

# cubeinspect(1)

## Name

cubeinspect - inspect Cube files searching for problems

## Synopsis

```
cubeinspect [-v|--verbose] [--include-pattern=PATTERN]...  
            [--inspect|--repair-buffer-overflow]  
            [--output-dir=DIRECTORY [--force-overwrite]]  
            file | directory...
```

```
cubeinspect [-h|--help] [--version] [--sysinfo]
```

## Description

**Cubeinspect** screens Cube files for conspicuous features that potentially may cause problems when processing the recordings. For some well-defined cases the utility can also be used to (partly) repair the problematic Cube file.

There are two modes of operation. By default, the **cubeinspect** utility runs in inspection mode (command line option **--inspect**). All *files* and *directories* given at the command line are searched recursively for Cube files, which in turn are scanned one by one for conspicuous features. To speed up the processing, Cube input can be restricted to contain only files with a name matching the patterns given by the **--include-pattern** option.

After all input files were scanned successfully a report is generated for each file. It contains general information about the Cube hardware as well as some statistics about the recording. Each report ends with a list of detected issues.

The other operational mode is the *repair* mode. Again the input files are processed one by one. But this time the indicated modification (option **--repair-buffer-overflow**) is applied to each file. The content of the Cube input is written directly to standard output (i.e. text console) unless the output directory (option **--output-dir**) is given as well.

## Options

The program pretty much follows expected Unix command line syntax. Some 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 cannot be replaced by a whitespace.

## **-h, --help**

Summarize the most important command line options and exit.

## **--version**

Print the **cubeinspect** release information and exit.

## **--sysinfo**

Report system settings relevant for **cubeinspect** and exit.

## **-v, --verbose**

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

## **--include-pattern=PATTERN**

Only process files whose filename matches the given *PATTERN*. Files with a name not matching the search *PATTERN* will be ignored. This option is useful to speed up recursive searches through large subdirectory trees and can be used more than once in the same command line. (In this case files matching any of the provided patterns are included.)

You can use two wild card characters (**\*** and **?**) when specifying an include *PATTERN*. For example using **'\*.123'** selects all files ending with the extension **.123**. Remember to enclose patterns with single quotes when using wildcards or spaces in the context of an **--include-pattern** command line option.

There is also a predefined GIPP filter. Using **--include-pattern=GIPP** automatically selects all data files following the default naming convention for files recorded by loggers borrowed from the Geophysical Instrument Pool Potsdam (GIPP).



The search *PATTERN* is only applied to the filename part and not to the full pathname of a file.

## **--inspect**

The utility will inspect files and diagnose problems. After (!) all input files were scanned a short report is generated for each file.

The report contains a *SYSTEM* section with some basic information about the used recorder (e.g. datalogger ID, installed firmware). It is followed by a *DATA* section providing some statistics about the samples that were recorded (channel minima/maxima given in counts and how often values were clipped). Finally, the *PROBLEMS* section summarizes all detected issues such as [buffer overruns](#), [clipped samples](#) or if the recorded data is affected by a [WNRO](#) event.

Please refer to the [Detected Problems](#) section below for more details on each issue.

## **--repair-buffer-overflow**

This option is used to fill gaps in the recorded data stream that were caused by buffer overrun errors. For a more information about this problem please refer to the [Buffer Overrun](#) description in the **Detected Problems** section below.

When running in *repair-buffer-overflow* mode the **cubeinspect** utility will read Cube data from a file and fill all buffer overrun gaps with dummy samples (zero amplitude). The resulting repaired/modified file can be processed as usual again.



Repairing buffer overruns cannot magically restore the lost samples! The missing samples are gone for good!

All the program can do is fill the gap with the correct number of "dummy samples" so the file can be processed.

### **--output-dir=*DIRECTORY***

Save the resulting files to this *DIRECTORY* instead of writing to the standard output (i.e. text console). The directory must already exist and be writable!

Already existing files in that directory will not be overwritten unless the option **--force-overwrite** is used as well.

### **--force-overwrite**

If this option is used, already existing files in the output directory will be overwritten without mercy!

The default behavior, however, is **not** to overwrite already existing files. Instead, a new file is created with an additional number before the file extension.

## Detected Problems

The **cubeinspect** utility will detect the following problems and errors in Cube files.

### Buffer Overrun

A buffer overrun is an internal recorder error that occurs when data is recorded faster than it can be written to the SD card storage. Eventually, the Cube/Nube internal memory buffer overflows and a handful of samples are lost (i.e. not written to SD card).

The result is a data gap in the (no longer) continues sample stream. The missing data prohibits straightforward conversion to other data formats (e.g. using **cube2mseed**, **cube2segy**, ...). Affected files become usable again when the gap is patched with (dummy) samples.

Unfortunately, the error cannot be avoided by simply switching to a "faster" SD card as the problem is not caused by an insufficient write speed. Most modern SD cards are easily fast enough for the samplings rates selectable in Cube/Nube data loggers.

Instead, the gaps are caused by the increased write latency of the SD card. This happens occasionally, mostly due to wear leveling when the card pauses until it could move data from on area of flash memory to another. Unfortunately, there is no external way to control the internal wear leveling algorithm inside the SD card.

Typically, buffer overrun problem surfaces first at the highest sampling rates while using old, worn out SD cards. In the beginning buffer overrun errors are very rare events. However, with time the

frequency of buffer overrun errors increases. There is no known solution other than replacing the worn out SD card by a new, high quality one.



You can use the `--repair-buffer-overrun` command line option (see [above](#)) to fill the gap with the correct number of dummy samples.

## Week Number Rollover (WNRO)

Cube data logger receive the recording time from the Global Positioning System (*GPS*). Unlike the commonly used Gregorian calendar, *GPS* (internally) expresses date/time information using two integer numbers. The first number counts the weeks since start of the *GPS* system on January 6th, 1980. The second number gives the seconds relative to the beginning of the week.

*GPS* Satellites transmit this week number as a 10-bit long integer, which will become zero again every 1024 weeks. This integer overflow is called week number rollover (*WNRO*) and is a common issue with all *GPS* receivers. Sooner or later, every *GPS* receiver will encounter this problem! A future, modernized *GPS* is proposed to use a 13-bit counter.

All Cube related GIPPTool utilities contain an algorithm that will detect *WNRO* and just correct the date automatically by adding 1024 weeks to the "wrong" recording time. However, in cases where the automatic detection and correction fails, the handling of *WNRO* can be manually controlled using the command line option `--wnro-correction`. Please refer to the documentation of the respective utility for usage details.



The **cubeinspect** utility only lists *WNRO* events as a convenience to the user. There is no need to "repair" Cube files as the issue is already handled transparently by the GIPPTool utilities.

## Clipped Samples

Recorded amplitudes are clipped if the voltage from the sensor exceeds the maximum voltage for the configured gain settings. The *clipped value count* line in the *DATA* section of the inspection report shows the number of clipped samples per recording channel. The number in the *PROBLEMS* section lists the total number of affected samples.

You can change the maximum recorded voltage by adjusting the gain settings.

## Environment

The following environment variables can optionally be used to influence the behavior of the GIPPTool utilities.

### GIPPTOOLS\_HOME

This environment variable can be used to set the GIPPTools installation directory.

In particular, the Java classes that make up the GIPPTools are read from *JAR* files in the **java** subdirectory located inside **GIPPTOOLS\_HOME**. The start scripts for the individual GIPPTool

utilities can be found inside the `bin` subdirectory.

## 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 path to the *JRE*, which should be used.

## GIPPTOOLS\_OPTS

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

## GIPPTOOLS\_LEAP

The GIPPTools require up-to-date leap second information to correctly interpret Cube files. Usually, this information is read from the `leap-seconds.list` file located in the `config` subdirectory of the GIPPTools installation directory (`GIPPTOOLS_HOME`). This environment variable can be used to provide a more up-to-date leap second list 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 helpful to troubleshoot the situation.

# Diagnostics

During execution, the **cubeinspect** utility will produce user feedback. In general, user messages are classified as *INFO*, *WARNING* or *ERROR*.

*INFO* messages usually report about the progress of the program run, give statistical information or write a final summary. They are only displayed when the `--verbose` command line option is used.

More important are *WARNING* messages. In general, they warn about any issues that may influence the outcome in unexpected ways. 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 cannot be resolved automatically. Program execution usually stops and the user must fix the cause of the error first.

# Exit codes

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

0

Success.

64

Command line syntax or usage error.

65

Input data was incorrectly formatted.

66

An input file did not exist or was not readable.

70

Error in internal program logic.

74

I/O error.

99

Other, unspecified errors.

## Examples

1. Scan a Cube file for conspicuous features.

```
cubeinspect 12200000.123
```

The program will output a short report about file `12200000.123` recorded by Cube #123.

2. Repair buffer overrun errors

A buffer overrun error results in lost samples in the recording. To patch the hole in the otherwise continuous time series dummy samples must be inserted at the location of the gap.

```
cubeinspect --repair-buffer-overrun < 12200000.123 > 12200000.patched.123
```

The utility will read in file `12200000.123` and replace lost samples due to *buffer overrun errors* by dummy samples with an amplitude of zero. The modified recording is redirected to file `12200000.patched.123`.

After the file has been repaired, it can take the place of the original file.



The file was not repaired in the sense that the lost samples miraculously were recovered. Only the gaps in the time series were filled with dummy samples!

## Files

`$GIPPTOOLS_HOME/bin/cubeinspect`

The **cubeinspect** "program". Usually just a copy of or a symbolic link pointing to the standard GIPPTools start script.

**\$GIPPTOOLS\_HOME/bin/gipptools**

The standard start script used to run all GIPPtool utilities.

## See also

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

## Bugs and caveats

- The **cubeinspect** should report more details and to detect more problems!