NAME

SoX − Sound eXchange_ng, another Swiss Army knife of audio manipulation

SYNOPSIS

sox_ng [*global-options*] [*format-options*] *infile1*
[[*format-options*] *infile2*] ... [*format-options*] *outfile*
[*effect* [*effect-options*]] ...

play_ng [*global-options*] [*format-options*] *infile1*
[[*format-options*] *infile2*] ... [*format-options*]
[*effect* [*effect-options*]] ...

rec_ng [*global-options*] [*format-options*] *outfile*
[*effect* [*effect-options*]] ...

DESCRIPTION

Introduction

SoX reads and writes audio files in most popular formats and can optionally apply effects to them. It can combine multiple input sources, synthesize audio and, on many systems, act as a general purpose audio player or as a multitrack audio recorder. It can also split the input into multiple output files.

All SoX functionality is available using just the **sox_ng** command. To simplify playing and recording audio, if SoX is invoked as **play_ng**, the output file is automatically set to be the default sound device and, if invoked as **rec_ng**, the default sound device is used as an input source. Additionally, the **soxi_ng** command provides a convenient way to query audio file header information.

The heart of SoX is a library called **libsox_ng**. Those interested in extending SoX or using it in other programs should refer to the **libsox_ng** manual page.

SoX is a command-line audio processing tool particularly suited to making quick, simple edits and to batch processing. If you need an interactive, graphical audio editor, use **audacity**(1).

*   *   *

The overall SoX processing chain can be summarized as follows:

Input(s) → Combiner → Effects → Output(s)

On the SoX command line, the positions of the Output(s) and the Effects are swapped w.r.t. the logical flow just shown and, while options pertaining to files are placed before their respective file names, the opposite is true for effects. To show how this works in practice, here is a selection of examples of how SoX might be used. The simple

```
sox_ng recital.au recital.wav
```

translates an audio file in Sun AU format to a Microsoft WAV file, while

```
sox_ng recital.au -b 16 recital.wav channels 1 rate 16k fade 3 norm
```

performs the same format translation but also applies four effects (down-mix to one channel, sample rate change, fade in and normalize) and stores the result at a bit-depth of 16.

```
sox_ng -r 16k -e signed -b 8 -c 1 voice-memo.raw voice-memo.wav
```

converts ‘raw’ (a.k.a. ‘headerless’) audio to a self-describing file format,

```
sox_ng slow.aiff fixed.aiff speed 1.027
```

adjusts audio speed,

```
sox_ng short.wav long.wav longer.wav
```

concatenates two audio files and

```
sox_ng -m music.mp3 voice.wav mixed.flac
```

mixes together two audio files.

```
play_ng "The Moonbeams/Greatest/*.ogg" bass +3
```

plays a collection of audio files applying a bass boosting effect,

```
play_ng -n -c1 synth sin %-12 sin %-9 sin %-5 sin %-2 fade h 0.1 1 0.1
```

plays a synthesized ‘A minor seventh’ chord with a pipe organ sound,

```
rec_ng -c 2 radio.aiff trim 0 30:00
```

records half an hour of stereo audio and

```
play_ng -q take1.aiff & rec -M take1.aiff take1-dub.aiff
```

(with a POSIX shell and where supported by hardware) records a new track in a multitrack recording. Finally,

```
rec_ng -r 44100 -b 16 -e signed-integer -p \
   silence 1 0.50 0.1% 1 10:00 0.1% | \
   sox_ng -p song.ogg silence 1 0.50 0.1% 1 2.0 0.1% : \
   newfile : restart
```

records a stream of audio such as an LP or cassette and splits it into multiple audio files at points where there are two seconds of silence. Also, it does not start recording until it detects some sound and stops after it sees ten minutes of silence.

The above is just an overview of SoX’s capabilities. Detailed explanations of how to use all SoX parameters, file formats and effects can be found below in this manual, in **soxformat_ng**(7) and in **soxi_ng**(1).

File Format Types

SoX can work with ‘self-describing’ and ‘raw’ audio files. ‘Self-describing’ formats (e.g. WAV, FLAC, MP3) have a header that completely describes the signal and encoding attributes of the audio data that follows. ‘raw’ or ‘headerless’ formats do not contain this information, so the audio characteristics of these must be described on the SoX command line, except for a few which can be inferred from the filename extension such as **.gsm**, always **−c1 −r8000 −e gsm**, and named raw formats like **.f32**, .s16 and .ul which give the encoding but not the sample rate or number of channels.

The following four characteristics are used to describe the format of audio data:

sample rate
: The sample rate in samples per second (‘Hertz’ or ‘Hz’). Digital telephony traditionally uses a sample rate of 8000Hz (8kHz) though 16 and even 32kHz are becoming more common. Audio Compact Discs use 44100Hz (44·1kHz), Digital Audio Tape and many computer systems use 48kHz and professional audio systems often use 96kHz.

sample size
: The number of bits used to store each sample. Today, 16-bit is commonly used, 8-bit was popular in the early days of computer audio and 24-bit is used in the professional audio arena.

data encoding
: The way in which each audio sample is represented (or ‘encoded’). Some encodings have variants with different byte-orderings or bit-orderings, some compress the audio data so that it takes up less disk space or transmission bandwidth than uncompressed formats. Commonly-used encoding types include floating point, μ-law, ADPCM, signed-integer PCM, MP3 and FLAC.

channels
: The number of audio channels contained in the file. One (‘mono’) and two (‘stereo’) are widely used and ‘surround sound’ audio typically contains six or more channels.

The term ‘bit rate’ is a measure of the amount of storage occupied by an encoded audio signal per unit of time. It can depend on all of the above and is typically denoted as a number of kilobits per second (kbps). An A-law telephony signal has a bit rate of 64 kbps, MP3-encoded stereo music typically has a bit rate of

128−196 kbps and FLAC-encoded stereo music typically has a bit rate of 550−760 kbps.

Most self-describing formats also allow textual 'comments' to be embedded in the file that can be used to describe the audio in some way, e.g. for music, the title, the author, etc.

One important use of audio file comments is to convey 'Replay Gain' information. SoX can apply Replay Gain automatically for formats that contain comments but does not generate it. By default, SoX copies comments from the first input file to output files that support comments, so output files may contain Replay Gain information which is incorrect. This can be fixed, when converting input files with **−−replay−gain** enabled, by removing all comments using **−−comment ""** or removing just the REPLAYGAIN comment with

```
soxi_ng −a in.au | grep −v REPLAYGAIN > comments
sox_ng --replay-gain=track in.au --comment-file comments out.au
```

Determining and setting the File Format

SoX uses several mechanisms to determine or set the format of an audio file. Depending on the circumstances, individual characteristics may be determined or set using different mechanisms.

To determine the format of an input file, SoX uses, in order of precedence and as given or available:

1. Command-line format options,
2. The contents of the file header,
3. The filename extension.

To set the output file format, SoX uses, in order of precedence and as given or available:

1. Command-line format options,
2. The filename extension,
3. The input file format characteristics or the closest that is supported by the output file type.

For all files, SoX exits with an error if the file type cannot be determined. Command-line format options may need to be added or changed to resolve the problem.

Playing & Recording Audio

The **play_ng** and **rec_ng** commands are provided so that basic playing and recording is as simple as

```
play_ng existing-file.wav
```

and

```
rec_ng new-file.wav
```

These two commands are functionally equivalent to

```
sox_ng existing-file.wav −d
```

and

```
sox_ng −d new-file.wav
```

Further options and effects (as described below) can be added to the commands in either form.

* * *

When playing a file with a sample rate that is not supported by the audio output device, SoX automatically invokes the **rate** effect to perform the necessary sample rate conversion. For compatibility with old hardware, the default **rate** quality level is set to 'low'. This can be changed by explicitly specifying the **rate** effect with a different quality level, e.g.

```
play_ng ... rate −m
```

or by using the **−−play−rate−arg** option (see below).

* * *

On some systems, SoX allows the audio playback volume to be adjusted while using **play_ng**. Where supported, this is achieved by tapping the '**v**' & '**V**' keys during playback. If there is a **softvol** effect in the chain, these keys will adjust that instead of the hardware mixer.

To help with setting a suitable recording level, SoX includes a peak level meter which can be invoked (before making the actual recording) as follows:

```
rec_ng -n
```

The recording level should be adjusted (using the system-provided mixer program, not SoX) so that the meter is, at most, occasionally full scale and never 'in the red' (an exclamation mark is shown). See the **−S** (**−−show−progress**) option below.

Accuracy

Many file formats that compress audio discard some of the audio signal information. Converting to such a format and back again will not produce an exact copy of the original audio. This is the case for many formats used in telephony (e.g. A-law, GSM) where low signal bandwidth is more important than high audio fidelity and for formats used in portable music players (e.g. MP3, Ogg Vorbis) where adequate fidelity can be retained even with the large compression ratios that are needed to make portable players practical.

Formats that discard audio signal information are called 'lossy'. Formats that do not are called 'lossless'. The term 'quality' is used as a measure of how closely the original audio signal is reproduced when using a lossy format.

Audio file conversion with SoX is lossless when it can be, i.e. when not using lossy compression, when not reducing the sampling rate or number of channels and when the number of bits used in the destination format is not less than in the source format. For example, converting from an 8-bit PCM format to a 16-bit PCM format is lossless but converting from an 8-bit PCM format to (8-bit) A-law isn't.

SoX converts all audio files to an internal, uncompressed 32-bit format before performing any audio processing. This means that manipulating a file that is stored in a lossy format can cause further losses in audio fidelity. E.g. with

```
sox_ng long.mp3 short.mp3 trim 10
```

SoX first decompresses the input MP3 file, then applies the **trim** effect and finally creates the output MP3 file by recompressing the audio with a possible reduction in fidelity above that which occurred when the input file was created. Hence, if what is ultimately desired is lossily compressed audio, it is best to perform all audio processing using lossless file formats and then convert to the lossy format only at the final stage. Applying multiple effects with a single SoX invocation will, in general, produce more accurate results than those produced using multiple SoX invocations.

Dithering

Dithering is a technique used to maximize the dynamic range of audio stored at a particular bit-depth. Any distortion introduced by quantization is decorrelated by adding a small amount of white noise to the signal. In most cases, SoX can determine whether the selected processing requires dither and will add it during output formatting if appropriate.

By default, SoX automatically adds TPDF dither when the output bit-depth is less than 24 and any of the following are true:

- bit-depth reduction has been specified explicitly using a command-line option
- the output file format supports only bit-depths lower than that of the input file format
- an effect has increased the effective bit-depth within the internal processing chain

For example, adjusting the volume with **vol 0.25** requires two additional bits in which to losslessly store its results (since 0·25 decimal equals 0·01 binary) so, if the input file bit-depth is 16, SoX's internal representation will use 18 bits after processing this volume change. In order to store the output at the same depth as the input, dithering is used to remove the additional bits.

Use the **−V** option to see what processing SoX has automatically added. The **−D** (**−−no−dither**) option may be given to override automatic dithering. To invoke dithering manually (e.g. to select a noise-shaping curve) use the **dither** effect.

Clipping

Clipping is distortion that occurs when an audio signal level (or 'volume') exceeds the range of the chosen representation. In most cases, clipping is undesirable and so should be corrected by adjusting the level prior to the point in the processing chain at which it occurs.

In SoX, clipping can happen when using the **vol** or **gain** effects to increase the audio volume. Clipping can also occur with many other effects, when converting one format to another and even when simply playing the audio.

Playing an audio file often involves resampling and processing by analog components and that can introduce a DC offset or amplification, all of which can produce distortion if the audio signal level was initially too close to the clipping point.

For these reasons, it is usual to make sure that an audio file's signal level has some 'headroom', i.e. it does not exceed a particular level below the maximum possible level of the given representation. Some standards bodies recommend as much as 9dB headroom, but in most cases, 3dB (≈ 70% linear) is enough. Note that this wisdom seems to have been lost in modern music production; in fact, many CDs, MP3s, etc. are now mastered at levels *above* 0dBFS and the audio is clipped as delivered.

SoX's **stat** and **stats** effects can assist in determining the signal level of an audio file. The **gain** or **vol** effect can be used to prevent clipping, e.g.

```
sox_ng dull.wav bright.wav gain −6 treble +6
```

guarantees that the treble boost will not clip.

If clipping occurs at any point during processing, SoX displays a warning message to that effect.

See the global **−G** (**−−guard**) option and the **gain** and **norm** effects.

Input File Combining

SoX's input combiner can be configured with the **−−combine** global option to combine multiple files using one of the following methods: **concatenate**, **sequence**, **mix**, **mix-power**, **merge** and **multiply**, with shorthands **−m** for **−−combine mix**, **−M** for **merge** and **−T** for **multiply**.

The default is **sequence** for **play_ng**, and **concatenate** for **sox_ng**.

For methods other than **sequence**, multiple input files must have the same sampling rate. If necessary, separate SoX invocations can be used to make sampling rate adjustments prior to combining them. and, with **concatenate**, the input files must also have the same number of channels.

The **sequence** combining method is similar to **concatenate** in that the audio from each input file is sent serially to the output file but here, the output file may be closed and reopened at the transition between input files. This may be just what is needed when sending different types of audio to an output device but is not generally useful when the output is a normal file.

With the **mix** or **mix-power** combining methods, the number of channels in each input file need not be the same but SoX issues a warning if they are not and some channels in the output file will not contain audio from every input file. A mixed audio file cannot be unmixed without reference to the original input files.

If the **merge** combining method is selected the number of channels in each input file need not be the same and a merged audio file comprises all channels from all the input files and unmerging is possible using multiple invocations of SoX with the **remix** effect. For example, two mono files could be merged to form one stereo file and the first and second mono files would become the left and right channels of the stereo file.

The **multiply** combining method multiplies the sample values of corresponding channels treated as numbers in the interval −1 to +1. If the number of channels in the input files is not the same, the missing channels will contain silence.

When combining input files, SoX applies any specified effects (including, for example, the **vol** volume adjustment effect) after the audio has been combined. However, it is often useful to be able to set the volume of the inputs individually (i.e. 'balance' them) before combining takes place. For all combining methods, input file volume adjustments can be made manually using the **−v** option, which can be given for one or more input files. If it is given for only some of the input files, the others receive no volume adjustment. In some circumstances, automatic volume adjustments may be applied. The global **−V** option can be used to show the input file volume adjustments that have been selected manually or automatically.

Some special considerations need to made when mixing input files:

Unlike the other methods, **mix** combining can cause clipping in the combiner if no balancing is performed. In this case, if manual volume adjustments are not given, SoX tries to ensure that clipping does not occur by automatically adjusting the volume (amplitude) of each input signal by a factor of $1/n$, where n is the number of input files. If this results in audio that is too quiet or otherwise unbalanced, the input file volumes can be set manually with **−v**. Using the **norm** effect on the mix is another alternative.

If mixed audio seems loud enough at some points but too quiet in others, dynamic range compression can be applied to correct this—see the **compand** effect.

With the **mix-power** combine method, the mixed volume is approximately equal to that of one of the input signals. This is achieved by balancing using a factor of $1/\sqrt{n}$ instead of $1/n$. Note that this balancing factor does not guarantee that clipping will not occur but the number of clips will usually be low and the resulting distortion is usually imperceptible.

Output Files

SoX's default behaviour is to take one or more input files and write them to a single output file.

This behaviour can be changed by placing the **newfile** pseudo-effect within the effects list and SoX will enter multiple output mode.

In multiple output mode, a new file is created when the effects prior to the **newfile** indicate that they are done. The effects chain listed after **newfile** is then started up and its output is saved to the new file.

In multiple output mode, a unique number is appended automatically to all filenames and, if the filename has an extension, the number is inserted before the extension. This behaviour can be customized by placing **%n** in the filename where the number should be substituted. An optional number can be placed after the **%** to indicate a minimum width for the number with leading zeroes. **%n** defaults to two digits or, if no **%n** is included, to three digits before the filename extension.

Multiple output mode is not very useful unless an effect that stops the effects chain is specified before **newfile**. If the end of the file is reached before the effects chain stops, no new file is created as it would be empty.

The following is an example of splitting the first 60 seconds of an input file into two 30 second files and ignoring the rest.

```
sox_ng song.wav ringtone%1n.wav trim 0 30 : newfile : trim 0 30
```

Stopping SoX

Usually, SoX completes its processing and exits automatically once it has read all audio data from the input files.

It can also be terminated earlier by sending it an interrupt signal, usually by pressing the keyboard interrupt key which is normally Ctrl-C. This is required in some circumstances such as when using SoX to make a recording. When SoX is playing multiple files, Ctrl-C behaves slightly differently: pressing it once skips to the next file; pressing it twice in quick succession causes SoX to exit.

Another way to stop processing early is to use an effect that has a time period or sample count; the **trim** effect is an example of this. Once all effects chains have stopped, SoX stops.

FILENAMES

Filenames can be simple file names, relative or absolute path names, URLs (for input files only) or special filenames. URL support requires one of the **wget**, **wget2** or **curl** programs to be installed.

Giving SoX an input or output filename that is the same as the name of a SoX effect does not work since SoX will treat it as an effect specification. You can work around this by calling the file **./chorus** on Unix or **.\chorus** on MS/DOS but it is not usually a problem since most audio filenames have a filename extension after a dot, which effect names do not.

Using the same file name as an input and an output is unlikely to work as intended because it is likely to truncate the file before reading all of it.

Special Filenames

The following filenames may be used in certain circumstances in place of a normal filename:

– SoX can be used in simple pipeline operations by using the special filename '–' which, if used as an input filename, causes SoX to read audio data from the 'standard input' (stdin) and, if used as the output filename, will cause SoX to send audio data to the 'standard output' (stdout). When using this option for the output file, and sometimes when using it for an input file, the file type (see **–t** below) must also be given.

"|*program* [*options*] ..."

An initial 'pipe' character specifies that the given command's standard output (stdout) should be used as an input file. Unlike the special filename **–**, this can be used for several inputs to one SoX command. For example, if a program `genw` generates a mono WAV signal on its standard output, the following command makes a stereo file from two generated signals:

```
sox_ng -M "|genw --imd -" "|genw --thd -" out.wav
```

For headerless (raw) audio and some other formats, **–t** needs to be given before the input command.

"*wildcard-filename*"

Specifies that filename 'globbing' (wildcard matching) should be performed by SoX instead of by the shell if the *wildcard-filename* contains the characters *****, **?** or characters ranges such as **[A-Z]**. This allows a single set of file options to be applied to a group of files. For example, if the current directory contains three files, `file1.vox`, `file2.vox` and `file3.vox`,

```
play_ng --rate 6k *.vox
```

is expanded by the shell (in most environments) to

```
play_ng --rate 6k file1.vox file2.vox file3.vox
```

which only treats the first 'vox' file as having a sample rate of 6k. With

```
play_ng --rate 6k "*.vox"
```

the given sample rate option is applied to all the files.

If you do not want SoX to glob the filenames, you can use the option **––no–glob** before each filename that should not be globbed, which is necessary if you need to process files whose names contain wildcard characters.

–p, **––sox–pipe**

This can be used in place of an output filename to specify that its output will be used as the input to another SoX command. For example, in the command:

```
play_ng "|sox_ng -n -p synth 2" "|sox_ng -n -p synth 2 tremolo 10"
```

play_ng thinks it's playing two files in succession that come from pipes, but in fact they both come from other invocations of SoX, each with different effects.

–p is in fact an alias for '**–t sox –**'.

−d, **−−default−device**

This can be used in place of an input or output filename to specify that the default audio device (if one has been built into SoX) is to be used. This is akin to invoking **rec_ng** or **play_ng** as described above.

−n, **−−null**

This can be used in place of an input or output filename to specify that a 'null file' is to be used. Here, 'null file' refers to a SoX-specific mechanism and is not related to any operating system mechanism with some special name.

Using a null file as an input is equivalent to using a normal audio file that contains an infinite amount of silence and, as such, is not generally useful unless used with an effect that specifies a finite time length such as **trim** or **synth**.

Using a null file as an output amounts to discarding the audio and is mainly useful with effects that produce information about the audio instead of affecting it such as **noiseprof**, **stat** and **spectrogram**.

The sampling rate associated with a null file is by default 48kHz but, as with a normal file, this can be overridden using format options such as **−r** (see below).

Supported File and Audio Device Types

See **soxformat_ng**(7) for a list and description of the supported file formats and audio device drivers.

OPTIONS

Global Options

These options can be specified on the command line at any point before the first effect name.

The **SOX_OPTS** environment variable can be used to provide alternative default values for SoX's global options. See **ENVIRONMENT** (below).

−−buffer *bytes*, **−−input−buffer** *bytes*

Set the size in bytes of the buffers used for processing audio (default 8192). **−−buffer** applies to input, effects and output processing; **−−input−buffer** applies only to input processing, for which it overrides **−−buffer** if both are given.

Large values for **−−buffer** may cause SoX to be become slow to respond to requests to terminate or to skip to the next input file.

−−clobber

Don't prompt before overwriting an existing file that has the same name as an output file. This is the default behaviour; to override it, use **−−no−clobber**.

−−combine concatenate|merge|mix|mix−power|multiply|sequence

Select the input file combining method. See **Input File Combining** above for a description of them.

−D, **−−no−dither**

Disable automatic dither—see **Dithering** above. This may be useful to ensure that SoX produces the same output in successive runs or if a file has been converted from 16 to 24 bit with the intention of doing some processing on it, but in fact no processing is needed after all and the original 16 bit file has been lost, in which case no dither is needed when converting the file back to 16 bits. See the **stats** effect for how to determine the actual bit-depth of the audio within a file.

−−effects−file *filename*

Read a file to obtain all effects and their arguments. The file is parsed as if the values were specified on the command line. A new line can be used in place of the special **:** marker to separate effect chains. For convenience, such markers at the end of the file are normally ignored; if you want to specify an empty last effects chain, use an explicit **:** by itself on the last line of the file. This option causes any effects specified on the command line to be discarded.

−G, **−−guard**
Automatically invoke the **gain** effect to guard against clipping. E.g.

```
sox_ng -G in.au -b 16 out.au rate 44100 dither -s
```

is shorthand for

```
sox_ng in.au -b 16 out.au gain -h rate 44100 gain -rh dither -s
```

See **−V, −−norm,** and the **gain** effect.

−h, **−−help**
Show SoX's version number and usage information.

−−help−effect *name*
Show usage information for the specified effect. The name **all** can be used to show it for all available effects.

−−help−format *name*
Show information about the specified file format. The name **all** can be used to show information for all supported formats.

−−i, **−−info**
If given as the first parameter to **sox_ng**, behave as **soxi_ng**.

−m|**−M**
Equivalent to **−−combine mix** and **−−combine merge** respectively.

−−magic
If SoX has been built with the optional 'libmagic' library, this option enables its use in helping to detect audio file types.

−−multi−threaded | −−single−threaded
By default, SoX is 'multi threaded' and processes audio channels for most multichannel effects in parallel on hyperthreading and multicore processors.

If the **−−single−threaded** option is given, it processes them one at a time, which may be more efficient on systems with less RAM.

A larger buffer size than the default may be needed to benefit more from multithreaded processing (e.g. 131072; see **−−buffer** above).

−−no−clobber
Prompt before overwriting an existing file with the same name as that given for the output file.

Unintentionally overwriting a file is easier than you might think, for example, if you accidentally enter

```
sox_ng file1 file2 effect1 effect2 ...
```

when what you really meant was

```
play_ng file1 file2 effect1 effect2 ...
```

then, without this option, file2 will be overwritten. Hence, using this option is recommended and can be set in the **SOX_OPTS** environment variable (see **ENVIRONMENT** below).

−−norm[**=***dB-level*]
Automatically invoke the **gain** effect to guard against clipping and to normalize the audio. E.g.

```
sox_ng --norm in.au -b 16 out.au rate 44100 dither -s
```

is shorthand for

```
sox_ng in.au -b 16 out.au gain -h rate 44100 gain -nh dither -s
```

Optionally, the audio can be normalized to a given level, usually below 0 dBFS:

```
sox_ng --norm=-3 in.au out.au
```

See **−V**, **−G** and the **gain** effect.

−−play−rate−arg *arg*
Selects a quality option to be used when the **rate** effect is invoked automatically when playing audio. This option is typically set via the **SOX_OPTS** environment variable (see **ENVIRONMENT** below) and its default value, when playing, is **−l** (low quality but fast). See the **rate** effect for other alternatives.

−−plot gnuplot|octave|off
If not set to **off** (the default if **−−plot** is not given), run in a mode that can be used in conjunction with the **gnuplot** program or the GNU Octave program to assist with the selection and configuration of many of the transfer function-based effects. For the first given effect that supports the selected plotting program, SoX outputs commands to plot the effect's transfer function and stops without actually processing any audio. E.g.

```
sox_ng --plot octave input-file -n highpass 1320 > highpass.plt
octave highpass.plt
```

−q, **−−no−show−progress**
Run in quiet mode when SoX wouldn't otherwise do so. This is the opposite of the **−S** option. To suppress error and warning messages, see **−V** below.

−R Run in 'repeatable' mode. When this option is given, SoX embeds a time stamp in the output file if its format supports comments and will seed pseudo random number generators, as used by **dither**, with that number, ensuring that successive SoX invocations with the same inputs and the same parameters yield the same output.

−−replay−gain track|album|off
Select whether or not to apply replay gain adjustment to input files. The default is **off** for **sox_ng** and **rec_ng**, **album** for **play_ng** when (at least) the first two input files are tagged with the same Artist and Album names and **track** for **play_ng** otherwise.

−S, **−−show−progress**
Display input file format/header information, processing progress as a percentage of the input file(s), elapsed time, remaining time (if known, in brackets) and the number of samples written to the output file. Also shown is a peak level meter, and an indication of whether clipping has occurred. The peak level meter shows up to two channels and is calibrated for digital audio as follows:

dB FSD	*Display*	*dB FSD*	Display
−25	−	−11	====
−23	=	−9	====−
−21	=−	−7	=====
−19	==	−5	=====−
−17	==−	−3	======
−15	===	−1	=====!
−13	===−		

A three-second peak-held value of the headroom in dBs is shown to the right of the meter if the headroom is less than 6dB.

This option is enabled by default when using SoX to play or record audio but can be disabled with **−q**.

−T Equivalent to **−−combine multiply**

−−temp *directory*
Specify that any temporary files should be created in the given directory. This can be useful if there are permission or free space problems with the default location. In this case, using '**−−temp**

.’ (to use the current directory) is often a good solution.

−−version
Show SoX’s version number and exit.

−V[*level*]
Set verbosity. This is particularly useful for seeing how any automatic effects have been invoked by SoX.

SoX displays messages on the console (stderr) according to the following verbosity levels:

0 No messages are shown at all; use the exit status to determine if an error has occurred.

1 Only error messages are shown. These are generated if SoX cannot complete the requested commands.

2 Warning messages are also shown. These are generated if SoX can complete the requested commands but not exactly according to the requested command parameters, or if clipping occurs. This is the default.

3 Descriptions of SoX’s processing phases are also shown, to see exactly how SoX is processing your audio.

4 to 6 Messages to help with debugging SoX are also shown.

Each occurrence of the **−V** option increases the verbosity level by 1. Alternatively, the verbosity level can be set to an absolute number by specifying it immediately after the **−V**, e.g. **−V0** shuts it up.

Input File Options

These options apply to the first input filename that follows them on the command line.

−−ignore−length
Override the audio length given in an audio file’s header. If this option is given, SoX keeps reading audio until it reaches the end of the input file.

−v, **−−volume** *factor*
Intended for use when combining multiple input files, this option adjusts the volume of the file that follows it on the command line by a factor of *factor*. This allows it to be ‘balanced’ w.r.t. the other input files. This is a linear (amplitude) adjustment, so a number less than 1 decreases the volume and a number greater than 1 increases it. If a negative number is given then, in addition to the volume adjustment, the audio signal will be inverted.

See the **norm**, **vol** and **gain** effects, **Input File Balancing** above and the **Special Filenames**’ section on *wildcard-filenames*.

Input & Output File Format Options

These options apply to the input or output file whose name they immediately precede on the command line and are used mainly when working with headerless file formats or when specifying a format for the output file that is different from that of the input file.

−b *bits*, **−−bits** *bits*
Set the number of bits (a.k.a. bit-depth or word length) in each encoded sample. It is not applicable to complex encodings such as MP3 or GSM and not necessary with encodings that have a fixed number of bits such as A-law, μ-law and ADPCM.

For an input file, the most common use for this option is to inform SoX of the number of bits per sample in a ‘raw’ (‘headerless’) audio file. For example

```
sox_ng −r 16k −e signed −b 8 input.raw output.wav
```

converts a particular ‘raw’ file to a self-describing ‘WAV’ file.

For an output file, this option can be used to set the output encoding size. By default, the output encoding size is be set to the input encoding size, provided it is supported by the output file type.

For example:

```
sox_ng input.cdda −b 24 output.wav
```

converts raw CD digital audio (16-bit, signed-integer) to a 24-bit (signed-integer) ‘WAV’ file.

−c *CHANNELS*, **−−channels** *CHANNELS*

Sets the number of audio channels in the audio file. This can be any number greater than zero.

For an input file, the most common use for this option is to inform SoX of the number of channels in a ‘raw’ (headerless) audio file. Occasionally, it may be useful to use this option with a headered file to override the (presumably incorrect) value in the header but this is only supported with certain file types. For example:

```
sox_ng −r 48k −e float −b 32 −c 2 input.raw output.wav
```

converts a ‘raw’ file to a self-describing ‘WAV’ file.

```
play_ng −c 1 music.wav
```

interprets the file data as belonging to a single channel regardless of what is indicated in the file header and if the file does in fact have two channels, it is played at half speed.

For an output file, this option provides a shorthand for specifying that the **channels** effect should be invoked in order to change (if necessary) the number of channels in the audio signal to the number given. For example, the following two commands are equivalent:

```
sox_ng input.wav −c 1 output.wav bass −b 24
sox_ng input.wav      output.wav bass −b 24 channels 1
```

though the second form is more flexible as it allows effects to be ordered arbitrarily.

−e *ENCODING*, **−−encoding** *ENCODING*

Set the audio encoding type, sometimes needed with file types that support more than one encoding scheme such as raw, WAV or AU but not with MP3 or FLAC. The available encoding types are as follows:

signed−integer
PCM data stored as signed (‘two’s complement’) integers. Commonly used with a 16 or 24 −bit encoding size. A value of 0 represents minimum signal power.

unsigned−integer
PCM data stored as unsigned integers. Commonly used with an 8-bit encoding size. A value of 0 represents maximum signal power.

floating−point
PCM data stored as IEEE 753 single precision (32-bit) or double precision (64-bit) floating point (‘real’) numbers. A value of 0 represents minimum signal power.

a−law International telephony standard for logarithmic encoding to 8 bits per sample. It has a precision equivalent to roughly 13-bit PCM and is sometimes encoded with reversed bit-ordering (see the **−X** option).

u−law/mu−law
The North American telephony standard for logarithmic encoding to 8 bits per sample, a.k.a. μ-law has a precision equivalent to roughly 14-bit PCM and is sometimes encoded with reversed bit-ordering (see the **−X** option).

oki−adpcm
OKI (a.k.a. VOX, Dialogic, or Intel) 4-bit ADPCM has a precision equivalent to roughly 12-bit PCM. ADPCM is a form of audio compression that makes a good compromise between audio quality and encoding/decoding speed.

ima−adpcm
IMA (a.k.a. DVI) 4-bit ADPCM has a precision equivalent to roughly 13-bit PCM.

ms-adpcm
Microsoft 4-bit ADPCM has a precision equivalent to roughly 14-bit PCM.

gsm−full−rate
GSM is currently used for the majority of the world's digital wireless telephone calls. It utilizes several audio formats with different bit rates and associated speech quality. SoX has support for GSM's original 13kbps 'Full Rate' audio format.

Encoding names can be abbreviated where this would not be ambiguous; e.g. **unsigned−integer** can be given as **un**, but not **u** (ambiguous with **u-law**).

For an input file, the most common use for this option is to inform SoX of the encoding of a 'raw' ('headerless') audio file (see the examples in **−b** and **−c** above).

For an output file, this option can be used (perhaps with **−b**) to set the output encoding. For example:

```
sox_ng input.cdda −e float output1.wav
sox_ng input.cdda −b 64 −e float output2.wav
```

converts a raw CD digital audio (16-bit signed integer) to floating point 'WAV' files of single and double precision respectively.

If this option is not given, the output encoding will be the same as the input encoding, provided it is supported by the output file type.

−−no−glob
Specifies that filename 'globbing' (wildcard matching) should not be performed by SoX on the following filename. For example, if the current directory contains the two files 'five-seconds.wav' and 'five*.wav', then

```
play_ng --no-glob "five*.wav"
```

can be used to play just the single file 'five*.wav'.

−r, −−rate *rate*[**k**]
Gives the sample rate in Hz (or kHz if followed by **k**) of the file.

For an input file, the most common use for this option is to inform SoX of the sample rate of a 'raw' ('headerless') audio file (see the examples in **−b** and **−c** above). Occasionally it may be useful to use this option with a 'headered' file to override the value in the header, though this is only supported with certain file types. For example, if audio was recorded with a sample rate of 48k from a source that played back a little too slowly, say 1·5%,

```
sox_ng −r 48720 input.wav output.wav
```

would correct the speed by changing only the file header (but see the **speed** effect for the more usual solution to this problem).

For an output file, this option provides a shorthand for specifying that the **rate** effect should be invoked in order to change (if necessary) the sample rate of the audio signal to the given value. For example, the following two commands are equivalent:

```
sox_ng input.wav −r 48k output.wav bass −b 24
sox_ng input.wav        output.wav bass −b 24 rate 48k
```

though the second form is more flexible as it allows **rate** options to be given and allows the effects to be ordered arbitrarily.

−t, **−−type** *FILE-TYPE*
Give the type of an audio file. For both input and output files, this option is commonly used to inform SoX of the type a 'headerless' audio file where the actual/desired type cannot be determined

from the filename extension. For example:

```
another-command | sox_ng -t mp3 - output.wav
sox_ng input.wav -t raw output.bin
```

It can also be used to override the type implied by an input filename extension but, if overriding with a type that has a header, SoX exit with an error message if such a header is not actually present.

There are also pseudo filetypes that tell SoX to use a specified format module that handles more than one type of audio file such as **−t sndfile** and **−t ffmpeg** or when a type of file can be handled by several different format modules, such as WAV files containing MP3-encoded data.

Furthermore, the file type can be used to select a particular audio device driver for recording and playing.

See **soxformat_ng**(7) for a list of supported file types.

−L, **−−endian little**
−B, **−−endian big**
−x, **−−endian swap**

These options specify whether the byte order of the audio data is, respectively, 'little endian', 'big endian' or the opposite to that of the system on which SoX is being used. Endianness applies only to data encoded as floating point or as signed or unsigned integers of 16 or more bits. It is often necessary to specify one of these options for headerless files and sometimes necessary for (otherwise) self-describing files. A given endian-setting option may be ignored for an input file whose header contains a specific endianness identifier or for an output file that is actually an audio device.

N.B. Unlike other format characteristics, the endianness (byte, nibble, & bit ordering) of the input file is not automatically used for the output file. For example, when the following is run on a little-endian system:

```
sox_ng -B audio.s16 trimmed.s16 trim 2
```

`trimmed.s16` will be created as little-endian;

```
sox_ng -B audio.s16 -B trimmed.s16 trim 2
```

must be used to preserve big-endianness in the output file.

The **−V** option can be used to check the selected orderings.

−N, **−−reverse−nibbles**

Specifies that the nibble ordering of the samples (i.e. the 2 halves of a byte) should be reversed, which is sometimes useful with ADPCM-based formats.

See the N.B. in the section on **−x** above.

−X, **−−reverse−bits**

Specifies that the bit ordering of the samples should be reversed, which is sometimes useful with a few (mostly headerless) formats.

See the N.B. in the section on **−x** above.

Output File Format Options

These options only apply to output files and may only precede an output filename on the command line.

−−add−comment *TEXT*

Append a comment to the output file header (where applicable).

−−comment *TEXT*

Specify the comment text to store in the output file header (where applicable).

SoX provides a default comment 'Processed by SoX' if this option (or **−−comment−file**) is not given. To specify that no comment should be stored in the output file, use **−−comment "" .**

−−comment−file *FILENAME*
Specify a file containing the comment text to store in the output file header (where applicable).

−C, **−−compression** *FACTOR*
Set the compression factor for variably-compressed output file formats. If this option is not given then a default compression factor applies. The compression factor is interpreted differently for different compressed file formats; see the description of the file formats that use this option in **sox-format_ng**(7) for more information.

EFFECTS

In addition to converting, playing and recording audio files, SoX can be used to invoke a number of audio effects. Multiple effects may be applied by specifying them one after the other at the end of the SoX command line, forming an 'effects chain'. Note that applying multiple effects in real time (i.e. when playing audio) may require a high performance computer.

Some of the SoX effects are primarily intended to be applied to a single instrument or 'voice'. To facilitate this, the **remix** effect and the global SoX option **−M** can be used to isolate then recombine tracks from a multitrack recording.

Multiple Effects Chains

A single effects chain is made up of one or more effects. Audio from the input runs through the chain until either the end of the input file is reached or an effect terminates the chain.

SoX supports running multiple effects chains over the input audio. In this case, when one chain indicates that it is done processing audio, the audio data is sent through the next effects chain. This continues until either no more effects chains exist or the input has reached the end of the file.

Effects chains can be separated by placing a **:** (colon) after an effect; any following effects are a part of a new effects chain.

It is important to place the effect that stops the chain as the first effect in the chain because any samples that are buffered by effects to the left of the terminating effect will be discarded. The amount of samples discarded is related to the **−−buffer** option and it should be kept small, relative to the sample rate, if the terminating effect cannot be first. Further information on stopping effects can be found in the **Stopping SoX** section.

There are a few pseudo-effects that can help when using multiple effects chains. These include **newfile**, which starts writing to a new output file before moving to the next effects chain, and **restart**, which moves back to the first effects chain. Pseudo-effects must be specified as the first effect in a chain and as the only effect in a chain (i.e. they must have a **:** before and after them).

Here is an example of multiple effects chains. It splits the input file into multiple files, each of 30 seconds in length and each output filename will have unique number in its name, as documented in the **Output Files** section.

```
sox_ng in.au out.au trim 0 30 : newfile : restart
```

Common Notation And Parameters

In the descriptions that follow, [square brackets] are used to denote parameters that are optional, {braces} to denote those that are both optional and repeatable, <angle brackets> to denote those that are repeatable but not optional and pipe characters '|' separate options from which to choose one of several alternatives. Where applicable, default values for optional parameters are shown (in parentheses).

The following parameters are used with, and have the same meaning for, several effects:

frequency
A frequency in Hz or, if followed by **k**, in kHz or, if preceded by **%**, in semitones relative to A (440Hz); alternatively, scientific note names (e.g. E2) may be used.

gain A power gain in dB. Zero gives no gain, less than zero gives an attenuation and greater than zero amplifies.

duration
See **Time Specifications** below.

position
A position within the audio stream; the syntax is [**=**|**−**|**+**]*timespec*, where *timespec* is a time specification (see below). The optional first character indicates whether the *timespec* is to be interpreted relative to the start (**=**) or end (**−**) of the audio or relative to the previous *position* (**+**) if the effect accepts multiple positional arguments. The audio length must be known for end-relative locations to work, though some effects do accept **−0** for the end of the audio even if the length is unknown. Which of **=**, **−** and **+** is the default depends on the effect and is shown in its syntax as, e.g., *position*(+).

Examples: '=2:00' is two minutes into the audio stream, '−100s' is one hundred samples before the end of the audio, '+0:12+10s' is twelve seconds and ten samples after the previous position and '−0.5+1s' is one sample less than half a second before the end of the audio.

width[**h**|**k**|**o**|**q**]
Used to specify the bandwidth of a filter. A number of different methods to specify the width are available (though not all for every effect). One of the characters shown may be appended to select the desired method as follows:

	Method	*Notes*
h	Hz	
k	kHz	
b	Hz	Old non-frequency-warped response
o	octaves	
q	Q-factor	See [2]
s	slope	

For each effect that uses this parameter, the default method (if no character is appended) is the one that is listed first in the first line of the effect's description.

Time Specifications

A *timespec* can be given in one the following two forms:

[[*hours*:]*minutes*:]*seconds*[**.***frac*][**t**]
For example, a time specification of '1:30.5' corresponds to one minute, thirty and ½ seconds. The component values do not have to be normalized; e.g. '1:23:45', '83:45', '79:0285', '1:0:1425', '1::1425' and '5025' are all equivalent.

*samples***s**
Specifies the number of samples directly, as in '8000s'. For large sample counts, *e notation* is supported: '1.7e6s' is the same as '1700000s'.

Time specifications can also be chained with **+** or **−** into a new time specification where the right part is added to or subtracted from the total so far. For example, '3:00−200s' means two hundred samples less than three minutes.

If a *time specification* is a plain whole number with no **t** or **s** suffix, whether it is taken as a number of seconds or a number of samples depends on the effect in question. At present, it always means seconds except for the *duration* parameters of the **silence** effect.

Supported Effects

To see whether SoX has support for an optional effect, enter **sox_ng −h** and look for its name in the **EFFECTS** list; a categorized list of the effects can be found in the accompanying README file.

allpass [**−1**|**−2**] *frequency* [*width*[**h**|**k**|**o**|**q**]]
Apply a two-pole all-pass filter with central frequency *frequency* and filter width *width*. An all-pass filter changes the audio's frequency to phase relationship without changing its frequency to

amplitude relationship. The filter is described in detail in [1].

-1 or **-2** use an experimental 1-pole or 2-pole filter, in which case *width* does not apply.

This effect supports the **−−plot** global option.

band [**−n**] *frequency* [*width*[**h**|**k**|**o**|**q**]]

Apply a band-pass filter. The frequency response drops logarithmically around the center *frequency*. The *width* parameter gives the slope of the drop: the frequencies at *frequency* + *width* and *frequency* − *width* will have half their original amplitudes. Its default value is half of the center frequency.

band defaults to a mode oriented to pitched audio, i.e. voice, singing or instrumental music. The **−n** (for noise) option uses the alternate mode for unpitched audio (e.g. percussion), though **−n** introduces a power gain of about 11dB in the filter, so beware of output clipping. **band** introduces noise in the shape of the filter, peaking at the center frequency and settling around it.

This effect supports the **−−plot** global option.

See **sinc** for a band-pass filter with steeper shoulders.

bandpass|**bandreject** [**−c**] *frequency width*[**h**|**k**|**o**|**q**|**b**]

Apply a two-pole Butterworth band-pass or band-reject filter with central frequency *frequency*, and (3dB-point) bandwidth *width*. The **−c** option applies only to **bandpass** and selects a constant skirt gain (peak gain = Q) instead of the default, a constant 0dB peak gain. The filters roll off at 6dB per octave (20dB per decade) and are described in detail in [1].

These effects support the **−−plot** global option.

See **sinc** for a band-pass filter with steeper shoulders.

bass|**treble** *gain* [*frequency* [*width*[**s**|**h**|**k**|**o**|**q**]]]

Boost or cut the bass (lower) or treble (upper) frequencies of the audio using a two-pole shelving filter with a response similar to that of a standard hifi's tone controls. This is also known as shelving equalization.

gain gives the gain at 0Hz for **bass** or, for **treble**, whichever is the lower of ~22kHz and the Nyquist frequency. Its useful range is about −20 (for a large cut) to +20 (for a large boost). Beware of **Clipping** when using a positive *gain*.

The filter can be fine-tuned using the following optional parameters:

frequency sets the filter's central frequency and so can be used to extend or reduce the frequency range to be boosted or cut. The default values are 100Hz for **bass** and 3kHz for **treble**.

width determines how steep the filter's shelf transition is. In addition to the common width specification methods, 'slope' (the default) may be used. Its useful range is about 0·3 for a gentle slope to 1 (the maximum) for a steep slope; and its default value is 0·5.

The filters are described in detail in [1].

These effects support the **−−plot** global option.

See **equalizer** for a peaking equalization effect.

bend [**−f** *frame-rate*(25)] [**−o** *oversampling*(16)]
{*start-position(+)*,*cents*,*end-position(+)*}

Changes the pitch by specified amounts at specified times without changing the duration. Each given triple: *start-position*,*cents*,*end-position* specifies one bend. *cents* is the number of cents (100 cents = 1 semitone) by which to bend the pitch. The other values specify the points in time at which to start and end bending the pitch. During each bend, the frequency changes logarithmically, i.e. by the same number of cents per second.

The pitch bending algorithm uses the Discrete Fourier Transform (DFT) at a particular frame rate and oversampling rate. The **–f** (from 10 to 80) and **–o** (from 4 to 32) parameters may be used to adjust these parameters and thus control the smoothness of the changes in pitch.

For example, an initial tone is generated, then bent three times, yielding four different notes in total:

```
play_ng –n synth 2.5 sin 667 gain 1 \
   bend .35,180,.25  .15,740,.53  0,–520,.3
```

Here, the first **bend** runs from 0.35 to 0.6 seconds and the second one from 0.75 to 1.28 seconds. Note that the clipping that is produced in this example is deliberate; to remove it, use **gain –5** in place of **gain 1**.

See **pitch**.

biquad *b0 b1 b2 a0 a1 a2*

Apply a biquad Infinite Impulse Response filter with the given coefficients, where *b** and *a** are the numerator and denominator coefficients respectively.

See http://en.wikipedia.org/wiki/Digital_biquad_filter (where a0 = 1).

This effect supports the **––plot** global option.

channels *channels*

Invoke a simple algorithm to change the number of channels in the audio signal to the given number: mixing if decreasing the number of channels or duplicating if increasing the number of channels.

The **channels** effect is invoked automatically if SoX's **–c** option specifies a number of channels that is different to that of the input file(s). Alternatively, if this effect is given explicitly, SoX's **–c** option need not be given. For example, the following two commands are equivalent:

```
sox_ng input.wav –c 1 output.wav bass –b 24
sox_ng input.wav      output.wav bass –b 24 channels 1
```

though the second form is more flexible as it allows the effects to be ordered arbitrarily.

For example, when making a stereo file quadraphonic, the left and right channels are copied into the third and fourth and when mixing a four-channel file down to stereo, the left channel is the mix of the first and third and the right of the second and fourth.

See **remix** for an effect that allows channels to be mixed and selected arbitrarily.

chorus [–n|–l|–q] [-s|-t] [*gain-in [gain-out* {*delay [decay [speed [depth [***–s**|**–t**]]]]}]]

Add a chorus effect to the audio. This can make a single voice sound like a chorus but can also be applied to instrumentation.

Chorus resembles an **echo** effect with a short delay but, while **echo**'s delay is constant, **chorus**' delay varies by a sinusoidal or triangular modulation.

See [3] for further discussion of the chorus effect.

The **–l** flag makes **chorus** do linear interpolation between samples when the offset into the delay line is not a whole number, which is about 15% slower but makes it considerably less noisy and **–q** asks for quadratic interpolation which is about 40% slower but makes it even less noisy. **–n** explicitly asks for no interpolation, the default, fast and fuzzy.

–s or **–t** before the stages change the default wave type for all of them.

All parameters are all optional and, if missing, assume the following values:

Parameter	**Range**	**Default**	**Description**
gain-in	-1-1	0.5	Proportion of input delivered clean to the adder

gain-out	-1-1	1	Final volume adjustment
delay	0-1000	40-60	Fixed delay in milliseconds
decay	-1-1	0.5	Volume of delayed output
speed	0-192k	0.25	Modulation frequency
depth	0-1000	2	Extra delay in milliseconds
wave	-s|-t	-s	Sinusoidal/triangular modulation

Each delay ranges from the fixed *delay* to *delay* + *depth*.

Gain-out is then applied to the sum of the input scaled by *gain-in* and the outputs from the delays scaled by their *decay*s.

For internal reasons regarding the speed of **chorus**, there is a limit of 256 chorus stages.

A typical delay is around 40ms to 60ms; the modulation speed is best near 0·25Hz and the modulation depth around 2ms. For example, a single delay:

```
play_ng guitar1.wav chorus 0.7 0.9 55 0.4 0.25 2 -t
```

Two delays of the original samples:

```
play_ng guitar1.wav chorus 0.6 0.9 50 0.4 0.25 2 -t \
    60 0.32 0.4 1.3 -s
```

A fuller-sounding chorus (with three additional delays):

```
play_ng guitar1.wav chorus 0.5 0.9 50 0.4 0.25 2 -t \
    60 0.32 0.4 2.3 -t 40 0.3 0.3 1.3 -s
```

flanger can do everything that **chorus** does except multiple stages but works in floating point internally instead of integers so is slower without a floating point processor:

```
chorus -l gain-in gain-out delay decay speed depth -wave
```

is equivalent to

```
flanger delay depth 0 100×decay÷gain-in speed wave 0 \
   vol gain-out÷(gain-in+decay)
```

For a flow diagram of how **chorus** works, say **sox_ng −−help−effect chorus**.

compand *attack1***,***decay1*{**,***attack***,***decay*}
[*soft-knee-dB***:**]*in-dB1*[**,***out-dB1*]{**,***in-dB***,***out-dB*}
[*gain* [*initial-volume-dB* [*delay*]]]

Compand (compress or expand) the dynamic range of the audio.

The *attack* and *decay* parameters (in seconds) determine the time over which the instantaneous level of the input signal is averaged to determine its volume; attacks refer to increases in volume and decays refer to decreases. For most situations, the attack time (its response to the music getting louder) should be shorter than the decay time because the human ear is more sensitive to sudden loud music than sudden soft music. When more than one pair of attack/decay parameters is specified, each input channel is companded separately and the number of pairs must agree with the number of input channels. Typical values are 0·3,0·8 seconds.

The second parameter is a list of points on the compander's transfer function specified in dB relative to the maximum possible signal amplitude. The input values must be in a strictly increasing order but the transfer function does not have to be monotonically rising. If omitted, the value of *out-dB1* defaults to the same value as *in-dB1*; levels below *in-dB1* are not companded but may have gain applied to them. The point '0,0' is assumed but may be overridden by '0,*out-dBn*'. If the list is preceded by a *soft-knee-dB* value, then the points at where adjacent line segments on the transfer function meet are rounded by the amount given. Typical values for the transfer function are '6:−70,−60,−20'.

The third (optional) parameter is an additional gain in dB to be applied at all points on the transfer function and allows easy adjustment of the overall gain.

The fourth (optional) parameter is an initial level to be assumed for each channel when companding starts. This lets you supply a nominal level initially so that, for example, a very large gain is not applied to initial signal levels before the companding action has begun to operate: it is quite probable that in such an event, the output would be severely clipped while the compander gain adjusts itself. A typical value (for audio which is initially quiet) is **−90** dB.

The fifth (optional) parameter is a delay in seconds. The input signal is analyzed immediately to control the compander, but it is delayed before being fed to the volume adjuster. Specifying a delay approximately equal to the attack/decay times allows the compander to operate in a predictive rather than a reactive mode. A typical value is 0·2 seconds.

*   *   *

The following example might be used to make a piece of music with both quiet and loud passages suitable for listening to in a noisy environment such as a moving vehicle:

```
sox_ng asz.wav asz-car.wav compand 0.3,1 6:-70,-60,-20 -5 -90 0.2
```

The transfer function (‘6:−70,...’) says that very soft sounds (below −70dB) remain unchanged. This stops the compander from boosting the volume on ‘silent’ passages such as between movements. However, sounds in the range −60dB to 0dB (maximum volume) are boosted so that the 60dB dynamic range of the original music is compressed 3-to-1 into a 20dB range, which is wide enough to enjoy the music but narrow enough to get around the road noise. The ‘6:’ selects 6dB soft-knee companding. The −5 dB output gain is needed to avoid clipping (the number is inexact and was derived by experimentation). The −90 dB for the initial volume will work fine for a clip that starts with near silence and the delay of 0·2 seconds makes the compander react more quickly to sudden volume changes.

In the next example, **compand** is used as a noise-gate for when the noise is at a lower level than the signal:

```
play_ng in.au compand .1,.2 -inf,-50.1,-inf,-50,-50 0 -90 .1
```

Here is another noise-gate, this time for when the noise is at a higher level than the signal (making it, in some ways, similar to a squelch effect):

```
play_ng in.au compand .1,.1 -45.1,-45,-inf,0,-inf 45 -90 .1
```

This effect supports the **−−plot** global option (for the transfer function).

For a flow diagram of how **compand** works, say **sox_ng −−help−effect compand**.

See **mcompand** for a multiple-band companding effect.

contrast [*enhancement-amount*(75)]

Comparable with compression, this effect modifies an audio signal to make it sound louder. *enhancement-amount* controls the amount of the enhancement and is a number in the range 0−100. Note that *enhancement-amount* = 0 still gives a significant contrast enhancement.

See the **compand** and **mcompand** effects.

dcshift *shift* [*limiter−gain*]

Apply a DC shift to the audio. This can be useful to remove a known DC offset (caused perhaps by a hardware problem in the recording chain) from the audio. The effect of a DC offset is reduced headroom and hence volume. The **stat** or **stats** effect can be used to determine if a signal has a DC offset.

The given *dcshift* value is a floating point number in the range of ±2 that indicates the amount to shift the audio (which is in the range of ±1).

An optional *limiter−gain* can be specified as well. It should have a value much less than 1 (e.g.

0·05 or 0·02) and is used only on peaks to prevent clipping.

An alternative approach to removing a DC offset (albeit with a short delay) is to use the **highpass** filter effect at a frequency of say 10Hz, as illustrated in the following example:

```
sox_ng −n dc.wav synth 5 sin %0 50
sox_ng dc.wav fixed.wav highpass 10
```

deemph

Apply Compact Disc (IEC 60908) de-emphasis with a treble attenuation shelving filter.

Pre-emphasis was applied in the mastering of some CDs issued in the early 1980s. These included many classical music albums, as well as now sought-after issues of albums by The Beatles, Pink Floyd and others. Pre-emphasis should be removed at playback time by a de-emphasis filter in the playback device. However, not all modern CD players have this filter and very few PC CD drives have it; playing pre-emphasized audio without the correct de-emphasis filter results in audio that sounds harsh and is far from what its creators intended.

With the **deemph** effect, it is possible to apply the necessary de-emphasis to audio that has been extracted from a pre-emphasized CD and then either burn the de-emphasized audio to a new CD (which will then play correctly on any CD player) or simply play the correctly de-emphasized audio files on the PC. For example:

```
sox_ng track1.wav track1−deemph.wav deemph
```

and then burn track1-deemph.wav to CD, or

```
play_ng track1−deemph.wav
```

or simply

```
play_ng track1.wav deemph
```

The de-emphasis filter is implemented as a biquad and requires the input audio sample rate to be either 44.1kHz or 48kHz. Its maximum deviation from the ideal response is only 0·06dB (up to 20kHz).

This effect supports the **−−plot** global option.

delay {*position(=)*}

Delay zero or more audio channels such that they start at the given *position*.

For example, **delay 1·5 +1 3000s** delays the first channel by 1·5 seconds, the second channel by 2·5 seconds (one second more than the previous channel), the third channel by 3000 samples and leaves other channels undelayed. The following (one long) command plays a chime sound:

```
play_ng −n synth −j 3 sin %3 sin %−2 sin %−5 sin %−9 \
   sin %−14 sin %−21 fade h .01 2 1.5 delay \
   1.3 1 .76 .54 .27 remix − fade h 0 2.7 2.5 norm −1
```

and this an arpeggiated guitar chord:

```
play_ng −n synth pl G2 pl B2 pl D3 pl G3 pl D4 pl G4 \
   delay 0 .05 .1 .15 .2 .25 remix − fade 0 4 .1 norm −1
```

With no parameters it does nothing. To delay all channels by the same amount, use the **pad** effect.

dither [**−S**|**−s**|**−f** *filter*] [**−a**] [**−p** *precision*]

Apply dithering to the audio. Dithering deliberately adds a small amount of noise to the signal in order to mask audible quantization effects that can occur if the output sample size is less than 24 bits. With no options, this effect adds TPDF white noise.

The **−S** option selects a slightly ‘sloped’ TPDF, biased towards higher frequencies. It can be used at any sampling rate but, below ≈22kHz, plain TPDF is probably better and, above ≈ 37kHz, noise-shaping (if available) is probably better.

The **−s** option enables noise-shaping with the **shibata** filter (the same as **−f shibata**) and with the **−f** option it is possible to select a particular noise-shaping filter from the following list: **lipshitz**, **f-weighted**, **modified-e-weighted**, **improved-e-weighted**, **gesemann**, **shibata**, **low-shibata**, **high-shibata**, **shibata-**(**A**|**B**)(**0**|**1**|**2**|**3**|**4**|**5**|**6**) and **shibata-A-saturated**. The latter **shibata-** ones use the new shaper coefficients from Naoki Shibata's **SSRC** package, described at https://shibatch.org/ssrc

The filter types are distinguished by the following properties: audibility of noise, level of (inaudible, but in some circumstances problematic) shaped high frequency noise and processing speed and they are available for the following sample rates:

Filter	*Sample rates*
lipshitz	44100
e- and f-weighted	48000
gesemann	44100, 48000
shibata	8000, 11025, 16000, 22050 32000, 37800, 44100, 48000
low-shibata	44100, 48000
high-shibata	44100
shibaba-A0 and A1	8000, 11025, 22050, 44100, 48000, 88200, 96000, 192000
shibata-A2	44100, 48000, 88200, 96000, 192000
shibata-A3 to A6	44100, 48000
shibata-B0 to B6	44100, 48000
shibata-A-saturated	8000, 11025, 22050

The **−a** option enables a mode where dithering (and noise-shaping if applicable) are automatically enabled only when needed. The most likely use for this is when applying fade in or out to an already dithered file, so that the redithering applies only to the faded portions. However, auto dithering is not foolproof, so the fades should be checked carefully for any noise modulation; if this occurs, then either redither the whole file or use **trim** and **fade** and concatenate the results.

The **−p** option overrides the target precision in bits and can be from 1 to 24.

If the SoX global option **−R** option is not given, the pseudo-random number generator used to generate the white noise is reseeded, i.e. the generated noise will be different on every invocation.

If the target precision is 1-bit, the **sdm** effect is applied automatically with default settings. Invoke it manually to control its options.

See the above section on **Dithering**.

dolbyb [**−e**|**−d**] [**−u** *upsamp*] [**−h**] [**−t** *gain*(1.0)] [**−a prec**(-5.0)] [**−f** {**1**|**2**|**3**|**4**}]

dolbyb is a Dolby B decoder/encoder based on dolbybcsoftwaredecode which simulates the operation of a Dolby B en/decoder's electronic circuit.

By default, **dolbyb** applies Dolby B decoding to its input signal; with **−e** it does Dolby B encoding. **−d** is also accepted but only for symmetry, as it is the default mode of operation.

−u sets the upsampling ratio to use in the sliding filter. Digital filtering only works well if the sample rate is well above the cutoff frequency of the filter. For Dolby B's sliding filter, that frequency can be as high as 34kHz and this does not work well if the sample rate is only 44.1Khz. To get around this, it upsamples the audio to a higher rate when it passes through this filter. By default, **−u0**, the upsampling rate is set automatically so that the upper sample rate is at least 200Khz; upsampling can be switched off with **−u1**.

If **−h** is given, upsampling is used throughout the effect from when the audio enters to when it leaves, not just in the sliding filter. As **dolbyb**'s up/downsampling algorithm is simple (repeating and averaging samples) you may obtain higher quality results by upsampling with **rate** before **dolbyb −u1** and downsampling it afterwards.

−t ("threshold") adjusts the gain when the audio is fed to the Dolby gain control circuits. When a tape deck is encoding or decoding a magnetic tape, it knows the signal level at the tape heads but with audio files the maximum signal level may not accurately represent the tape's maximum flux density (200nWb/m for cassette tapes), giving erroneous results. The **−t** option adjusts the volume level at which the sliding filter reacts to overcome this. Its default value is 1.0, which assumes that the maximum amplitude of the signal represents the maximum recording level on tape; higher values assume that it was recorded too quietly and values below 1.0 are for when it was recorded too loud.

To begin with, when you have little idea of what level to use, try a wide range of levels like 5, 10, 15 and 20. If the result sounds muffled, the threshold is too low and if it seems to have too much treble, the threshold is too high. Once you know the approximate level, you can try more closely-spaced levels and listen carefully to find the best level possible. Logic would suggest listening to where tracks fade out, to see if the treble increases, but this method doesn't seem to work well and the best way seems to be to see how low the level can be set before the results sound dull and muffled, then choose a level a bit higher than this; you can just about hear the difference between results that differ in threshold setting by about 2.

In decode mode, the program has to use trial and error to get the right output sample values. **−a** sets how accurate it needs to be before it is considered OK. A figure of 0.0 dB would mean an accuracy of about 1 sample value. The default is -5.0 dB, which is accurate to less than one sample value.

−f selects one of four types of filter to use. The program originally simulated an analog circuit for a Dolby B noise reducer. However, too much filtering in the side path was altering the phase of the side path audio, which caused problems when the side path was recombined with the main signal. Basically signals don't add together very well if there is too much difference in the phase. To fix this, there are now 4 filter modes with hopefully less of a phase change:

−f1 is the original method.

−f2 is a newer method that seems to work better than 1.

−f3 is another rearrangement which in practice doesn't seem to be any better than 1.

−f4 seems to work best, hence it is the default mode.

For further detail on these parameters and advice on digitizing and processing Dolby B-encoded tapes, consult the wiki pages at https://codeberg.org/sox_ng/libdolbyb

dop DSD over PCM. 1-bit DSD data is packed into 24-bit samples for transport over non-DSD-aware links.

downsample [*factor*(2)]

Downsample the signal by an integer factor: Only the first of each *factor* samples is retained, the others are discarded.

No decimation filter is applied. If the input is not a properly band-limited baseband signal, aliasing will occur. This may be desirable, e.g., for frequency translation.

The new lower sample rate propagates forward in the effects chain but, unless you specify the new sample rate with **−r** before the output filename or with a final (no-op) **rate** effect, it will be resampled back up to the original sample rate.

For a general resampling effect with antialiasing, see **rate**. See **upsample**.

earwax

This effect takes a 44.1kHz stereo signal and adds audio cues that, when listened to on headphones, move the sound stage from inside your head to outside and in front of you, as if listening to loudspeakers.

To see how **earwax** works, say **sox_ng −−help−effect earwax**.

echo *gain-in gain-out* <*delay decay*>

Add echoes to the audio. In nature, echoes are reflected sound and digital echo effects emulate this and are often used to help fill out the sound of a single instrument or vocal.

Gain-in controls how much of the input signal is delivered clean to the output, *delay* is the time difference in milliseconds between the original signal and its reflection, *decay* is the loudness of the reflected signal and *gain-out* is a final volume adjustment of the result.

There is no limit to the number of delay/decay pairs you can use and gains and decays can be negative or greater than 1 if you wish.

echo extends the length of the signal by the maximum delay time.

For example, this makes it sound as if there are twice as many instruments as are actually playing:

```
play_ng lead.aiff echo 0.8 0.88 60 0.4
```

If the delay is very short, it sound like a metallic robot: music:

```
play_ng lead.aiff echo 0.8 0.88 6 0.4
```

A longer delay sounds like an open air concert in the mountains:

```
play_ng lead.aiff echo 0.8 0.9 1000 0.3
```

One mountain more, and:

```
play_ng lead.aiff echo 0.8 0.9 1000 0.3 1800 0.25
```

For a flow diagram of how **echo** works, say **sox_ng −−help−effect echo**.

echos *gain-in gain-out* <*delay decay*>

Echos stands for 'Echo in Sequel' and adds a sequence of echoes to the audio. That is, the first echo takes the input, the second the input and the first echo, the third the input and the output of the second echo and so on. A single **echos** has the same effect as a single **echo**. Each *delay decay* pair gives the delay in milliseconds (with a minimum of one sample) and the decay of that echo. *Gain-out* is a final volume multiplier applied to the sum of the input × *gain-in* and the delays' outputs × their respective decays.

echos extends the length of the signal by the maximum delay time.

For example:

The sample is bounced twice in symmetric echos:

```
play_ng lead.aiff echos 0.8 0.7 700 0.25 700 0.3
```

The sample is bounced twice in asymmetric echos:

```
play_ng lead.aiff echos 0.8 0.7 700 0.25 900 0.3
```

The sample sounds as if it were played in a garage:

```
play_ng lead.aiff echos 0.8 0.7 40 0.25 63 0.3
```

For a flow diagram of how **echos** works, say **sox_ng −−help−effect echos**.

equalizer *frequency width*[**q**|**o**|**h**|**k**] *gain*

Apply a two-pole peaking equalization filter. With this filter, the signal level at and around a selected frequency can be increased or decreased while, unlike band-pass and band-reject filters, the level at all other frequencies is unchanged.

frequency gives the filter's central frequency in Hz, *width* gives its bandwidth and *gain* the required gain or attenuation in dB. Beware of **Clipping** when using a positive *gain*.

In order to produce complex equalization curves, this effect can be given several times, each with a different central frequency.

The filter is described in detail in [1].

This effect supports the **−−plot** global option.

fade [*type*] *fade-in-length* [*stop-position(=)* [*fade-out-length*]]

Apply a fade effect to the beginning, end, or both of the audio.

An optional *type* can be specified to select the shape of the fade curve: **q** for quarter of a sine wave, **h** for half a sine wave, **t** for linear (‘triangular’) slope, **l** for logarithmic, and **p** for inverted parabola. The default is logarithmic.

A fade-in starts from the first sample and ramps the signal level from 0 to full volume over the time given as *fade-in-length*. Specify 0 if no fade-in is wanted.

For a fade-out, the audio is truncated at *stop-position* and the signal level is ramped from full volume down to 0 over an interval of *fade-out-length* before the *stop-position*. If *fade-out-length* is not specified, it defaults to the same value as *fade-in-length*. No fade-out is performed if *stop-position* is not specified. If the audio length can be determined from the input file header and any previous effects, then ‘−0’ (or, for historical reasons, ‘0’) may be specified for *stop-position* to indicate the usual case of a fade out that ends at the end of the input audio stream.

See the **splice** effect.

fir [*coefs-file*|*coef* <*coef*>]

Use SoX’s FFT convolution engine with given Finite Impulse Response filter coefficients. If a single argument is given, it is the name of a file containing the filter coefficients (white space separated; may contain ‘#’ comments). If the filename is ‘−’ or if no argument is given, the coefficients are read from the ‘standard input’ (stdin); otherwise, coefficients may be given on the command line. Examples:

```
sox_ng in.au out.au fir .0195 −.082 .234 .891 −.145 .043

sox_ng in.au out.au fir coefs.txt
```

with `coefs.txt` containing

```
# HP filter: freq=10000
  1.2311233052619888e-01
 -4.4777096106211783e-01
  5.1031563346705155e-01
 -6.6502926320995331e-02
```

This effect supports the **−−plot** global option.

firfit [*knots-file*|<*freq gain*>]

Use SoX’s FFT convolution engine to make a filter whose frequency response approximates a spline passing through a series of frequency/gain pairs. If a single argument is given, it is the name of a file containing the knots (white space separated; may contain ‘#’ comments). If the given filename is ‘−’ or if no argument is given, the knots are read from the ‘standard input’ (stdin); otherwise, knots may be given on the command line.

Gains are in dB and the knot frequencies must be in increasing order.

Examples:

```
sox_ng in.au out.au firfit 20 0 10000 −3
```

gives a gentle low-pass filter and

```
sox_ng in.au out.au firfit knots.txt
```

with knots.txt containing

```
# Approximate telephone response
```

```
300  −100
400   −10
480     0
2800    0
3000  −10
3400 −100
```

approximates the response of a carbon microphone telephone.

This effect supports the **−−plot** global option.

flanger [-n|-l|-q] [-s|-t] [*delay*(0) [*depth*(2) [*regen*(0) [*width*(71) [*speed*(0·5) [*shape*(sine)] [*phase*(25) [*interp*(linear)]]]]]]]

Apply a flanging effect to the audio. See [3] for a detailed description of flanging.

The parameters give the base delay and the added swept delay in milliseconds, the percentage of regeneration (the delayed signal feedback), *width* the percentage of delayed signal that is mixed with the original, *speed* the number of sweeps per second, the shape of the swept wave (**sine** or **triangle**), the percentage of phase shift of the swept wave in multichannel flanges (0 = 100 = the same phase on each channel) and the type of digital delay line interpolation (**none**, **linear** or **quadratic**).

sine, **triangle**, **none**, **linear** and **quadratic** can be abbreviated.

The input and the delay's output are mixed and balanced so they don't clip, so a *width* of 100 gives 50:50 mixing; to obtain only the delayed output and none of the input, specify *width* as **inf**.

Despite containing a delay, **flanger** does not extend the length of the signal so, if you also want the last dregs of the delayed output and feedback, **pad** the signal beforehand.

sine, **triangle**, **none**, **linear** and **quadratic** can be abbreviated and **−s**, **−t**, **−n**, **−l** and **−q** are alternative ways to set the waveshape and the interpolation type without having to specify the rest of the parameters.

For a flow diagram of how **flanger** works, say **sox_ng −−help−effect flanger**.

gain [**−e**|**−B**|**−b**|**−r**] [**−n**] [**−l**|**−h**] [*gain-dB*(0)]

Apply amplification or attenuation to the audio signal or, in some cases, to some of its channels. Note that use of any of **−e**, **−B**, **−b**, **−r** and **−n** requires temporary file space to store the audio to be processed, so may be unsuitable for use with streamed audio.

Without other options, *gain-dB* adjusts the signal power level by the given number of dB: positive amplifies (beware of clipping), negative attenuates. With other options, the *gain-dB* amplification or attenuation is applied after the processing due to those options.

With the **−e** option, the levels of the audio channels of a multichannel file are equalized, i.e. gain is applied to all channels other than that with the highest peak level so that all channels attain the same peak level (but, without also giving **−n**, the audio is not normalized).

The **−B** (balance) option is similar to **−e**, but with **−B,** the RMS level is used instead of the peak level. **−B** might be used to correct stereo imbalance caused by an imperfect record turntable cartridge. Note that, unlike **−e**, **−B** might cause some clipping.

−b is similar to **−B** but has clipping protection, i.e. if necessary to prevent clipping whilst balancing, attenuation is applied to all channels. In conjunction with **−n**, **−B** and **−b** are synonymous.

The **−r** option is used in conjunction with a prior invocation of **gain** with the **−h** option—see below for details.

The **−n** option normalizes the audio to 0dB FSD. It is often used in conjunction with a negative *gain-dB* so that the audio is normalized to a given level below 0dB. For example,

```
sox_ng in.au out.au gain −n
```

normalizes to 0dB, and

```
sox_ng in.au out.au gain −n −3
```

normalizes to −3dB.

The **−l** option invokes a simple limiter. For example,

```
sox_ng in.au out.au gain −l 6
```

applies 6dB of gain but never clips. Note that limiting more than a few dBs more than occasionally in a piece of audio is not recommended as it can cause audible distortion. See the **compand** effect for a more capable limiter.

The **−h** option is used to apply gain to provide headroom for subsequent processing. For example, with

```
sox_ng in.au out.au gain −h bass +6
```

6dB of attenuation is applied prior to the bass boosting effect, ensuring that it does not clip. Of course, with **bass**, it is obvious how much headroom is needed but, with other effects (e.g. rate, dither), it is not always as clear. Another advantage of using **gain −h** rather than an explicit attenuation is that, if the headroom is not used by subsequent effects, it can be reclaimed with **gain −r**, for example:

```
sox_ng in.au out.au gain −h bass +6 rate 44100 gain −r
```

The above effects chain guarantees never to clip nor amplify; it attenuates if necessary to prevent clipping, but by only as much as is needed to do so.

Output formatting (dithering and bit-depth reduction) also requires headroom which cannot be reclaimed, e.g.

```
sox_ng in.au out.au gain −h bass +6 rate 44100 gain −rh dither
```

Here, the second **gain** invocation reclaims as much of the headroom as it can from the preceding effects but retains as much headroom as is needed for subsequent processing. The SoX global option **−G** can be given to automatically invoke **gain −h** and **gain −r**.

See the **norm** and **vol** effects.

highpass|**lowpass** [**−1**|**−2**] *frequency* [width[**q**|**o**|**h**|**k**]]

Apply a high-pass or low-pass filter with 3dB point *frequency*. The filter can be either single-pole (with **−1**), or double-pole (the default, or with **−2**). *width* applies only to double-pole filters; the default is Q = 0·707 and gives a Butterworth response. The filters roll off at 6dB per pole per octave (20dB per pole per decade). The double-pole filters are described in detail in [1].

These effects support the **−−plot** global option.

See **sinc** for filters with a steeper roll-off.

hilbert [**−n** *taps*]

Apply an odd-tap Hilbert transform filter, phase shifting the signal by 90 degrees.

This is used in many matrix coding schemes and for analytic signal generation. The process is often written as a multiplication by *i* (or *j*), the imaginary unit.

An odd-tap Hilbert transform filter has a band-pass characteristic, attenuating the lowest and highest frequencies. Its bandwidth can be controlled by the number of filter taps, which can be specified with **−n**. By default, the number of taps is chosen for a cutoff frequency of about 75 Hz.

This effect supports the **−−plot** global option.

ladspa [**−l**] [**−r**] *module* [*plugin*] {*argument*}

Apply a LADSPA [5] (Linux Audio Developer's Simple Plugin API) plugin. Despite the name, LADSPA is not Linux-specific and a wide range of effects is available as LADSPA plugins, such as CMT [6] (the Computer Music Toolkit) and Steve Harris's plugin collection [7]. The first argument is the plugin module, the second the name of the plugin (a module can contain more than one plugin) and any other arguments are for the control ports of the plugin. Missing arguments are supplied by default values if possible.

Normally, the number of input ports of the plugin must match the number of input channels and the number of output ports determines the output channel count. However, the **−r** (replicate) option allows cloning a mono plugin to handle multichannel input.

Some plugins introduce latency which SoX may optionally compensate for. The **−l** (latency compensation) option automatically compensates for latency as reported by the plugin via an output control port named "latency".

If it is set, the environment variable **LADSPA_PATH** is used as the search path for plugins. See **LADSPA_PATH** in the section **ENVIRONMENT**.

loudness [*gain* [*reference*]]

Loudness control is similar to the **gain** effect but provides equalization for the human auditory system. See http://en.wikipedia.org/wiki/Loudness for a detailed description of loudness. The gain is adjusted by the given *gain* parameter (usually negative) and the signal equalized according to ISO 226 w.r.t. a reference level of 65dB, though an alternative *reference* level may be given if the original audio has been equalized at some other level. A default gain of −10dB is used if a *gain* value is not given.

See the **gain** effect.

lowpass [**−1**|**−2**] *frequency* [width[**q**|**o**|**h**|**k**]]

Apply a low-pass filter. See the description of the **highpass** effect for details.

mcompand "*compand-args***"** {*frequency* **"***compand-args***"**}

The quoted *compand-args* are as for the **compand** effect:
*attack1***,***decay1*{**,***attack***,***decay*}
[*soft-knee-dB***:**]*in-dB1*[**,***out-dB1*]{**,***in-dB***,***out-dB*}
[*gain* [*initial-volume-dB* [*delay*]]]

The multi-band compander is similar to the single-band compander but the audio is first divided into bands using Linkwitz-Riley crossover filters and a separately specifiable compander is run on each band. See the **compand** effect for the definition of its parameters. Compand parameters are specified between double quotes and the crossover frequency for that band is given by *crossover-freq*; these can be repeated to create multiple bands.

The following examples approximate Dolby A compression and decompression, as used for tape noise reduction in professional recording studios:

```
# Dolby A compressor
sox_ng in.au dolbyA.au mcompand \
   ".1,.1 4:-56,-46,-36,-26,-26,-20,-17,-15,-9,-9" 80 \
   ".1,.1 4:-56,-46,-36,-26,-26,-20,-17,-15,-9,-9" 3k \
   ".1,.1 4:-56,-46,-36,-26,-26,-20,-17,-15,-9,-9" 9k \
   ".1,.1 4:-56,-42,-36,-23,-26,-18,-17,-14,-9,-9"

# Dolby A decompressor
sox_ng dolbyA.au out.au mcompand \
   ".1,.1 4:-46,-56,-26,-36,-20,-26,-15,-17,-9,-9" 80 \
   ".1,.1 4:-46,-56,-26,-36,-20,-26,-15,-17,-9,-9" 3k \
   ".1,.1 4:-46,-56,-26,-36,-20,-26,-15,-17,-9,-9" 9k \
   ".1,.1 4:-42,-56,-23,-36,-18,-26,-14,-17,-9,-9"
```

Real Dolby A probably compands each channel separately but that is left as an exercise to interested readers.

See **compand** for a single-band companding effect.

noiseprof [*profile-file*]
Calculate a profile of the audio for use in noise reduction. See the description of the **noisered** effect for details.

noisered [*profile-file* [*amount*]]
Reduce noise in the audio signal by profiling and filtering. This effect is moderately effective at removing consistent background noise such as hiss or hum. To use it, first run SoX with the **noiseprof** effect on a section of audio that ideally would contain silence but in fact contains noise—such sections are typically found at the beginning or the end of a recording. **noiseprof** writes a noise profile to *profile-file* or to stdout if no *profile-file* or if '−' is given. E.g.

```
sox_ng speech.wav −n trim 0 1.5 noiseprof speech.noise−profile
```

To actually remove the noise, run SoX again, this time with the **noisered** effect; **noisered** reduces noise according to a noise profile generated by **noiseprof**, from *profile-file* if it is given or from stdin if no *profile-file* or if '−' is given. E.g.

```
sox_ng speech.wav cleaned.wav noisered speech.noise−profile 0.3
```

How much noise should be removed is specified by *amount*—a number between 0 and 1 with a default of 0·5. Higher numbers remove more noise but present a greater likelihood of removing wanted components of the audio signal. Before replacing an original recording with a noise-reduced version, experiment with different *amount* values to find the optimal one for your audio; use headphones to check that you are happy with the results, paying particular attention to quieter sections of the audio.

On most systems, the two stages—profiling and reduction—can be combined using a pipe, e.g.

```
sox_ng noisy.wav −n trim 0 1 noiseprof | \
   play_ng noisy.wav noisered
```

norm [*dB-level(0)*]
Normalize the audio. **norm** is just an alias for **gain −n**; see the **gain** effect for details.

oops Out Of Phase Stereo effect. Mixes stereo to twin mono where each mono channel contains the difference between the left and right stereo channels. This is sometimes known as the 'karaoke' effect as it often has the effect of removing most or all of the vocals from a recording. It is equivalent to **remix 1,2i 1,2i**.

overdrive [*gain*(20) [*color*(20)]]
Non-linear distortion. The *color* parameter controls the amount of even harmonic content in the overdriven output. Both parameters range from 0 to 100.

pad { *[%]length*[*@position(=)*] }
Pad the audio with silence at the beginning, at the end or at any specified points throughout the audio. *length* is the amount of silence to insert and *position* the position in the input audio stream at which to insert it. Any number of lengths and positions may be specified, provided that a specified position is not less that the previous one. *Position* is optional for the first and last lengths specified and if omitted correspond to the beginning and the end of the audio respectively. For example, **pad 1.5 1.5** adds 1·5 seconds of silence at each end of the audio, whilst **pad 4000s@3:00** inserts 4000 samples of silence 3 minutes into the audio. If silence is wanted only at the end of the audio, either specify the end position or specify a zero-length pad at the start.

If a pad specification starts with with a **%** sign, the output is padded to a multiple of *length* at the specified position. For example, **pad 0 %10** adds silence at the end of the audio up to the next multiple of 10 seconds.

See **delay** for an effect that can add silence at the beginning of the audio on a channel-by-channel basis.

phaser [**−n**|**l**|**q**] [**−s**|**t**] [*gain-in*(.4) *gain-out*(.74) *delay*(3) *regen*(.4) *speed*(.5) [**−s**|**−t**(**−s**)]
Add a phasing effect to the audio. See [3] for a detailed description of phasing.

delay gives the maximum delay in milliseconds from 0 to 1000, *regen* the amount of feedback from the delay from −1 to +1 and *speed* the frequency of delay-time modulation wave in Hz.

The modulation is either sinusoidal (**−s**), which is preferable for multiple instruments, or triangular (**−t**) which gives single instruments a sharper phasing effect. *regen* can be from −1 to +1 but should usually be less than 0·5 to avoid clipping and *gain-out* is the final volume adjustment from −1 to +1.

The **−l** flag makes **phaser** do linear interpolation between samples when the offset into the delay line is not a whole number, which is about 15% slower but much less noisy and **−q** does quadratic interpolation, which is about 50% slower but even less noisy. **−n** explicitly asks for no interpolation, the default, fast and fuzzy.

In sox_ng, **−s** or **−t** can be given at the start or the end; to be compatible with earlier versions of SoX, supply all the parameters with one of these at the end, use *gain-in* and *gain-out* from 0 to 1, *delay* from 0 to 5, *speed* from 0·1 to 2 and don't use interpolation.

Technically, the SoX **phaser** is not a phaser; it is a flanger. A flanger does comb filtering with equidistant spacing (e.g. 100Hz, 200Hz, 300Hz, 400Hz, ...), while a real phaser does comb filtering with factored spacing (e.g. 100Hz, 200Hz, 400Hz, 800Hz, ...) that sounds more harmonic.

For example:

```
play_ng snare.flac phaser 0.8 0.74 3 0.4 0.5 -t
```

Gentler:

```
play_ng snare.flac phaser 0.9 0.85 4 0.23 1.3 -s
```

A popular sound:

```
play_ng snare.flac phaser 0.89 0.85 1 0.24 2 -t
```

More severe:

```
play_ng snare.flac phaser 0.6 0.66 3 0.6 2 -t
```

For a flow diagram of how **phaser** works, say **sox_ng −−help−effect phaser**.

pitch [**−q**] *shift* [*segment* [*search* [*overlap*]]]
Change the audio pitch but not the tempo.

shift gives the pitch shift as positive or negative 'cents' (i.e. 100ths of a semitone).

Pitch and **tempo** share the same fundamental algorithm; see the **tempo** effect for a description of the other parameters.

See the **bend**, **speed** and **tempo** effects.

rate [**−q**|**−l**|**−m**|**−g**|**−h**|**−e**|**−v**|**−u**] [override-options] [*frequency*]
Change the audio sampling rate (i.e. resample the audio) to any given *frequency* (even non-integer if this is supported by the output file format) using a quality level defined as follows:

	Quality	*B/W*	*Rej dB*	*Typical Use*
−q	quick	n/a	≈30 @ Fs/4	playback on ancient hardware
−l	low	80%	100	playback on old hardware
−m	medium	95%	100	audio playback
−g	generic	95%	100	16-bit
−h	high	95%	125	20-bit for 16-bit mastering
−e	extreme	95%	150	24-bit
−v	very high	95%	175	28-bit for 24-bit mastering
−u	ultra	95%	200	32-bit

These can also be selected with **−Q** *n* with *n* from 0 to 7.

B/W (bandwidth) is the percentage of the audio frequency band that is preserved and *Rej dB* is the level of noise rejection. Increasing levels of resampling quality come at the expense of increasing amounts of time to process the audio. If no quality option is given, the quality level used is 'high' when processing audio and 'low' when playing it. See **Playing & Recording Audio** above.

The 'quick' algorithm uses cubic interpolation; all others use band-limited interpolation. By default, all algorithms have a linear phase response; for 'medium' and above, the phase response is configurable (see below).

The **rate** effect is invoked automatically if SoX's **−r** option specifies a rate that is different to that of the input file(s). Alternatively, if this effect is given explicitly, then SoX's **−r** option need not be given. For example, the following two commands are equivalent:

```
sox_ng input.wav -r 48k output.wav bass -b 24
sox_ng input.wav        output.wav bass -b 24 rate 48k
```

though the second command is more flexible as it allows **rate** options to be given, and allows the effects to be ordered arbitrarily.

A user notes that resampling tracks and then concatenating them is more likely to create clicks at the joints than joining them first and resampling the result, due to edge effects.

Override Options

The simple quality selection described above provides settings that satisfy the needs of the vast majority of resampling tasks. Occasionally, however, it may be desirable to fine-tune the resampler's filter response; for qualities 'medium' and above, this can be achieved using the override options in the following table:

−M/−I/−L	Phase response=minimum/intermediate/linear
−s	Steep filter (bandwidth=99%)
−a	Allow aliasing/imaging above the pass band
−b *width*	Any bandwidth % (74−99·7 or 85-99·7 with 0
−p *phase*	Any phase response (0=minimum, 25=intermediate, 50=linear, 100=maximum)

All resamplers use filters that can sometimes create 'echo' (a.k.a. 'ringing') artefacts with transient signals such as those that occur with 'finger snaps' or other highly percussive sounds. Such artefacts are much more noticeable to the human ear if they occur before the transient ('pre-echo') than if they occur after it ('post-echo'). Note that the frequency of any such artefacts is related to the smaller of the original and new sampling rates but if this is at least 44·1kHz, the artefacts will lie outside the range of human hearing.

A phase response setting may be used to control the distribution of any transient echo between 'pre' and 'post': with minimum phase, there is no pre-echo but the longest post-echo; with linear phase, pre- and post-echo are in equal amounts (in signal terms, but not in audibility); the intermediate phase setting attempts to find the best compromise by selecting a small length (and level) of pre-echo and a medium-length of post-echo.

A minimum, intermediate or linear phase response is selected using the **−M**, **−I** and **−L** options; a custom phase response can be created with the **−p** option. Note that phase responses between ‘linear’ and ‘maximum’ (greater than 50) are rarely useful.

A resampler’s bandwidth setting determines how much of the frequency content of the original signal (w.r.t. the original sample rate when upsampling or the new sample rate when downsampling) is preserved during conversion. The term ‘pass band’ is used to refer to all frequencies up to the bandwidth point (e.g. for a 44·1kHz sampling rate and a resampling bandwidth of 95%, the pass band represents frequencies from 0Hz (DC) to circa 21kHz). Increasing the resampler’s bandwidth results in a slower conversion and can increase transient echo artefacts (and vice versa).

The **−s** ‘steep filter’ option changes the resampling bandwidth from the default of 95% (based on the 3dB point) to 99%. The **−b** option allows the bandwidth to be set to any value in the range 74−99·7% but bandwidth values greater than 99% are not recommended for normal use as they can cause excessive transient echo.

If the **−a** option is given, aliasing/imaging above the pass band is allowed. For example, with 44·1kHz sampling rate and a resampling bandwidth of 95%, this means that frequency content above 21kHz can be distorted. However, since this is above the pass band (i.e. above the highest frequency of interest/audibility), this may not be a problem. The benefits of allowing aliasing/imaging are reduced processing time and reduced (by almost half) transient echo artefacts.

The **−d** option sets the bit-accuracy in the range 15 to 33, or **−R** sets the bit-accuracy to obtain rejection of a specified number of dB.

Examples:

```
sox_ng input.wav −b 16 output.wav rate −s −a 44100 dither −s
```

is default (high) quality resampling with overrides for a steep filter, to allow aliasing, at a 44·1kHz sample rate and noise-shaped dithering to a 16-bit WAV file.

```
sox_ng input.wav −b 24 output.aiff rate −v −I −b 90 48k
```

is very high quality resampling with overrides for an intermediate phase, a bandwidth of 90%, at a 48k sampling rate and storing the output to a 24-bit AIFF file.

Advanced Options

The **−i** option forces the use of a particular interpolator coefficient from -1 to 2.

The **−c** option tries to limit the number of coefficients to a number of kilobytes; its argument can be from 100 up.

The **−B** option sets the percentage of the pass-band to preserve, from 53 to 95.

The **−A** option sets the percentage of the bandwith without aliasing, from 85 to 100.

−f sets zero pass-band roll-off instead of 0.01dB for -Q 0-2.

−n disables internal small-integer optimizations and

−t increases the irrational ratio accuracy.

remix [**−a**|**−m**] [**−p**] <*out-spec*>
out-spec = **0** | *in-spec*{**,***in-spec*}
in-spec = [*in-chan*][**−**[*in-chan2*]][*vol-spec*]
vol-spec = **p**|**i**|**v**[*volume*]

Select and mix input audio channels into output audio channels. Each output channel is specified in turn by a given *out-spec* which is a list of contributing input channels and volume specifications.

Note that this effect operates on the audio channels within the SoX effects processing chain; it should not be confused with the **−m** global option, where multiple files are mix-combined before entering the effects chain.

An *out-spec* contains comma-separated input channel numbers and hyphen-delimited channel number ranges; alternatively, **0** may be given to create a silent output channel. For example,

```
sox_ng input.wav output.wav remix 6 7 8 0
```

creates an output file with four channels, where channels 1, 2, and 3 are copies of channels 6, 7, and 8 in the input file, and channel 4 is silent. Whereas

```
sox_ng input.wav output.wav remix 1−3,7 3
```

creates a (somewhat bizarre) stereo output file where the left channel is a mix-down of input channels 1, 2, 3 and 7 and the right channel is a copy of input channel 3.

Where a range of channels is specified, the channel numbers to the left and right of the hyphen are optional and default to 1 and to the number of input channels respectively. Thus

```
sox_ng input.wav output.wav remix −
```

performs a mix-down of all input channels to mono.

By default, where an output channel is mixed from multiple input channels, each input channel is scaled by a factor of $^1/_n$. Custom mixing volumes can be set by following a given input channel or range of input channels with a *vol-spec* (volume specification) which is one of the letters **p**, **i**, or **v**, followed by a volume number, the meaning of which depends on the given letter:

Letter	*Volume number*	*Notes*
p	power adjust in dB	0 = no change
i	power adjust in dB	As for **p** but invert the audio
v	voltage multiplier	1 = no change; 0·5 ≈ 6dB attenuation; 2 ≈ 6dB gain; −1 = invert

If an *out-spec* includes at least one *vol-spec* then, by default, $^1/_n$ scaling is not applied to any other channels in the same *out-spec* (though maybe in other *out-spec*s) though the **−a** (automatic) option can be given to retain the automatic scaling in this case. For example,

```
sox_ng input.wav output.wav remix 1,2 3,4v0.8
```

results in channel level multipliers of 0·5,0·5 and 1,0·8, whereas

```
sox_ng input.wav output.wav remix −a 1,2 3,4v0.8
```

results in channel level multipliers of 0·5,0·5 and 0·5,0·8.

The **−m** (manual) option disables all automatic volume adjustments, so

```
sox_ng input.wav output.wav remix −m 1,2 3,4v0.8
```

results in channel level multipliers of 1,1 and 1,0·8.

The volume number is optional and omitting it corresponds to no volume change; however, the only case in which this is useful is in conjunction with **i**. For example, if *input.wav* is stereo, then

```
sox_ng input.wav output.wav remix 1,2i
```

is a mono equivalent of the **oops** effect.

If the **−p** option is given, any automatic $^1/_n$ scaling is replaced by $^1/\sqrt{n}$ (‘power’) scaling; this gives a louder mix but one that may occasionally clip.

One use of the **remix** effect is to split an audio file into a set of files, each containing one of the constituent channels in order to perform subsequent processing on individual audio channels.

When more than a few channels are involved, a script such as the following is useful:

```
#! /bin/sh
chans=`soxi_ng -c "$1"`
while [ $chans -ge 1 ]; do
   chans0=`printf %02i $chans`   # 2 digits hence up to 99 chans
   out=`echo "$1"|sed "s/\(.*\)\.\(.*\)/\1-$chans0.\2/"`
   sox_ng "$1" "$out" remix $chans
   chans=`expr $chans - 1`
done
```

If a file *input.wav* containing six audio channels were given, the script would produce six output files: *input-01.wav*, *input-02.wav*, ..., *input-06.wav*.

See the **swap** effect.

repeat [*count*(1)|**−**]
Repeat the entire audio *count* times, or once if *count* is not given. The special value **−** requests infinite repetition. It requires temporary file space to store the audio to be repeated. Note that repeating once yields two copies: the original audio and the repeated audio.

reverb [**−w**] [*reverberance*(50%) [*HF-damping*(50%) [*room-scale*(100%) [*stereo-depth*(100%) [*pre-delay*(0ms) [*wet-gain*(0dB)]]]]]]

Add reverberation to the audio using the 'freeverb' algorithm. A reverberation effect is sometimes desirable for concert halls that are too small or contain so many people that the hall's natural reverberance is diminished. Applying a small amount of stereo reverb to a dry mono signal usually makes it sound more natural. See [3] for a detailed description of reverberation.

This effect increases the volume of the audio and continues to reverberate after the input finishes so, to prevent clipping and keep the audible part of the final reverberation, a typical invocation might be:

```
play_ng dry.au gain -3 pad 0 1 reverb
```

The **−w** option can be given to select only the 'wet' signal, thus allowing it to be processed further, independently of the 'dry' signal. E.g.

```
play_ng -m in.au "|sox_ng in.au -p reverse reverb -w reverse"
```

for a reverse reverb effect.

reverse Reverse the audio completely. Requires temporary file space to store the audio to be reversed.

riaa Apply RIAA vinyl playback equalization. The sampling rate must be 44·1, 48, 88·2, 96 or 192kHz.

This effect supports the **−−plot** global option.

saturation [*type* [*blend* [*offset* [*drive*|*color*|*threshold*]]]]
Add saturation, which can produce effects ranging from subtle warmth to crunchy fuzz. The *type* parameter selects the saturation type: **tanh** (the default), **sqrt** or **diode**.

For all types, the *blend* parameter (default 1) controls the mixture of wet and dry signals in the output, with 1 being fully wet. The *offset* parameter (default 0) adds a DC offset to the input to produce asymmetric distortion. The offset is removed from the output, so that a zero input level produces a zero output level, but when the input is non-zero the output waveform is likely to be asymmetric.

The **tanh** saturation type uses the hyperbolic tangent function to apply soft clipping. The *drive* parameter (default 1) controls the input gain and thus the amount of distortion.

The **sqrt** saturation type uses a mixture of two functions: *x*sqrt(|x|)* and *sgn(x)*sqrt(|x|)*, which give different tonal qualities to the output. The *color* parameter (default 0.5) controls the mixture of these functions, with 0 being purely *x*sqrt(|x|)* and 1 being purely *sgn(x)*sqrt(|x|)*.

The **diode** saturation type models the effect of using a pair of diodes to clip the signal when it exceeds a *threshold* (default 0.5). The *blend* parameter, by mixing the wet and dry signals, effectively controls the amount of attenuation that occurs above the threshold, from no attenuation when *blend* is 0 to complete attenuation (hard clipping) when *blend* is 1.

See the **overdrive** effect for another kind of non-linear distortion. When the *offset* parameter is used to produce asymmetric distortion, the **highpass** effect can be used to rebalance the waveform's positive and negative amplitude.

sdm [**−f** *filter*] [**−t** *order*] [**−n** *num*] [**-l** *latency*]

Apply a 1-bit sigma-delta modulator producing DSD output. The input should be previously upsampled, e.g. with the **rate** effect, to a high rate, 2·8224MHz for DSD64. The **−f** option selects the noise-shaping filter from the following list where the number indicates the order of the filter:

clans-4	sdm-4
clans-5	sdm-5
clans-6	sdm-6
clans-7	sdm-7
clans-8	sdm-8

The noise filter may be combined with a partial trellis/viterbi search by supplying the following options:

−t *order*
Trellis order, max 32.

−n *num* Number of paths to consider, max 32.

−l *latency*
Output latency, max 2048.

The result of using these parameters is hard to predict and can include high noise levels or instability. Caution is advised.

silence [**−l**] *above-periods* [*duration threshold*[**d**|**%**]]
[*below-periods duration threshold*[**d**|**%**]]

Removes silence from the beginning, middle or end of the audio, where 'silence' is determined by a specified threshold.

The *above-periods* value is used to indicate whether audio should be trimmed at the beginning of the audio. A value of zero indicates that no silence should be trimmed from the beginning in which case *duration* and *threshold* are omitted. When a non-zero *above-periods* is specified, you must also specify a *duration* and *threshold* and it trims audio until it finds non-silence. It will normally be 1 when trimming silence from the beginning of the audio, but it can be increased to higher values to trim all audio up to the Nth non-silence period. For example, if you have an audio file with two songs that each contains 2 seconds of silence before the song, you could specify an *above-period* of 2 to strip out both silences and the first song.

duration indicates the amount of time for which non-silence must be detected before it stops trimming the silence before it. By increasing *duration*, short bursts of quiet noise can be treated as silence and trimmed off. *duration* has the peculiarity that a bare number is interpreted as a sample count, not as a number of seconds. To specify seconds, either use the **t** suffix (as in **2t**), a decimal point (as in **2.0**) or specify minutes too (as in **0:02**).

threshold indicates the maximum sample value in any channel is considered silence. For digital audio, a value of 0 may be fine but for audio recorded from analog you may wish to increase the value to include background noise. *threshold* numbers may be suffixed with **d** to indicate that the value is in decibels or **%** to indicate a percentage of the maximum possible sample value. By default, it is in percent.

To trim silence from the end of the audio, specify a *below-periods* count, which means to remove all audio after the last onset of silence is detected. Normally, this will be 1 but it can be increased to leave shorter periods of silence and the audio that follows them intact. For example, if you have a track with 1 second of silence in the middle and 1 second at the end, you could set *below-period* to 2 to leave the middle silence and what follows it and remove from the final silence on.

When *below-periods* is given, its *duration* specifies the length of silence that must exist before audio is not copied any more. By specifying a higher *duration*, shorter silences that are wanted can be left in the audio. For example, if you have a song with 1 second of silence in the middle and 2 seconds of silence at the end, a *duration* of 2 could be used to skip over the middle silence and trim the end instead of starting trimming from half way through.

Unfortunately, the length of the silence at the end has to be longer than any preceding silence for this to work so you must know the length of the silence at the end.

A more reliable way to trim silence from the end is to use the **silence** effect in combination with the **reverse** effect. By first reversing the audio, you can use the *above-periods* to trim from what looks like the front of the file, then reverse it again to get back to normal.

To remove silence from the middle of a file, give a negative *below-periods*. This value is then treated as positive value and is also used to indicate that the effect should restart processing as specified by the *above-periods*, making it suitable for removing periods of silence in the middle of the audio.

The **−l** option indicates that *below-periods*' *duration* of 'silent' audio should be left intact at the beginning of each period of silence, for example, if you want to remove long pauses between words but do not want to remove the pauses completely.

The following example shows how this effect can be used to make a recording that does not contain the silence that usually occurs between pressing the record button and the start of the performance:

`rec_ng` *parameters filename other-effects* silence 1 5 2%

This example should remove the start of the recording until there's a period of non-silence longer than 0.2s and louder than 0.1%, then start searching for a silence that's longer than 1s and quieter than 3% and remove it if found, leaving the first 1s of it in place, then start copying again until a silence is found that's longer than 1s and quieter than 3%, trim that to 1s and so on.

```
sox_ng in.au out.au silence −l 1 0.2 0.1% −1 1.0 3%
```

sinc [**−a** *att*|**−b** *beta*] [**−p** *phase*|**−M**|**−I**|**−L**] [**−t** *tbw*|**−n** *taps*]
[*freqHP*][*−freqLP* [**−t** tbw|**−n** taps]] [**−r**]] [**−d**]]

Apply a kaiser-windowed low-pass, high-pass, band-pass or band-reject filter to the signal. The *freqHP* and *freqLP* parameters give the frequencies of the 6dB points of a high-pass and low-pass filter that may be invoked individually or together. If both are given, *freqHP* less than *freqLP* creates a band-pass filter and *freqHP* greater than *freqLP* creates a band-reject filter. For example, the invocations

```
sinc 3k
sinc −4k
sinc 3k−4k
sinc 4k−3k
```

create a high-pass, low-pass, band-pass and band-reject filter respectively.

The default stop band attenuation of 120dB can be overridden with **−a**; alternatively, the kaiser window's 'beta' parameter can be given directly with **−b**.

The default transition bandwidth of 5% of the total band can be overridden with **−t** (and *tbw* in Hertz); alternatively, the number of filter taps can be given directly with **−n** and is limited to the range of 11−32767.

If both *freqHP* and *freqLP* are given, a **−t** or **−n** option given to the left of the frequencies applies to both frequencies; one of these options given to the right of the frequencies applies only to *freqLP*.

The **−p**, **−M**, **−I** and **−L** options control the filter's phase response; see the **rate** effect for details.

The **−r** option controls whether the filter should round the number of taps to the closest integer instead of truncating it.

The **−d** option specifies that, if a low-pass filter is being created and the cutoff frequency is at or above the Nyquist frequency, the **sinc** effect should be deleted from the effects chain instead of failing.

This effect supports the **−−plot** global option.

softvol [*volume*(1.0) [*double-time*(0) [*headroom*(0)]]]

The soft volume effect applies a simple multiplier to the audio ensuring that it does not clip. When a sample would have clipped the volume multiplier is automatically reduced to compensate.

It is a simple compander with the advantages of running fast, having no pre- or post-echo and reacting on the crests of the wave, so its volume-reduction glitches don't add audible noise.

volume sets the initial volume multiplier; the default of 1.0 means no change.

double-time says that the volume should slowly increase at a rate that makes it double every *double-time* seconds. A good value for usual music is 10 and the default value of 0 says that the volume should not increase automatically.

headroom is in dB and limits the loudest amplitude to less than the 32-bit maximum. This may be necessary when the final bit-depth reduction and/or dithering make it clip. A value of 0.1 is sufficient to protect down to a bit-depth of 8 with dithering.

When playing sound in interactive mode, the '**v**' and '**V**' keys reduce and increase the volume if there is a **softvol** in the effects chain. If there are more than one, which one it adjusts is probably random.

spectrogram [*options*]

Create a spectrogram of the audio. The audio is passed unmodified through the SoX processing chain. This effect is optional—type **sox_ng −−help** and check the list of supported effects to see if it has been included.

The spectrogram is rendered in a Portable Network Graphic (PNG) file and shows time in the X axis, frequency in the Y axis and audio signal magnitude in the Z axis, represented by the color (or optionally the intensity) of the pixels in the X-Y plane. If the audio signal contains multiple channels, these are shown from top to bottom starting from channel 1, which is the left channel for stereo audio.

For example, if 'my.wav' is a stereo file, then

```
sox_ng my.wav −n spectrogram
```

creates a spectrogram of the entire file in the file 'spectrogram.png'. More often though, analysis of a smaller portion of the audio is required; e.g. with

```
sox_ng my.wav −n remix 2 trim 20 30 spectrogram
```

the spectrogram shows information only from the second (right) channel of thirty seconds of audio starting from twenty seconds in. To analyze a small portion of the frequency domain, the **rate** effect may be used, e.g.

```
sox_ng my.wav −n rate 6k spectrogram
```

allows detailed analysis of frequencies up to 3kHz (half the sampling rate) i.e. where the human auditory system is most sensitive. See also the −R option below. With

```
sox_ng my.wav −n trim 0 10 spectrogram −x 600 −y 200 −z 100
```

the given options control the size of the spectrogram's X, Y & Z axes (in this case, the spectrogram area of the produced image will be 600 by 200 pixels in size and the Z axis range will be 100 dB). Note that the produced image includes axes, legends etc. and will be larger than the specified spectrogram size unless the **−r** option is given: if each spectrogram is $x \times y$ and there are c channels, the image will be $x + 144$ by $(y \times c) + 78$, plus $c - 1$ if **−a** was not given, and 20 pixels higher than this if you gave **−t** *Title*. A raw spectrogram will be x by $y \times c$.

In this example

```
sox_ng −n −n synth 6 tri 10k:14k spectrogram −z 100 −w kaiser
```

an analysis window with high dynamic range is selected to best display the spectrogram of a swept triangular wave. For a similar example, append the following to the 'chime' command in the description of the **delay** effect (above):

```
rate 2k spectrogram −X 200 −Z −10 −w kaiser
```

Options are also available to control the appearance (color set, brightness, contrast etc.) and filename of the spectrogram; e.g. with

```
sox_ng my.wav −n spectrogram −m −l −o print.png
```

a spectrogram is created suitable for printing on a black and white printer.

Options

−x *num* Change the (maximum) width (X axis) of the spectrogram from its default value of 800 pixels to a given number between 100 and a million. See **−X** and **−d**.

−X *num*
X axis pixels per second; the default is auto-calculated to fit the audio to the X axis size if its duration is known or given with −d, or 100 otherwise. If given without a **−x** option when the length of the audio is known, this option determines the width of the spectrogram; otherwise, it affects the duration of the spectrogram. *num* can be from 1 (low time resolution) to 5000 (high time resolution) and need not be an integer. SoX may make a slight adjustment to the given number for processing quantization reasons; if so, SoX reports the actual number used (viewable when the SoX global option **−V** is in effect).

−y *num* Sets the size of the Y axis per channel in pixels; this is the number of frequency 'bins' used in the Fourier analysis that produces the spectrogram. By default the Y axis size is chosen automatically, depending on the **-Y** height and the number of channels, with a minimum of 64.

−Y *num*
Sets the total height of the spectrogram(s). The default value is 550 pixels and the maximum is a million. If *num* is not an exact multiple of the number of channels, the actual total height will be a few pixel rows less.

−z *num* Z axis (color) range in dB, default 120. This sets the dynamic range of the spectrogram to be −*num* dBFS to 0 dBFS. *Num* may range from 20 to 180. Decreasing dynamic range effectively increases the contrast of the spectrogram display and vice versa.

−Z *num*
Sets the upper limit of the Z axis in dBFS. A negative *num* effectively increases the brightness of the spectrogram display and vice versa.

−n Normalizes the upper limit of the Z axis so that the loudest pixels are shown using the brightest color in the palette—a kind of automatic **−Z** flag.

−q *num* Sets the Z axis quantization, i.e. the number of different colors (or intensities) in which to render Z axis values. A small number (e.g. 4) gives a poster-like effect making it easier to discern magnitude bands of similar level and results in a smaller PNG file. The number given specifies the number of colors to use in the Z axis range; two colors are reserved to represent out-of-range values.

−w *name*
Select a window function: **Hann** (the default), **Hamming**, **Bartlett**, **Rectangular**, **Kaiser** or **Dolph**. The spectrogram is produced using the Discrete Fourier Transform (DFT) algorithm and a significant parameter of this algorithm is the choice of window function. By default, SoX uses the Hann window, which has good all-round properties for frequency resolution and dynamic range. For better frequency resolution but lower dynamic range, select a Hamming window; for higher dynamic range but poorer frequency resolution, select a Dolph window.

−W *num*
Window adjustment parameter. This can be used to make small adjustments to the Kaiser and Dolph windows. A positive number (up to ten) increases its dynamic range, a negative number decreases it.

−s Allow slack overlapping of DFT windows. This can, in some cases, increase image sharpness and give greater adherence to the **−x** value but at the expense of a little spectral loss.

−m Creates a monochrome spectrogram (the default is color).

−h Selects a high-color palette which is less visually pleasing than the default color palette but it may make it easier to differentiate different levels. If this option is used in conjunction with **−m**, the result is hybrid monochrome/color palette.

−p *num* Permute the colors in a color or hybrid palette. The *num* parameter, from 1 (the default) to 6, selects the permutation.

−l Creates a ‘printer-friendly’ spectrogram with a light background (the default has a dark background).

−a Suppress the display of the axis lines. This is sometimes useful in helping to discern artefacts at the spectrogram edges.

−r Raw spectrogram: suppress the display of axes and legends.

−A Selects an alternative, fixed color set. This is provided only for compatibility with spectrograms produced by another package. It should not normally be used as it has some problems, not least, a lack of differentiation at the bottom end which results in masking of low-level artefacts.

−t *text* Set the image title, the text to display above the spectrogram. If you need it to be ‘chorus’ or some other effect’s name, surround it by spaces inside double quotes.

−c *text* Set (or clear) the image comment, the text to display below and to the left of the spectrogram.

−o *file* The name of the spectrogram output PNG file, default ‘spectrogram.png’. If ‘-’ is given, the spectrogram is sent to the ‘standard output’ (stdout).

−L Plot the frequency on a logarithmic axis.

−R *L*:*H* Specify the frequency range (from *L* to *H*). The frequencies can have an optional suffix

```
sox_ng mymusic.mp3 −n spectrogram −L −R 100:8k
```

By default, the lowest frequency is 0Hz for a linear graph or 1Hz for a logarithmic graph and the highest is the Nyquist frequency.

Advanced Options

In order to process a smaller section of audio without affecting other effects or the output signal (unlike when the **trim** effect is used), the following options may be used:

−d *duration*

This option sets the X axis resolution such that audio with the given *duration* (a time specification) fits the selected (or default) X axis width. It defaults, if the audio length is known, to the audio length minus the start time. For example,

```
sox_ng input.mp3 output.wav −n spectrogram −d 1:00 stats
```

creates a spectrogram showing the first minute of the audio, while the **stats** effect is applied to the entire audio signal.

See **−X** for an alternative way of setting the X axis resolution.

−S *position(=)*

Start the spectrogram at the given point in the audio stream. For example

```
sox_ng input.aiff output.wav spectrogram −S 1:00
```

creates a spectrogram showing all but the first minute of the audio (the output file, however, receives the entire audio stream).

For the ability to perform off-line processing of spectral data, see **stat −freq**.

speed *factor*[**c**]

Adjust the audio speed (pitch and tempo together). *factor* is either the ratio of the new speed to the old speed (greater than 1 speeds it up, less than 1 slows it down) or, if the letter **c** is appended, it's the number of cents (100ths of a semitone) by which the pitch (and tempo) should be adjusted: greater than 0 increases, less than 0 decreases.

Technically, the speed effect only changes the sample rate information, leaving the samples themselves untouched. The **rate** effect is invoked automatically to resample to the output sample rate, using its default quality/speed. For higher quality or higher speed resampling, in addition to the **speed** effect, specify the **rate** effect with the desired quality option.

See the **bend**, **pitch** and **tempo** effects.

speexdsp [**−agc** [*target_level*(100)]] [**−denoise** [*max_db*(15)]] [**−dereverb**] [**−fps** *frames_per_second*(20)] [**−spf** *samples_per_frame*]

Use the Speex DSP library to improve perceived sound quality.

If no options are specified, the **−agc** and **−denoise** features are enabled.

−agc [*target_level*]

Enable automatic gain control and optionally specify a target volume level from 1 to 100.

−denoise [*max_db*]

Enable noise reduction and optionally specify the maximum attenuation from 1 to 100.

−dereverb

Enable reverb reduction.

−fps *frames_per_second*

Specify the number of frames per second from 1-100.

−spf *samples_per_frame*

Specify the number of samples per frame. The default is derived from the **−fps** setting so that frames abut but do not overlap.

splice [**−h**|**−t**|**−q**] {*position(=)*[**,***excess*[**,***leeway*]]}

Splice audio sections together. This effect provides two things over simple audio concatenation: a (usually short) cross-fade is applied at the join and a wave similarity comparison is made to help determine the best place at which to make the join.

One of the options **−h**, **−t**, or **−q** may be given to select the fade envelope as half cosine wave (the default), triangular (a.k.a. linear), or quarter cosine wave (e.g. for a cross-fade of correlated audio).

	Audio	*Fade level*	*Transitions*
−h	correlated	constant gain	smooth
−t	correlated	constant gain	abrupt
−q	uncorrelated	constant power	smooth

To perform a splice, first use the **trim** effect to select the audio sections to be joined together. As when performing a tape splice, the end of the section to be spliced onto should be trimmed with a small *excess* (default 0·005 seconds) after the ideal joining point. The beginning of the audio section to splice on should be trimmed with the same *excess* before the ideal joining point plus an additional *leeway* (default 0·005 seconds). SoX should then be invoked with the two audio sections as input files and the **splice** effect given with the position at which to perform the splice—this is length of the first audio section (including the excess).

The following diagram uses the tape analogy to illustrate the splice operation. The effect simulates the diagonal cuts and joins the two pieces:

```
  length1   excess
-----------><--->
_________   :   :   _________________
         \  :   :  :\     `
          \ :   :  : \     `
           \:   :  :  \     `
            *   :  :   * - - *
             \  :  :   :\     `
              \ :  :   : \     `
_______________\:  :   :  \_____`____
                   :   :   :     :
                   <--->   <----->
                   excess  leeway
```

where * indicates the joining points.

For example, a long song begins with two verses which start (as determined e.g. by using the **play_ng** command with the **trim** (*start*) effect) at times 0:30·125 and 1:03·432. The following commands cut out the first verse:

```
sox_ng too-long.wav part1.wav trim 0 30.130
```

(5 ms excess, after the first verse starts)

```
sox_ng too-long.wav part2.wav trim 1:03.422
```

(5 ms excess plus 5 ms leeway, before the second verse starts)

```
sox_ng part1.wav part2.wav just-right.wav splice 30.130
```

For another example, the SoX command

```
play_ng "|sox_ng -n -p synth 1 sin %1" "|sox_ng -n -p synth 1 sin %3"
```

generates and plays two notes, but there is a nasty click at the transition; the click can be removed by splicing instead of concatenating the audio, i.e. by appending **splice 1** to the command. Clicks at the beginning and end of the audio can be removed by *preceding* the splice effect with **fade q .01 2 .01**.

Provided your arithmetic is good enough, multiple splices can be performed with a single **splice** invocation. For example, with a Bourne shell script ‘acpo’:

```
#! /bin/sh
```

```
# Audio Copy and Paste Over
# acpo infile copy-start copy-stop paste-over-start outfile
# No chained time specifications allowed for the parameters
# (i.e. such that contain +/-).
e=0.005                        # Using default excess
l=$e                           # and leeway.
sox_ng "$1" piece.wav trim $2-$e-$l =$3+$e
sox_ng "$1" part1.wav trim 0 $4+$e
sox_ng "$1" part2.wav trim $4+$3-$2-$e-$l
sox_ng part1.wav piece.wav part2.wav "$5" \
   splice $4+$e +$3-$2+$e+$l+$e
```

two splices are used to 'copy and paste' audio.

It is also possible to use this effect to perform general cross-fades, e.g. to join two songs. In this case, *excess* would typically be a number of seconds, the **−q** option would typically be given to select an 'equal power' cross-fade and *leeway* should be zero (which is the default if **−q** is given). For example, if f1.wav and f2.wav are audio files to be cross-faded, then

```
sox_ng f1.wav f2.wav out.wav splice -q $(soxi_ng -D f1.wav),3
```

cross-fades the files where the point of equal loudness is 3 seconds before the end of f1.wav, i.e. the total length of the cross-fade is $2 \times 3 = 6$ seconds (`$(...)` is POSIX shell notation that is replaced by the output of the enclosed command).

stat [**−s** *scale*] [**−rms**] [**−freq**] [**−v**] [**−d**] [**−a**] [**−h**]

Display time and frequency domain statistical information about the audio. Audio is passed unmodified through the SoX processing chain.

The information is output to the 'standard error' (stderr) stream and is calculated (where n is the duration of the audio in samples, c is the number of audio channels, r is the audio sample rate and x_k represents the value (in the range −1 to +1) of each successive sample in the audio), as follows:

Samples read	$n \times c$
Length (seconds)	$n \div r$
Scaled by	See **−s** below.
Maximum amplitude	$\max(x_k)$ The maximum sample value in the audio; usually this will be a positive number.
Minimum amplitude	$\min(x_k)$ The minimum sample value in the audio; usually this will be a negative number.
Midline amplitude	$\frac{1}{2}\min(x_k)+\frac{1}{2}\max(x_k)$
Mean norm	$\frac{1}{n}\Sigma\lvert x_k\rvert$ The average of the absolute value of each sample in the audio.
Mean amplitude	$\frac{1}{n}\Sigma x_k$ The average of each sample in the audio. If this figure is non-zero, then it indicates the presence of a DC offset which could be removed using the **dcshift** effect.
RMS amplitude	$\sqrt{(\frac{1}{n}\Sigma x_k^2)}$ The level of a DC signal that would have the same power as the audio's average power.
Maximum delta	$\max(\lvert x_k - x_{k-1}\rvert)$
Minimum delta	$\min(\lvert x_k - x_{k-1}\rvert)$
Mean delta	$\frac{1}{n-1}\Sigma\lvert x_k - x_{k-1}\rvert$
RMS delta	$\sqrt{(\frac{1}{n-1}\Sigma(x_k - x_{k-1})^2)}$
EBUR128 Momentary	The maximum momentary loudness over 400ms
EBUR128 Short Term	The maximum short term loudness over 3 seconds
EBUR128 Integrated	The integrated loudness over the whole file
EBUR128 True Peak	The maximum of the True Peak of each channel
Rough frequency	In Hz.

Volume Adjustment The parameter to the **vol** effect which would make the audio as loud as possible without clipping. See the discussion on **Clipping** above for reasons why it is rarely a good idea actually to do this.

Note that the delta measurements are not applicable to multichannel audio and EBU R 128 (=ITU-R BS.1770) measurements are in Loudness Units referenced to Full Scale (LUFS),

The **−s** option can be used to scale the input data by a given factor. The default value of *scale* is 2147483647 (the maximum value of a 32-bit signed integer) as internal effects always work with those. A lower value means that a different sample value should be considered as the full-scale amplitude.

The **−rms** option converts all average values to ‘root mean square’ format.

The **−freq** option outputs the input’s power spectrum (a 4096-point DFT) instead of the statistics listed above. This should only be used with a single-channel audio file.

The **−v** option displays only the ‘Volume Adjustment’ value.

The **−d** option displays a hex dump of the 32-bit signed PCM data audio in SoX’s internal buffer. This is mainly used to help track down endian problems that sometimes occur in cross-platform versions of SoX.

The **−a** option outputs the average power spectrum instead of the power spectrum for each 4096-point DFT.

The **−h** option uses the "histogram algorithm" to calculate the integrated EBU R-128 loudness, which requires less memory but is less accurate.

The **−j** option outputs the statistics in JSON format, e.g.:

```
{
  "samples_read": 22699008,
  "length": 236.448,
  "scaled_by": 2.14748e+09,
  "maximum_amplitude": 0.818604,
  "minimum_amplitude": −0.532471,
  "midline_amplitude": 0.143066,
  "mean_norm": 0.0352694,
  "mean_amplitude": 0.00180676,
  "rms_amplitude": 0.056726,
  "maximum_delta": 0.367126,
  "minimum_delta": 0,
  "mean_delta": 0.0177341,
  "rms_delta": 0.0268538,
  "rough_frequency": 3616,
  "volume_adjustment": 1.22159
}
```

If **−rms** was given, `"scaled_by"` will be `"scaled_by_rms"` and if **−e** was given, you also get

```
  "ebur128_momentary": −30.3408,
  "ebur128_short_term": −35.4501,
  "ebur128_integrated": −21.3583,
```

Some fields may be absent if their values are incalculable (EBUR128 figures) or would be infinite (like the RMS of silence).

As JSON uses scientific notation, it can shows the values of very small numbers that the usual output shows as zero.

The most common use of **stat** is to measure the characteristics of a single audio file, for which the syntax is:

```
sox_ng file.wav −n stat
```

where **−n** means "No audio output is required."

stats [**−b** *bits*|**−x** *bits*|**−s** *scale*] [**−w** *time*] [**−j**]

Display time domain statistical information about the audio channels; audio is passed unmodified through the SoX processing chain. Statistics are calculated and displayed for each audio channel and, where applicable, an overall figure is also given.

For example, for a typical well-mastered stereo music file:

```
              Overall     Left      Right
DC offset    0.000803 -0.000391  0.000803
Min level   -0.750977 -0.750977 -0.653412
Max level    0.708801  0.708801  0.653534
Pk lev dB       -2.49     -2.49     -3.69
RMS lev dB     -19.41    -19.13    -19.71
RMS Pk dB      -13.82    -13.82    -14.38
RMS Tr dB      -85.25    -85.25    -82.66
Crest factor        -      6.79      6.32
Flat factor      0.00      0.00      0.00
Pk count            2         2         2
Bit-depth       16/16     16/16     16/16
Num samples     7.72M
Length s      174.973
Scale max    1.000000
Window s        0.050
```

DC offset, *Min level*, and *Max level* are shown, by default, in the range ±1. If the **−b** (bits) options is given, these three measurements are scaled to a signed integer with the given number of bits from 2 to 32. For example, for 16 bits, the scale would be −32768 to +32767. The **−x** option behaves the same way as **−b** except that the signed integer values are displayed in hexadecimal. The **−s** option scales the three measurements by a given floating point number.

Pk lev dB and *RMS lev dB* are the standard peak and RMS levels measured in dBFS. *RMS Pk dB* and *RMS Tr dB* are peak and trough values of the RMS level measured over a short window (default: 50ms). That can be changed with the **−w** option in seconds from 0.01 to 10.

Crest factor is the ratio of peak to RMS level (note: not in dB).

Flat factor is a measure of the flatness (i.e. consecutive samples with the same value) of the signal at its peak levels (i.e. either *Min level* or *Max level*).

Pk count is the number of occasions (not the number of samples) that the signal attained either *Min level*, or *Max level*. The primary goal of the Peak Count value is to answer the question "has this audio been clipped?", quite possibly as a result of the frowned-upon-by-some but common practice of 'brick wall limiting' in modern mastering. The closer the "Peak Count" is to 1, the higher the confidence that the audio has not been clipped.

The right-hand *Bit-depth* figure is the standard definition of bit-depth, i.e. that all bits other than this number of the most significant bits are always zero. The left-hand figure is the number of bits at the least significant end of those most significant bits that would be sufficient to represent all sample values accurately (including the sign bit).

In mathematical terms, the right-hand figure is the ordinal, counting from the most significant bit, of the least significant bit that is set to one in at least one sample. The left-hand figure is the ordinal, counting from the least significant repeated sign bit across all samples, of the least significant bit that is set to one in at least one sample.

Bit-depths are not intended to be properties of the signal per se but properties of its 2's-complement PCM encoding.

The primary use case of bit-depth measurement concerns manipulation of PCM audio by simple bit shifting, to answer questions such as: "Is it likely that this 24-bit PCM file was created by simply converting a 16-bit PCM file to 24-bit?" or "Can I losslessly shift all the samples in this PCM audio file m-bits left or n-bits right?"

For multichannel audio, an overall figure for each of the above measurements is given and derived from the channel figures as follows: *DC offset*: maximum magnitude; *Max level*, *Pk lev dB*, *RMS Pk dB*, *Bit-depth*: maximum; *Min level*, *RMS Tr dB*: minimum; *RMS lev dB*, *Flat factor*, *Pk count*: average; *Crest factor*: not applicable.

Length s is the duration in seconds of the audio and, unlike **stat**, *Num samples* is equal to the sample rate multiplied by *Length*. *Scale max* is the scaling applied to the first three measurements; specifically, it is the maximum value that could apply to *Max level*. *Window s* is the length of the window used for the peak and trough RMS measurements.

The **−j** option outputs JSON with three fields:

`"channel_count"`
: An integer.

`"overall"`
: An object with a member for each row of the first column of the usual output, which are all numbers except for `"bit_depth"`, which is an array of two numbers.

`"channels"`
: An array of objects with the per-channel values.

To know the overall and the channels' member names, have a look at the output.

Like **stat**, the usual way to measure the characteristics of a single audio file is:

```
sox_ng file.wav −n stats
```

stretch [*factor* [*window* [*fade* [*shift* [*fading*]]]]]

Change the audio duration but not its pitch by cross-fading between short windows of samples. This effect is broadly equivalent to the **tempo** effect with *factor* inverted and *search* set to zero so, in general, its results are comparatively poor; it is retained as it can sometimes outperform **tempo** for small *factor*s.

factor determines the change in length: >1 lengthens and <1 shortens. By default, it is 1 (no change)

window is the length of the cross-fading window in milliseconds with a default of 20.

The *fade* option chooses the type of crossfading: **linear** and **half-cosine** give equal-gain crossfading and cannot clip; **sqrt** and **quarter-cosine** give two kinds of equal-power crossfading.

The *shift* ratio can be from 0 to 1 and its default depends on the stretch factor: 1 when speeding up, 0·8 when slowing down.

The *fading* ratio, from 0 to 0·5, seems to be how much of each window is cross-faded with the adjacent ones. The default value depends on *factor* and *shift*: 1·0 − (*factor* × *shift*) if speeding up, 1·0 − *shift* if slowing down, with a maximum of 0·5.

The duration of **stretch**'s output is slightly longer than the duration of the input multiplied by *factor* as it has to empty the delay line it uses; **tempo** is more precise.

swap Swap stereo channels. If the input is not stereo, pairs of channels are swapped and a possible odd last channel is passed through. E.g., for seven channels, the output order will be 2, 1, 4, 3, 6, 5, 7.

See **remix** for an effect that allows arbitrary channel selection, ordering and mixing.

synth [**−j** *key*] [**−n**] [*length* [*offset* [*phase* [*p1* [*p2* [*p3*]]]]]]
{*type* [*combine* [*fixed*[**,***extra*[**,***mix*]]]] [*freq*[**:**|**+**|**/**|**−***freq2*] [*offset* [*phase* [*p1* [*p2* [*p3*]]]]]]}

synth generates fixed or swept frequency audio tones with various wave shapes and wide-band noise of various colors. Multiple synth effects can be cascaded to produce more complex waveforms and at each stage it is possible to choose whether the generated waveform is mixed with or modulated onto the output of the previous stage, and the audio for each channel in a multichannel audio file can be synthesized independently.

It generates audio at maximum volume (0dBFS), which means that there is a high chance of clipping so, in many cases, you will want to follow it with the **gain** effect to prevent this from happening. (See **Clipping** above.)

Though this effect is used to generate audio, an input file must still be given, the characteristics of which are used to set the synthesized audio length, the number of channels and the sampling rate. However, since the input file's audio is not normally needed, a 'null file' (with the special input filename **−n**) is often given instead and the length specified as a parameter to **synth** or by some other effect that has an associated length.

By default, the tuning used with note notations is equal temperament; the **−j** *key* option selects just intonation, where *key* is a whole number of semitones relative to A (so for example, −9 or 3 selects the key of C) or a note in scientific notation. ,SP By default, the **synth** effect incorporates the functionality of **gain −h** (see the **gain** effect for details); **synth**'s **−n** option may be given to disable this behaviour.

length is the length of audio to synthesize. A value of 0 indicated to use the input length, which is also the default. Note that, if the input is **−n** and the *length* is 0 or absent, it continues generating audio until it is stopped in some other way.

type is one of

sine A sinusoidal wave is the default type and ignores all the *p* parameters.

square A square wave. *p1* sets the percentage of each cycle that is 'on' with a default of 50.

```
|________         |  +1
|        |        |
|________|________|   0
|        |        |
|        |________|  -1
|                 |
0        p1       1
```

triangle
p1 sets the percentage of each cycle that is 'rising' with a default of 50.

```
|     .     |  +1
|    / \    |
|__/___\__|   0
| /      \  |
|/        \|  -1
|           |
0     p1    1
```

sawtooth
A sawtooth wave. With a *phase* of 0 it starts at -1 and rises to 1, and of 10 it starts at -0.9. The offset makes no difference.

```
|     /|  +1
|    / |
|___/__|   0
|  /   |
| /    |  -1
0      1
```

trapezium

The trapezoidal wave starts at -1, rises linearly to 1, stays there, falls linearly to -1, stays there and repeats. *p1* sets the percentage of the cycle in which the wave is rising with a default of 10, *p2* sets the percentage through each cycle at which falling begins with a default of 50 and *p3* sets the percentage through each cycle at which falling ends with a default of 60.

```
|    _______            |+1
|   /       \           |
|__/_________\__________| 0
| /           \         |
|/             \________|-1
|                       |
0   p1     p2   p3      1
```

exp The exponential wave rises from -1 to 1 where it peaks and immediately begins an exponential fall. *p1* sets the position of the maximum with a default of 50. *p2* sets the minimum amplitude in multiples of 2dB down from the maximum with a default of 50 (100dB); values below 50 raise the shoulders of the wave and values above 50 lower the shoulders, increasing the pointedness of the spike.

```
|                            | +1
|             /\             |
|           _'  `_           | 0
|         _-      -_         |
|_____---'          `---_____| f(p2)
|                            |
0             p1              1
```

whitenoise

Random noise with equal power at every frequency. All noise generators ignore the *frequency* and *phase* parameters but if a DC offset is given, the signal's amplitude is automatically adjusted to prevent clipping so, for noise in the range 0 to 1, an offset of 0.5 would give a signal ranging from 0·0 to 1·0 and -0.9 from -1.0 to -0.8

noise is a handy alias for **whitenoise**

tpdfnoise

Noise with a Triangular Probability Density Function.

pinknoise

Random noise with the power at each frequency inversely proportional to the frequency.

brownnoise

Random noise with the power at each frequency inversely proportional to the frequency squared.

pluck A plucked string simulation in which an array of sample values representing a taut string is set in motion with a burst of noise and decayed over time.

A plucked note's *frequency* can be from 27.5 to 4220Hz and the sampling rate must be between 44100 and 48000Hz.

If a DC *offset* is used, the amplitude is automatically adjusted to prevent clipping.

p1 affects the sustain with a default of 40 (2dB per second); higher values give a slower decay and lower values a faster one.

p2 and *p3* are tone controls for the initial excitation, with default values of 20 and 90 and a special case when *p3* is exactly 100. If the *phase* is non-zero, it uses a different kind of random numbers.

If the *offset*, *phase* and *p* parameters are given before the first *type*, they set the default values for all the following stages.

combine is one of

create Puts each stage's output in a new output channel and is the default:

mix Mixes the generated audio 50:50 with the input signal.

amod Amplitude-modulates (multiplies) the input signal by the synthesized one considered as a value from 0 (for the most negative value) to 1 (for the most positive value).

fmod Multiplies the input signal with the synthesized one (ring modulation).

vdelay Mixes the input signal with a delayed version of it using the synthesized signal to modulate the depth of the delay. The following three-part option *fixed*[,*extra*[,*mix*]] specifies the fixed and additional parts of the delay in milliseconds and what percentage of the output consists of the delayed signal from 0 for all input signal to 100 for all delayed signal with a default of 50 (half and half).

The synthesized signal's value from −1 to +1 varies the delay from *fixed* seconds to *fixed* + (0 to *extra*) seconds.

It interpolates linearly between the input samples and can be used to make precision phaser, flanger and chorus-like effects, vibrato and frequency modulation (FM) synthesis (actually phase modulation, as used in the Yamaha DX7).

A chorus-like effect:

```
sox_ng solo.au -d synth sine vdelay 50,2,50 .25 0 75
```

A flanger:

```
sox_ng solo.au -d synth triangle vdelay 0,2,41.52 0.5 0 0
```

freq and *freq2* are the frequencies at the beginning and end of the synthesis and the default frequency is 440Hz.

If *freq2* is given, *length* must also have been given and the generated tone is swept between the given frequencies. The two given frequencies must be separated by one of the characters ':', '+', '/' and '−', which specify the sweep function as follows:

: Linear: the tone changes by a fixed number of hertz per second.

+ Square: a second-order function is used to change the tone.

/ Exponential: the tone changes by a fixed number of semitones per second.

− Exponential: as '/', but the initial phase is always zero, and with stepped (less smooth) frequency changes.

The frequency or frequency range is not used for the noise types.

offset is the bias (DC offset) of the signal in percent; default=0.

phase is the phase shift as a percentage of 1 cycle with a default of 0 (not used for noise).

For example, the following produces a 3-second 48kHz audio file containing a sine wave swept from 300 to 3300Hz:

```
sox_ng -n output.wav synth 3 sine 300-3300
```

Multiple channels can be synthesized by specifying the set of parameters shown between curly braces multiple times; the following puts the swept tone in the left channel and brown noise in the right:

```
sox_ng -n output.wav synth 3 sine 300-3300 brownnoise
```

The following example shows how two synth effects can be cascaded to create a more complex waveform:

```
play_ng -n synth 0.5 sine 200-500 synth 0.5 sine fmod 700-100
```

The following could be used to help tune a guitar:

```
for n in E2 A2 D3 G3 B3 E4; do
   play_ng -n synth 4 pluck $n repeat 2; done
```

tempo [**−q**] [**−m**|**−s**|**−l**] *factor* [*segment*(82) [*search*(14.68) [*overlap*(12)]]]

Change the audio playback speed but not its pitch. This effect uses the WSOLA (Waveform Similarity OverLap and Add) algorithm. The audio is chopped up into segments which are then shifted in the time domain and overlapped (cross-faded) at points where their waveforms are most similar as determined by the measurement of 'least squares'.

By default, linear searches are used to find the best overlapping points. If the optional **−q** parameter is given, tree searches are used instead. This makes the effect work more quickly, but the result may not sound as good. However, if you must improve the processing speed, this generally reduces the sound quality less than reducing the *search* or *overlap* values.

The **−m** option is used to optimize the default values of *segment*, *search* and *overlap* for music processing.

The **−s** option is used to optimize default values of *segment*, *search* and *overlap* for speech processing.

The **−l** option is used to optimize default values of *segment*, *search* and *overlap* for 'linear' processing that tends to cause more noticeable distortion but may be useful when *factor* is close to 1.

If **−m**, **−s** or **−l** is specified, the default value of *segment* is based on *factor*, while default *search* and *overlap* values are based on *segment*. Any values you provide override these default values.

factor gives the ratio of new tempo to the old tempo, so 1.1 speeds the tempo up by 10% and 0.9 slows it down by 10%.

The optional *segment* parameter selects the algorithm's segment size in milliseconds. If no other flags are specified, the default value is 82, which is suited to small changes in the tempo of music. For larger changes (e.g. a factor of 2), 41 may give a better result. The **−m**, **−s**, and **−l** flags cause *segment*'s default value to be adjusted automatically based on *factor*.

The optional *search* parameter gives the audio length in milliseconds over which the algorithm searches for overlapping points. If no other flags are specified, the default value is 14.68. Larger values use more processing time and may or may not produce better results. A practical maximum is half the value of *segment*. Search can be reduced to cut processing time at the risk of degrading output quality. The **−m**, **−s** and **−l** flags cause the search default to be adjusted automatically based on *segment*.

The optional *overlap* parameter gives the segment overlap length in milliseconds. Its default value is 12 but the **−m**, **−s** and **−l** flags automatically adjust it based on the segment size. Increasing *overlap* increases processing time but may increase quality. A practical maximum for *overlap* is a little less then *search*.

See **speed** for an effect that changes tempo and pitch together, **pitch** and **bend** for effects that change pitch only and **stretch** for an effect that changes the tempo using a different algorithm.

treble *gain* [*frequency* [*width*[**s**|**h**|**k**|**o**|**q**]]]
Apply a treble tone control effect. See the description of the **bass** effect for details.

tremolo *speed* [*depth*]
Apply a tremolo (low frequency sinusoidal amplitude modulation) effect to the audio. The frequency of the tremolo in Hz is given by *speed* and its *depth* is a percentage with a default of 40.

trim {*position(+)*}
Cuts out portions of the audio. Any number of *position*s may be given; audio is not sent to the output until the first *position* is reached. The effect then alternates between copying and discarding audio at each *position*. Using a value of 0 for the first *position* parameter allows copying from the beginning of the audio.

For example,

```
sox_ng in.au out.au trim 0 10
```

copies the first ten seconds, while

```
play_ng in.au trim 12:34 =15:00 -2:00
```

and

```
play_ng in.au trim 12:34 2:26 -2:00
```

both play from 12 minutes 34 seconds into the audio up to 15 minutes in (i.e. 2 minutes and 26 seconds long) then resume playing two minutes before the end.

upsample [*factor(2)*]
Upsample the signal by an integer factor: *factor*−1 zero-valued samples are inserted between each pair of input samples. As a result, the original spectrum is replicated into the new frequency space and attenuated. This attenuation can be compensated for by adding **vol** *factor*. The **upsample** effect is typically used in combination with filtering effects.

For a general resampling effect with antialiasing, see **rate**. See **downsample**.

vad [*options*]
The Voice Activity Detector attempts to trim silence and quiet background sounds from the ends of (fairly high resolution i.e. 16-bit, 44−48kHz) recordings of speech. The algorithm currently uses a simple cepstral power measurement to detect voice, so may be fooled by other things, especially music. The effect can trim only from the front of the audio, so in order to trim from the back, the **reverse** effect must also be used. E.g.

```
play_ng speech.wav norm vad
```

to trim from the front,

```
play_ng speech.wav norm reverse vad reverse
```

to trim from the back and

```
play_ng speech.wav norm vad reverse vad reverse
```

to trim from both ends. The use of the **norm** effect is recommended, but remember that neither **reverse** nor **norm** is suitable for use with streamed audio.

Options
Default values are shown in parentheses, the allowed range in square brackets.

−t *num* (7) [0 − 20]
The measurement level used to trigger activity detection. This might need to be changed depending on the noise level, signal level and other characteristics of the input audio.

−T *num* (0.25) [0.01 − 1]
The time constant (in seconds) used to help ignore short bursts of sound.

−s *num* (1) [0.1 − 4]
The amount of audio (in seconds) to search for quieter/shorter bursts of audio to include prior to the detected trigger point.

−g *num* (0.25) [0.1 − 1]
Allowed gap (in seconds) between quieter/shorter bursts of audio to include prior to the detected trigger point.

−p *num* (0) [0 − 4]
The amount of audio (in seconds) to preserve before the trigger point and any found quieter/shorter bursts.

Advanced Options
These allow fine tuning of the algorithm's internal parameters.

−b *num* (0.35) [0.1 − 10]
The algorithm uses adaptive noise estimation/reduction in order to detect the start of the wanted audio. This option sets the time in seconds for the initial noise estimate.

−N *num* (0.1) [0.1 − 10]
Time constant used by the adaptive noise estimator when the noise level is increasing.

−n *num* (0.01) [0.001 − 0.1]
Time constant used by the adaptive noise estimator when the noise level is decreasing.

−r *num* (1.35) [0 − 2]
Amount of noise reduction to use in the detection algorithm.

−f *num* (20) [5 − 50]
Frequency of the algorithm's processing/measurements.

−m *num* (0.1) [0.01 − 1]
Measurement duration. By default, it is twice the measurement period; i.e. with 50% overlap, but if you set **-f**, you also need to change **−m** to 2 divided by its value to keep a 50% overlap.

−M *num* (0.4) [0-1 − 1]
Time constant used to smooth spectral measurements.

−h *freq* (50) [10 −]
'Brick-wall' frequency of the high-pass filter applied at the detector algorithm's input.

−l *freq* (6000) [1000 −]
'Brick-wall' frequency of the low-pass filter applied at the detector algorithm's input.

−H *freq* (150) [10 −]
'Brick-wall' frequency of the high-pass lifter used in the detector algorithm.

−L *freq* (2000) [1000 −]
'Brick-wall' frequency of the low-pass lifter used in the detector algorithm.

See the **silence** effect.

vol *gain* [*type* [*limiter−gain*]]
Apply amplification or attenuation to the audio signal. Unlike **−v**, which is used for balancing multiple input files as they enter the SoX effects processing chain, **vol** is an effect like any other so can be applied anywhere in the processing chain and several times if necessary.

The amount to change the volume is given by *gain* which is interpreted, according to the given *type*, as follows: if *type* is **amplitude** (or is omitted), *gain* is an amplitude ratio (voltage or linear), if **power**, a power ratio (wattage or voltage squared) and if **dB**, a power change in dB.

When *type* is **amplitude** or **power**, a *gain* of 1 leaves the volume unchanged, less than 1 decreases it, and greater than 1 increases it; a negative *gain* inverts the audio signal in addition to adjusting its volume.

When *type* is **dB**, a *gain* of 0 leaves the volume unchanged, less than 0 decreases it and greater than 0 increases it.

See [4] for a detailed discussion on electrical (and hence audio signal) voltage and power ratios.

Beware of **Clipping** when the increasing the volume.

The *gain* and the *type* parameters can be concatenated if desired, e.g. **vol 10dB**.

An optional *limiter−gain* value can be specified and should be a value much less than 1 (e.g. 0·05 or 0·02) and is used only on peaks to prevent clipping. Not specifying this parameter causes no limiter to be used. In verbose mode, this effect displays the percentage of the audio that needed to be limited.

See **gain** for a volume-changing effect with different capabilities and **compand** for a dynamic range compression/expansion/limiting effect.

ENVIRONMENT

SoX reacts to the following environment variables. To set them on Unix with most shells, use, for example:

```
AUDIODRIVER=oss
export AUDIODRIVER
play_ng ...
```

with Unix csh:

```
setenv AUDIODRIVER oss
```

or, on Microsoft Windows:

```
set AUDIODRIVER=waveaudio
```

MS-Windows GUI: via Control Panel : System : Advanced : Environment Variables

Mac OS X GUI: Refer to Apple's Technical Q&A QA1067 document.

AUDIODRIVER

On some systems, SoX may have more than one type of audio driver, e.g. ALSA and OSS or SUNAU and AO and they can have more than one audio device (a.k.a. 'sound card'). If more than one audio driver has been built into SoX and the default selected by SoX when recording or playing is not the one that is wanted, the **AUDIODRIVER** environment variable can be used to override the default. For example, on Unix systems:

```
AUDIODRIVER=oss
export AUDIODRIVER
play_ng ...
```

If it is unset, SoX tries to use, in order, **coreaudio**, **pulseaudio**, **alsa**, **waveaudio**, **sndio**, **oss**, **sunau** and **ao**. For further details on these, see their entries in **soxformat_ng**(7).

AUDIODEV

Override the default audio device, e.g.

```
AUDIODEV=/dev/dsp2
export AUDIODEV
play_ng ...
sox_ng ... −t oss
```

or

```
AUDIODEV=hw:soundwave,1,2
export AUDIODEV
play_ng ...
sox_ng ... −t alsa
```

If AUDIODEV is unset and the audio driver is **oss**, SoX also responds to the standard environment variable **OSS_AUDIODEV**.

LADSPA_PATH

A colon-separated list of directories in which to search for LADSPA plugins. The default depends on how SoX was built but on Unix it defaults to `/usr/lib/ladspa`, on MacOS/X to `/Library/Audio/Plug-Ins/LADSPA`. Windows doesn't have a "usual place" for LADSPA plugins, but Ardour puts them in `C:\Program Files\Ardour6\lib\ardour6\ladspa` or similar.

LD_LIBRARY_PATH

When searching for the dynamic libraries in which most effects and format handlers may be stored, according to how your SoX was built, look in this colon-separated list of directories before the default location.

MIXERDEV

When playing a file, use the specified mixer device for the 'v' and 'V' volume control keys.

SOX_OPTS

Provide alternative default values for SoX's global options. For example:

```
SOX_OPTS="--buffer 20000 --play-rate-arg -hs --temp /mnt/temp"
export SOX_OPTS
```

Note that setting **SOX_OPTS** can create unwanted changes in the behaviour of scripts or other programs that invoke SoX. **SOX_OPTS** might best be used for things that reflect the environment in which SoX is being run and enabling options such as **−−no−clobber** by default might be handled better using a shell alias since that will not affect SoX's operation in scripts or when it is used by other programs.

One way to ensure that scripts and programs cannot be affected by **SOX_OPTS** is to clear **SOX_OPTS** at the start of the script, but this loses the benefit of **SOX_OPTS** carrying system-wide defaults.

TEMP and **TMP**

On Windows, tmpfile() is broken — it creates the file in the root directory of the current drive instead of in a valid temporary directory — but if **TEMP** or **TMP** are set, it creates them in the directory indicated, otherwise in the current directory. To force use of tmpfile(), use **--temp** .

Alternatively, and on Unix, use **--temp** (see **Global Options** above).

EXIT STATUS

SoX exits 0 when there is no error, 1 if there is a problem with the command-line parameters or 2 if an error occurs during audio processing.

BUGS

Please report any bugs found in this version of SoX to the mailing list <sox-ng@groups.io>.

CITATION

To cite SoX in publications please use:

```
Lance Norskog, Chris Bagwell et al. (2015).
SoX: Sound eXchange, the Swiss Army knife of audio manipulation.
URL http://sox.sourceforge.net
```

A BibTeX entry for SoX users is

```
@manual{SoX2015,
  title = "SoX: Sound eXchange, the Swiss Army knife of audio manipulation",
  author = "Norskog, Lance and Bagwell, Chris and others",
  edition = "14.4.2",
  year = 2015,
  url = "http://sox.sourceforge.net",
}
```

SEE ALSO

soxi_ng(1), **soxformat_ng**(7), **libsox_ng**(3) **audacity**(1), **ecasound**(1), **gnuplot**(1), **octave**(1).
The sox_ng web site at https://codeberg.org/sox_ng/sox_ng
SoX scripting examples at https://codeberg.org/sox_ng/sox_ng/src/branch/main/scripts

References

[1] R. Bristow-Johnson, *Cookbook formulae for audio EQ biquad filter coefficients*, https://www.w3.org/TR/audio-eq-cookbook

[2] Wikipedia, *Q-factor*, http://en.wikipedia.org/wiki/Q_factor

[3] Scott Lehman, *Effects Explained*, https://codeberg.org/sox_ng/Effects-Explained

[4] Wikipedia, *Decibel*, http://en.wikipedia.org/wiki/Decibel

[5] Richard Furse, *Linux Audio Developer's Simple Plugin API*, http://www.ladspa.org

[6] Richard Furse, *Computer Music Toolkit*, http://www.ladspa.org/cmt/overview.html

[7] Steve Harris, *LADSPA plugins*, http://plugin.org.uk

LICENSE

AUTHORS

Lance Norskog, Chris Bagwell and many others listed in the AUTHORS file that is distributed with the source code.