

# cubeinfo(1)

## Name

cubeinfo - summarize the content of a Cube recording

## Synopsis

```
cubeinfo [-v|--verbose] [--include-pattern=PATTERN]...  
          [--output-dir=DIRECTORY [--force-overwrite]]  
          [--format=FORMAT]  
          [file | directory]...
```

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

## Description

**Cubeinfo** reads from a *file* (or standard input) and returns a short textual report of the content of the read Cube recording. If a *directory* is given instead, **cubeinfo** searches recursively for input files inside the directory.

The report is written to standard output (i.e. console) or saved in an output directory (use option **--output-dir**). Different report variants (*SUMMARY*, *DUMP*, ...) are predefined and can be selected using the **--format** option.

## 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 can not be replaced by a whitespace.

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

Summarize the most important command line options and exit.

### **--version**

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

### **--sysinfo**

Report system settings relevant for **cubeinfo** 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) **cubeinfo** 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.

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

Save the resulting reports to this *DIRECTORY*. 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.

### **--format=*FORMAT***

Select one of the following predefined output formats:

#### ***INFO***

Give a brief, minimal description of the content of the Cube recording. The report will contain the Cube instrument id and configuration, battery levels, as well as the approximate start and stop time of the recording. This is the default output format.

#### ***SUMMARY***

A slightly more verbose description of the content also includes statistics about how many data blocks (header, trailer, samples, GPS, ...) are contained in the Cube recording.



This output format requires that the Cube file must be read completely, which may take a few seconds for extensive recordings.

## **GPS, GNSS, TAIP**

List the content of all "GPS" blocks contained in the Cube recording. This output format is useful to get e.g. the number of satellites that were used for the last GPS fix, the age of the GPS fix or the offset between GPS and UTC timescale (caused by leap seconds) that was reported. This may be useful to get a rough idea about the quality of the received GNSS signal.

It is also possible to read the actual geographic position of a Cube while it was recording from the output. However, the `cubeaux` utility is probably be easier to use!

The returned information will depend on the build-in/connected clock hardware, which differs between Cube recorders. Usually the report contains at least the Cube block number and the (UTC) time derived from the "GPS" block. Additional information is provided on an "as available" base and may include the number of leap seconds between GPS and UTC time (as e.g. reported by the GPS satellite and/or as officially announced by the International Earth Rotation and Reference Systems Service, [IERS](#), the time source (2D-GPS, 3D-GPS, ...), the geographic position of the Cube (latitude, longitude and maybe elevation), internal temperature (as measured by the clock hardware), the number of satellites received, and/or the age of the GPS fix ("less than 10s old").



The time information provided by the GPS output format is not identical to the recording time of any sample! Additional time corrections (depending amongst others on Cube hardware and configured sample rate) must be applied first to get a precise recording time.

## **DEBUG, DUMP**

Returns a textual representation of every data block contained in the Cube file. This rather voluminous report includes everything there is and should probably be used for debugging purpose only!

# **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 **cubeinfo** 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 **cubeinfo** from scripts or other programs to see if **cubeinfo** 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. To learn about the content of the single Cube file called **recording.cube**, you simply use one of the following:

```
cubeinfo recording.cube
```

```
cubeinfo < recording.cube
```



The first variant will usually be much faster as **cubeinfo** will jump directly from the beginning of the file, where it reads the "header block", to the end of the file where the "trailer block" required by the INFO report is located.

The second variant does not have that option as it must always read the complete data stream piped in from console. There is, however, no significant difference when requesting a report format other than the default INFO format as the complete recording must be read for other predefined report formats.

2. To get information about how many data blocks are contained in the Cube file **02161251.034** use the *SUMMARY* output format:

```
cubeinfo --format=SUMMARY 02161251.034
```

This will not only return a table of the fields contained in the header and trailer block of the Cube file but also count every data block contained in the file and report those statistics too.



Using the *SUMMARY* output format is a convenient method to quickly check if the Cube file is "complete". If you suspect your Cube recording was cut-off early or might be otherwise damaged you can look for header, trailer and end-of-file block counts. A sound Cube file should contain exactly one of each!

3. To find out at which coordinates the Cube was placed when it recorded the file **02161251.034** you use *GPS* report format:

```
cubeinfo --format=GPS 02161251.034
```

The resulting output contains, besides other information, the coordinates (marked as *lat* and *lon*) of the Cube in degrees. Depending on the build-in GPS hardware additional information about elevation might or might not be available.

## Files

### **\$GIPPTOOLS\_HOME/bin/cubeinfo**

The **cubeinfo** "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

- If a Cube file was cut-off early (i.e. it does not contain a trailer block at the end of the recording) no information about the end of the recording can be given! Although this should be obvious it always seems to surprise the user.