mseedrecover(1)

Name

mseedrecover - recover miniSEED data from corrupted or damaged files

Synopsis

> mseedrecover [--dry-run] [-v|--verbose]
> [--start-offset=*OFFSET*] [--stop-offset=*OFFSET*]
> [--search-interval=*SIZE*] [--no-dedupe]
> [--output-dir=*DIRECTORY* [--force-overwrite]
> [--force-concat] [--force-sort]]
> [*file* | *directory*]...

> mseedrecover [-h|--help] [--version] [--sysinfo]

Description

Mseedrecover reads from a *file* and tries to find valid miniSEED records in the input. If a *directory* is given at the command line, **mseedrecover** searches recursively for input files inside that directory. If no file or directory argument was given, the standard input is scanned for miniSEED data instead.

While reading, the input is searched for certain byte patterns that are typical for all miniSEED records. If a potential record is located, **mseedrecover** will attempt to read it completely. If successful, the integrity of the data is checked next. Intact miniSEED records are written to standard output (i.e. console) or saved in an output directory (use option **--output-dir**). All damaged or incomplete fragments are discarded. Non-miniSEED data will be ignored and skipped!

Unlike most other GIPPtool utilities, **mseedrecover** is quite unassuming about its input. If necessary, **mseedrecover** will read data from a device file directly (see the example section below) or scan an (uncompressed) disk image file.

Options

The program pretty much follows expected Unix command line syntax. Some of the command line options have two variants, one long and an additional short one (for convenience). These are shown below, separated by commas. However, most options only have a long variant. The ‘=’ for options that take a parameter is required and can not be replaced by a whitespace.

-h, --help

Print a brief summary of all available command line options and exit.

--version*: Print the **mseedrecover** release information and exit.

--sysinfo*: Provide some basic system information and exit.

--dry-run

Perform a trial run with no changes and modifications made whatsoever to the miniSEED data, while at the same time producing (almost) the same user feedback as a real run. It is most commonly used in combination with the **--verbose** options to see what an **mseedrecover** command is about to do before one actually runs it for real.

-v, --verbose

This option increases the amount of information given to the user during the program execution. By default (i.e. without this option) **mseedrecover** only reports warnings and errors. (See the diagnostics section below.)

--start-offset=*OFFSET*

Before any data is analyzed, the first *OFFSET* bytes are skipped and ignored from the input. Use this option to "fast forward" the input to a position where you suspect recoverable miniSEED data.

--stop-offset=*OFFSET*

The search for miniSEED data fragments will be aborted after the stop offset limit was passed. The *OFFSET* is as absolute position from the beginning of the input and given in bytes (and not relative to the **--start-offset** parameter).

This offset is not used as an absolute, hard stop criteria! If the limit is reached while **mseedrecover** currently extracts valid miniSEED data from the input, the program will rather finish extracting the current (intact) data fragment than to stop abruptly in the middle of some miniSEED record.

--search-interval=*SIZE*

Search interval in bytes. When searching for miniSEED data in the input, the program will look every *SIZE* bytes if a valid miniSEED record starts at that offset. Although in principal, any positive number is possible as interval, it probably does not make sense to use anything other than a power of two (i.e. 1, 2, 4, 8, 16, etc.).

The default value of 512 bytes seems to be the best compromise between speed and thoroughness of the search for most situation. However, to make absolutely sure you dont miss anything chose an interval of 1.

-no-dedupe

Disable the detection of duplicate miniSEED records. The default behavior however is to automatically detect duplicates and to only save one single copy per record.

Multiple instance of the same miniSEED record are often encountered when rescuing data directly from a disk drive while bypassing the filesystem level (using "block level access"). This is because EDR/EDL recorder create a new copy of an existing miniSEED file before appending more records to it. After the data is appended the original file is deleted. Although not longer accessible from the filesystem level the data of the original, old file is still on the disk where it may become overwritten eventually. These already deleted files are the origin of the multiple copies.

--output-dir=*DIRECTORY*

Save all recovered records to this *DIRECTORY*. The directory must already exist and be writable! Already existing miniSEED files in that directory will not be overwritten unless the option **--force-overwrite** is used as well.

--force-overwrite

Overwrite already existing files in the output directory, which happen to have the same filename as freshly recovered miniSEED fragments, without mercy!

The default behavior however is **not** to overwrite already existing files. Instead a new file is created with the file number .1 (or .2, .3, etc.) added before the extension.

--force-concat

Concatenate the miniSEED output, creating as few new files as possible. This means that a new output file is only started when there is a (data) discontinuity in the miniSEED input. Without discontinuity the converted data is simply appended to the currently used output file.

By default however, a separate new output file is started for every single miniSEED input file.

--force-sort

Sort the recovered miniSEED fragments. If this option is set, an additional (sub-)directory level (consisting of *year* and *day-of-year*) will be created inside the output directory and the recovered miniSEED records are placed in the respective subdirectories.

The **mseedrename** command provides a more elaborated method to (re-)organize miniSEED files.

Environment

The following environment variables can optionally be used to influence the behavior of the various GIPPtool utilities during startup.

GIPPTOOLS_HOME

This environment variable is used to find the location of the GIPPtools installation directory. In particular, the Java class files that make up the GIPPtools are expected to be in the java subdirectory of **GIPPTOOLS_HOME**.

GIPPTOOLS_JAVA

The utilities of the GIPPtools are written in the programming language Java and consequently need a Java Runtime Environment (JRE) to execute. Use this variable to specify the location of the JRE which should be used.

GIPPTOOLS_OPTS

You can use this environment variable for additional fine-tuning of the Java runtime environment. This is typically used to set the Java heap size available to GIPPtool programs.

It is usually not necessary to define any of those variables as suitable values should be selected automatically. However, if the automatic detection build into the start script fails or you need to choose between different GIPPtool or Java runtime releases installed on your computer, these environment variables might become quite helpful to troubleshoot the situation.

Diagnostics

Mseedrecover occasional will produce user feedback. In general, user messages are classified as *INFO*, *WARNING* or *ERROR*. The *INFO* messages are only displayed when the **--verbose** command line option is used. They usually report about the progress of the program run, give statistical information or write a final summary.

More important are *WARNING* messages. In general, they warn about (possible) problems that may influence the output. Although the program will continue with execution, you certainly should check the results carefully. You might not have gotten what you (thought you) asked for. Finally, *ERROR* messages inform about problems that can not be resolved automatically. Program execution usually stops and the user must fix the problem first.

A good method to see what will happen is to use the **--dry-run** and the **--verbose** command line option at the same time. If user feedback indicates that **mseedrecover** works as expected it can be started again, this time without the **--dry-run** option.

Exit codes

Use the following program exit codes when calling **mseedrecover** from scripts or other programs to see if **mseedrecover** finished successfully. Any non-zero code indicates an *ERROR*.

0

Success.

64

Command line syntax or usage error.

66

An input file did not exist or was not readable.

74

I/O error.

99

Other, unspecified errors.

Examples

1. You have a file that you know must contain miniSEED data. However, none of the other GIPPtool utilities will accept it as input. All you get are error messages, complaining about an "IntegrityException" or other strange things. In other words, you have a damaged file! To see what can be salvaged from the damaged.mseed file you might try the following:

   ```
   mseedrecover --dry-run --verbose --search-interval=1 damaged.mseed
   ```

   From the user feedback generated during the trial run you gather that there is hope to recover at least some of the original content of the file so you next run

   ```
   mseedrecover --verbose --search-interval=1 damaged.mseed > recovered.mseed
   ```

   This will write the recovered data in the respective output file. Comparing the file size of the two files then gives you an idea, how much data was lost.

2. There is a problem with an EDL hard disk. Maybe you cannot copy the miniSEED files from the "data" partition because of a corrupted file systems. Or maybe an overeager assistant accidentally formatted the "data" partition before the recorded miniSEED files were saved. Whatever the reason, to directly recover miniSEED data from an EDL disk (without going through the file system) you might run the following command:

   ```
   mseedrecover --output-dir=./rescue911 --force-sort /dev/sdx3
   ```

   This assumes that you can access the "data" partition of the faulty disk via the device file /dev/sdx3 (EDL disks always use the third partition to store miniSEED data). By reading from the device file directly, you bypass the faulty file system and read the "data" partition block by block. The recovered miniSEED fragments are written to "*year*"-"*day-of-year*" subdirectories in the rescue911 directory.

Files

$GIPPTOOLS_HOME/bin/mseedrecover

The **mseedrecover** "program". Usually just a symbolic link pointing to the standard GIPPtools start script.

$GIPPTOOLS_HOME/bin/gipptools

The GIPPtools start script. Almost all utilities of the GIPPtools package are started from this shell script.

See also

gipptools(1), **cube2ascii**(1), **cube2mseed**(1), **cube2segy**(1), **cubeevent**(1), **cubeinfo**(1), **mseed2ascii**(1), **mseed2mseed**(1), **mseed2pdas**(1), **mseed2segy**(1), **mseedcut**(1), **mseedinfo**(1), **mseedrename**(1)

Bugs and caveats

- The **mseedrecover** program only recovers data in miniSEED format! It cannot recover data in other formats, such as EDL log or configuration files. Unless you can recover those files by other means (maybe some "undelete" or "unformat" tool can help) they are lost!
- When using a device file as input (see example section) you might recover field-data from other experiments. Possibly from experiments many years ago. This is should be of no (privacy) concern as only raw data, miniSEED files are affected. However, you should be aware of it.