dates are specified (entered). Defaults to DD/MM/YYYY. ¹
-disptime	TRIDISPDATEFORMAT	The format in which dates are displayed. Defaults to the same as -datefmt or to DD/MM/YYYY.
-from	TRIDATEFROM	Start date. Any versions returned will have a creation date on or after this date.
-to	TRIDATETO	End date. Any versions returned will have a creation date on or before this date.
-vp	TRIVIEWPATH	The view path. Paths can be separated either in DOS (\) or Unix (/) format.
-s	TRIRECSEARCH	Recursive search. If specified then <i>TriFind</i> will descend into each subdirectory beneath the specified view path.
-contains	TRISEARCHTEXT	Specifies the text to search for in the versions retrieved.

¹ USA and Canadian users may wish to change the default date entry format to MM/DD/YYYY. The default can be found in `trifind.pc`. You will need to recompile following the change.

<i>command line option</i>	<i>environment variable</i>	<i>description</i>
-attr	TRISEARCHATTR	Specifies the attributes to be returned for each matched version. Valid attributes are given below.
-nh	TRINOHEADER	Do not include headers in the output.
-t	TRITAB	Separate the output attributes with tab characters (or that specified with -fs) rather than an attribute per line.
-fs	TRIFIELDSEP	When used in conjunction with -t, specifies the character(s) used to separate the attributes.
-nocase	TRICASEINSENSITIVE	Makes the search string specified with -contains case insensitive.
-nobase	TRINOBASEVERSIONS	Exclude BASE versions from the search.
-baseonly	TRIBASEVERSIONSONLY	Exclude non-BASE versions from the search.
-allinview	TRIALLINVIEW	By default, <i>TriFind</i> will only return the latest matching version in each view. Using this option allows <i>TriFind</i> to return all versions in a view that match the other specified criteria.
-extract	TRIVERSIONTOEXTRACT	Extract the version and write it to standard output (see below).
-tag	TRITAGTYPE	Only return versions with the specified tag type. Valid tag types are listed below.
-h	TRIHELP	Show Help Text.

Specifying Output Attributes

Versions found that match the specified criteria are output to standard output. The attributes printed are configurable by use of the `-attr` switch. The attributes required should be separated by commas and should be one or more of the following:

<i>Attribute Name</i>	<i>Description</i>
project	The name of the project containing the matched version.
envobjid	The internal object id of the project containing the matched version.
itemname	The item name of the matched version
itemobjid	The internal object id of the matched item
pathname	The path name of the directory containing the matched version
fullitempath	The full item path (directory+item) of the matched version
viewname	The name of the view containing the matched version
viewobjid	The internal object id of the view containing the matched version
statename	The name of the state containing the matched version
stateobjid	The internal object id of the state containing the matched version
package	The name of the package with which the matched version is associated
packageobjid	The internal object id of the package with which the matched version is associated
version	The version number of the matched version
versionobjid	The internal object id of the matched version
creationtime	The date/time when the version was created. Output in the format specified by the <code>-disptimefmt</code> switch (or by the <code>TRIDISPDATEFORMAT</code> environment variable)



If no `-attr` flag is specified then it defaults to:

`"project,version,package,creationtime,fullitempath"`

Searching by Version Tag

The `-tag` option allows you to restrict the search to versions whose tag matches that specified. Valid options to `-tag` are:

<i>Tag Name</i>	<i>Description</i>
all	Default – effectively ignores tag type
none	Only normal versions match
reserved	Only Reserved versions match
merged	Only Merged-Tagged versions match
removed	Only Removed versions match
any	Matches versions with any tag other than Normal.

Extracting a version using *TriFind*

You can use *TriFind* to extract the contents of a specified version which it will then write to standard out. You can do this by using the `-extract` option and specifying the version object id as its parameter.

The version object id can be found as a result of a *TriFind* search (it can be specified in the displayed attributes with `-attr`). Thus, using *TriFind*, a Harvest administrator can find and extract versions.

Here is an example:

```
trifind -usr harvest -pw harvest -extract 17463 > extracted.file
```

This will extract the version identified by the version object id (versionobjid) 17463 and write it to the file “extracted.file”.

Examples

The following examples assume that a trienv file has been created in the appropriate locations containing the Oracle username and password, the recursive flag and the view path for the desired repository:

e.g:

```
#  
# Sample config file  
#  
TRIUSER=harvest          # username  
TRIPASS=harvest          # password  
TRIRECSEARCH=Y           # recursive search  
TRIVIEWPATH=/sdkrep      # viewpath
```

1) Search for versions containing the text “appended”:

```
trifind -contains "appended"  
Project          SDKSampleProject3  
Version          2  
Package          pag test pack  
Creation Time    10/11/2003  
Full Item Path   \sdkrep\HSDKTestFile.txt  
  
Project          SDKSampleProject3  
Version          0  
Package          BASE  
Creation Time    09/11/2003  
Full Item Path   \sdkrep\HSDKTestFile.txt  
  
Project          SDKSampleProject2  
Version          0  
Package          BASE  
Creation Time    09/11/2003  
Full Item Path   \sdkrep\HSDKTestFile.txt  
  
Project          SDKSampleProject  
Version          1  
Package          Package- 7
```



Creation Time 24/09/2003
Full Item Path \sdkrep\HSDKTestFile.txt

2) Search for *all versions* in views containing the text “appended”:

```
trifind -contains "appended" -allinview
Project SDKSampleProject3
Version 2
Package pag test pack
Creation Time 10/11/2003
Full Item Path \sdkrep\HSDKTestFile.txt
```

```
Project SDKSampleProject3
Version 1 ←
Package pag test pack
Creation Time 09/11/2003
Full Item Path \sdkrep\HSDKTestFile.txt
```

```
Project SDKSampleProject3
Version 0
Package BASE
Creation Time 09/11/2003
Full Item Path \sdkrep\HSDKTestFile.txt
```

```
Project SDKSampleProject2
Version 0
Package BASE
Creation Time 09/11/2003
Full Item Path \sdkrep\HSDKTestFile.txt
```

```
Project SDKSampleProject
Version 1
Package Package- 7
Creation Time 24/09/2003
Full Item Path \sdkrep\HSDKTestFile.txt
```

Note the difference here is that by using the `-allinview` flag, earlier versions of items within the same view are included in the search.

You can use this to scan all versions in a repository. For example, if you were trying to establish at which version a line was removed from a file, you would use `-allinview` and `-contains`.



3) Adjust the output to include the item and version object ids

```
trifind -contains "appended" -allinview -attr  
"project,version,package,creationtime,fullitempath,versionobjid,itemobjid"
```

```
Project          SDKSampleProject3  
Version          2  
Package          pag test pack  
Creation Time    10/11/2003  
Full Item Path   \sdkrep\HSDKTestFile.txt  
versionobjid     1093  
itemobjid        920
```

```
Project          SDKSampleProject3  
Version          1  
Package          pag test pack  
Creation Time    09/11/2003  
Full Item Path   \sdkrep\HSDKTestFile.txt  
versionobjid     1091  
itemobjid        920
```

```
Project          SDKSampleProject3  
Version          0  
Package          BASE  
Creation Time    09/11/2003  
Full Item Path   \sdkrep\HSDKTestFile.txt  
versionobjid     1090  
itemobjid        920
```

```
Project          SDKSampleProject2  
Version          0  
Package          BASE  
Creation Time    09/11/2003  
Full Item Path   \sdkrep\HSDKTestFile.txt  
versionobjid     1088  
itemobjid        920
```

```
Project          SDKSampleProject  
Version          1  
Package          Package- 7  
Creation Time    24/09/2003  
Full Item Path   \sdkrep\HSDKTestFile.txt  
versionobjid     1052  
itemobjid        920
```



4) Get the contents of a specified version

Note, in the example above, we displayed the version object id (versionobjid). Version 1 in SDKSampleProject3 has versionobjid=1091. Let's print it out:

```
trifind -extract 1091
This is a test file: ./HSDKTestFile.txt
This has been appended to the file.
This is version 1 in SDKSampleProject3
```

5) Put TriFind output into machine-readable format.

```
trifind -contains "version 1" -nh -t -fs '|' -allinview
SDKSampleProject3|2 |pag test pack|10/11/2003|\sdkrep\HSDKTestFile.txt|
SDKSampleProject3|1 |pag test pack|09/11/2003|\sdkrep\HSDKTestFile.txt|
```

Note the `-t` and `-fs` combination which specifies that each matching version should occupy a single line with each field separated by a `'|'` character.