

NAME

SoX – Sound eXchange, the Swiss Army knife of audio manipulation

DESCRIPTION

This manual describes SoX audio effects; the SoX manual set starts with **sox**(1).

In addition to converting and playing audio files, SoX can be used to invoke a number of audio ‘effects’. Multiple effects may be applied by specifying them one after another at the end of the SoX command line. Note that applying multiple effects in real-time (i.e. when playing audio) is likely to need a high performance computer; stopping other applications may alleviate performance issues should they occur.

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 multi-track recording.

In the descriptions that follow, brackets [] are used to denote parameters that are optional, braces { } to denote those that are both optional and repeatable, and angle brackets < > to denote those that are repeatable but not optional. Where applicable, default values for optional parameters are shown in parenthesis ().

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

centre[**k**]

See *frequency*.

frequency[**k**]

A frequency in Hz, or, if appended with ‘k’, kHz.

gain A power gain in dB. Zero gives no gain; less than zero gives an attenuation.

width[**h**|**k**|**o**|**q**]

Used to specify the band-width 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:

	<i>Method</i>	<i>Notes</i>
h	Hz	
k	kHz	
o	Octaves	
q	Q-factor	See [2]

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

To see if SoX has support for an optional effect, enter **sox -h** and look for its name under the list: ‘EFFECTS’.

SOX EFFECTS

allpass *frequency*[**k**] *width*[**h**|**k**|**o**|**q**]

Apply a two-pole all-pass filter with central frequency (in Hz) *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].

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

band [**-n**] *center*[**k**] [*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 *center* + *width* and *center* – *width* will be half of their original amplitudes. **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 un-pitched audio (e.g. percussion). **Warning:** **-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, i.e. peaking at the *center* frequency and settling around it.

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

See also **filter** for a bandpass filter with steeper shoulders.

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

Apply a two-pole Butterworth band-pass or band-reject filter with central frequency *frequency*, and (3dB-point) band-width *width*. The **-c** option applies only to **bandpass** and selects a constant skirt gain (peak gain = Q) instead of the default: 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 also **filter** for a bandpass filter with steeper shoulders.

bandreject *frequency*[**k**] *width*[**h|k|o|q**]

Apply a band-reject filter. See the description of the **bandpass** effect for details.

bass|treble *gain* [*frequency*[**k**] [*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 hi-fi's tone-controls. This is also known as shelving equalisation (EQ).

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

If desired, 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 value is 100 Hz (for **bass**) or 3 kHz (for **treble**).

width determines how steep is the filter's shelf transition. In addition to the common width specification methods described above, 'slope' (the default, or if appended with 's') may be used. The useful range of 'slope' is about 0.3, for a gentle slope, to 1 (the maximum), for a steep slope; the default value is 0.5.

The filters are described in detail in [1].

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

See also **equalizer** for a peaking equalisation effect.

chorus *gain-in gain-out* <*delay decay speed depth* **-s|-t**>

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

Chorus resembles an echo effect with a short delay, but whereas with echo the delay is constant, with chorus, it is varied using sinusoidal or triangular modulation. The modulation depth defines the range the modulated delay is played before or after the delay. Hence the delayed sound will sound slower or faster, that is the delayed sound tuned around the original one, like in a chorus where some vocals are slightly off key. See [3] for more discussion of the chorus effect.

Each four-tuple parameter *delay/decay/speed/depth* gives the delay in milliseconds and the decay (relative to *gain-in*) with a modulation speed in Hz using *depth* in milliseconds. The modulation is either sinusoidal (**-s**) or triangular (**-t**). *Gain-out* is the volume of the output.

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

Two delays of the original samples:

```
play 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 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
```

compond *attack1,decay1*{*attack2,decay2*}
[*soft-knee-dB*][:*in-dB1*][:*out-dB1*][:*in-dB2*][:*out-dB2*]
[*gain* [*initial-volume-dB* [*delay*]]]

Compond (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 (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. Where more than one pair of attack/decay parameters are specified, each input channel is compounded 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 compounded (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 will be 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 permits the user to 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 properly 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 analysed 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 effectively operate in a 'predictive' rather than a reactive mode. A typical value is **0.2** seconds.

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

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 asz.au asz-car.au compond 0.3,1 6:-70,-60,-20 -5 -90 0.2
```

The transfer function ('6:-70,...') says that very soft sounds (below -70dB) will remain unchanged. This will stop the compander from boosting the volume on 'silent' passages such as between movements. However, sounds in the range -60dB to 0dB (maximum volume) will be boosted so that the 60dB dynamic range of the original music will be 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) has the effect of causing the compander to react a bit more quickly to sudden volume changes.

See also **mcompond** 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. **contrast** is often used in conjunction with the **norm** effect as follows:

```
sox infile outfile norm -i contrast
```

dcshift *shift* [*limitergain*]

DC Shift the audio, with basic linear amplitude formula. This is most useful if your audio tends to not be centered around a value of 0. Shifting it back will allow you to get the most volume adjustments without clipping.

The first option is the *dcshift* value. It is a floating point number that indicates the amount to shift.

An optional *limitergain* 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; i.e.

```
sox -n out.au synth 5 sin %0 50 highpass 10
```

deemph

Apply a treble attenuation shelving filter to audio in audio-CD format. The frequency response of pre-emphasized recordings is rectified. The filter is defined in the standard document ISO 908.

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

See also the **bass** and **treble** shelving equalisation effects.

delay {*length*}

Delay one or more audio channels. *length* can specify a time or, if appended with an ‘s’, a number of samples. For example, **delay 1-5 0 0.5** delays the first channel by 1.5 seconds, the third channel by 0.5 seconds, and leaves the second channel (and any other channels that may be present) undelayed. The following (one long) command plays a chime sound:

```
play -n synth sin %-21.5 sin %-14.5 sin %-9.5 sin %-5.5 \
sin %-2.5 sin %2.5 gain -5.4 fade h 0.008 2 1.5 \
delay 0 .27 .54 .76 1.01 1.3 remix - fade h 0.1 2.72 2.5
```

dither [*depth*]

Apply dithering to the audio. Dithering deliberately adds digital white noise to the signal in order to mask audible quantization effects that can occur if the output sample size is less than 24 bits. By default, the amount of noise added is ½ bit; the optional *depth* parameter is a (linear or voltage) multiplier of this amount.

This effect should not be followed by any other effect that affects the audio.

earwax

Makes audio easier to listen to on headphones. Adds ‘cues’ to 44.1kHz stereo (i.e. audio CD format) audio so that when listened to on headphones the stereo image is moved from inside your head (standard for headphones) to outside and in front of the listener (standard for speakers). See <http://www.geocities.com/beinges> for a full explanation.

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

Add echoing to the audio. Echoes are reflected sound and can occur naturally amongst mountains (and sometimes large buildings) when talking or shouting; digital echo effects emulate this behaviour and are often used to help fill out the sound of a single instrument or vocal. The time difference between the original signal and the reflection is the ‘delay’ (time), and the loudness of the reflected signal is the ‘decay’. Multiple echoes can have different delays and decays.

Each given *delay decay* pair gives the delay in milliseconds and the decay (relative to gain-in) of that echo. Gain-out is the volume of the output. For example: This will make it sound as if there are twice as many instruments as are actually playing:

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

If the delay is very short, then it sound like a (metallic) robot playing music:

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

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

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

One mountain more, and:

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

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

Add a sequence of echoes to the audio. Each *delay decay* pair gives the delay in milliseconds and the decay (relative to gain-in) of that echo. Gain-out is the volume of the output.

Like the echo effect, echos stand for ‘ECHO in Sequel’, that is the first echos takes the input, the second the input and the first echos, the third the input and the first and the second echos, ... and so on. Care should be taken using many echos; a single echos has the same effect as a single echo.

The sample will be bounced twice in symmetric echos:

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

The sample will be bounced twice in asymmetric echos:

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

The sample will sound as if played in a garage:

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

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

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

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

In order to produce complex equalisation 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.

See also **bass** and **treble** for shelving equalisation effects.

fade [*type*] *fade-in-length* [*stop-time*] [*fade-out-length*]

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

For fade-ins, this starts from the first sample and ramps the volume of the audio from 0 to full volume over *fade-in-length* seconds. Specify 0 seconds if no fade-in is wanted.

For fade-outs, the audio will be truncated at *stop-time* and the volume will be ramped from full volume down to 0 starting at *fade-out-length* seconds before the *stop-time*. If *fade-out-length* is not specified, it defaults to the same value as *fade-in-length*. No fade-out is performed if *stop-time* is not specified. If the file length can be determined from the input file header and length-changing effects are not in effect, then **0** may be specified for *stop-time* to indicate the usual case of a

fade-out that ends at the end of the input audio stream.

All times can be specified in either periods of time or sample counts. To specify time periods use the format hh:mm:ss.frac format. To specify using sample counts, specify the number of samples and append the letter 's' to the sample count (for example '8000s').

An optional *type* can be specified to change the type of envelope. Choices are **q** for quarter of a sine wave, **h** for half a sine wave, **t** for linear slope, **l** for logarithmic, and **p** for inverted parabola. The default is logarithmic.

filter [*low*]-[*high*] [*window-len*] [*beta*]

Apply a sinc-windowed lowpass, highpass, or bandpass filter of given window length to the signal. *low* refers to the frequency of the lower 6dB corner of the filter. *high* refers to the frequency of the upper 6dB corner of the filter.

A low-pass filter is obtained by leaving *low* unspecified, or 0. A high-pass filter is obtained by leaving *high* unspecified, or 0, or greater than or equal to the Nyquist frequency.

The *window-len*, if unspecified, defaults to 128. Longer windows give a sharper cut-off, smaller windows a more gradual cut-off.

The *beta* parameter determines the type of filter window used. Any value greater than 2 is the beta for a Kaiser window. $\text{Beta} \leq 2$ selects a Nuttall window. If unspecified, the default is a Kaiser window with beta 16.

In the case of Kaiser window ($\text{beta} > 2$), lower betas produce a somewhat faster transition from pass-band to stop-band, at the cost of noticeable artifacts. A beta of 16 is the default, beta less than 10 is not recommended. If you want a sharper cut-off, don't use low beta's, use a longer sample window. A Nuttall window is selected by specifying any '*beta*' ≤ 2 , and the Nuttall window has somewhat steeper cut-off than the default Kaiser window. You will probably not need to use the beta parameter at all, unless you are just curious about comparing the effects of Nuttall vs. Kaiser windows.

flanger [*delay*] [*depth*] [*regen*] [*width*] [*speed*] [*shape*] [*phase*] [*interp*]

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

All parameters are optional (right to left).

	Range	Default	Description
<i>delay</i>	0 – 10	0	Base delay in milliseconds.
<i>depth</i>	0 – 10	2	Added swept delay in milliseconds.
<i>regen</i>	-95 – 95	0	Percentage regeneration (delayed signal feedback).
<i>width</i>	0 – 100	71	Percentage of delayed signal mixed with original.
<i>speed</i>	0.1 – 10	0.5	Sweeps per second (Hz).
<i>shape</i>		sin	Swept wave shape: sine triangle .
<i>phase</i>	0 – 100	25	Swept wave percentage phase-shift for multi-channel (e.g. stereo) flange; 0 = 100 = same phase on each channel.
<i>interp</i>		lin	Digital delay-line interpolation: linear quadratic .

gain *dB-gain*

Apply an amplification or an attenuation to the audio signal. This is an alias for the **vol** effect—handy for those who prefer to work in dBs by default.

highpass|lowpass [-1|-2] *frequency*[k] [*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 also **filter** for filters with a steeper roll-off.

key **[−q]** *shift* [*segment* [*search* [*overlap*]]]

Change the audio key (i.e. pitch but not tempo) using a WSOLA algorithm.

shift gives the key shift as positive or negative ‘cents’ (i.e. 100ths of a semitone). See the **tempo** effect for a description of the other parameters.

See also **pitch** for a similar effect.

ladspa 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. Only plugins with at most one audio input and one audio output port can be used. If found, the environment variable LADSPA_PATH will be used as search path for plugins.

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

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

mcompand "*attack1,decay1*{*attack2,decay2*}

[soft-knee-dB:]in-dB1[,out-dB1]{,in-dB2,out-dB2}

[gain [initial-volume-dB [delay]]]" {xover-freq[k] "attack1,..."}]

The multi-band compander is similar to the single-band compander but the audio is first divided into bands using Butterworth cross-over filters and a separately specifiable compander 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 *xover-freq*; these can be repeated to create multiple bands.

For example, the following (one long) command shows how multi-band companding is typically used in FM radio:

```
play track1.wav gain -3 filter 8000- 32 100 mcompand \  
"0.005,0.1 -47,-40,-34,-34,-17,-33" 100 \  
"0.003,0.05 -47,-40,-34,-34,-17,-33" 400 \  
"0.000625,0.0125 -47,-40,-34,-34,-15,-33" 1600 \  
"0.0001,0.025 -47,-40,-34,-34,-31,-31,-0,-30" 6400 \  
"0,0.025 -38,-31,-28,-28,-0,-25" \  
gain 15 highpass 22 highpass 22 filter -17500 256 \  
gain 9 lowpass -1 17801
```

The audio file is played with a simulated FM radio sound (or broadcast signal condition if the low-pass filter at the end is skipped). Note that the pipeline is set up with US-style 75us preemphasis.

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

mixer **[−l|−r|−f|−b|−1|−2|−3|−4|*n*{,*n*}]**

Reduce the number of audio channels by mixing or selecting channels, or increase the number of channels by duplicating channels. Note: 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).

This effect is automatically used when the number of input channels differ from the number of

output channels. When reducing the number of channels it is possible to manually specify the **mixer** effect and use the **-l**, **-r**, **-f**, **-b**, **-1**, **-2**, **-3**, **-4**, options to select only the left, right, front, back channel(s) or specific channel for the output instead of averaging the channels. The **-l**, and **-r** options will do averaging in quad-channel files so select the exact channel to prevent this.

The **mixer** effect can also be invoked with up to 16 numbers, separated by commas, which specify the proportion (0 = 0% and 1 = 100%) of each input channel that is to be mixed into each output channel. In two-channel mode, 4 numbers are given: $l \rightarrow l$, $l \rightarrow r$, $r \rightarrow l$, and $r \rightarrow r$, respectively. In four-channel mode, the first 4 numbers give the proportions for the left-front output channel, as follows: $lf \rightarrow lf$, $rf \rightarrow lf$, $lb \rightarrow lf$, and $rb \rightarrow lf$. The next 4 give the right-front output in the same order, then left-back and right-back.

It is also possible to use the 16 numbers to expand or reduce the channel count; just specify 0 for unused channels.

Finally, certain reduced combination of numbers can be specified for certain input/output channel combinations.

In Ch	Out Ch	Num	Mappings
2	1	2	$l \rightarrow l$, $r \rightarrow l$
2	2	1	adjust balance
4	1	4	$lf \rightarrow l$, $rf \rightarrow l$, $lb \rightarrow l$, $rb \rightarrow l$
4	2	2	$lf \rightarrow l \& rf \rightarrow r$, $lb \rightarrow l \& rb \rightarrow r$
4	4	1	adjust balance
4	4	2	front balance, back balance

See also **remix** for a mixing effect that handles any number of channels.

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** will write out a noise profile to *profile-file*, or to stdout if no *profile-file* or if ‘-’ is given. E.g.

```
sox speech.au -n trim 0 1.5 noiseprof speech.noise-profile
```

To actually remove the noise, run SoX again, this time with the **noisered** effect; **noisered** will reduce noise according to a noise profile (which was generated by **noiseprof**), from *profile-file*, or from stdin if no *profile-file* or if ‘-’ is given. E.g.

```
sox speech.au cleaned.au 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 will 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 noisy.au -n trim 0 1 noiseprof | play noisy.au noisered
```

norm [-i] [*level*]

Normalise audio to 0dB FSD or to a given level relative to 0dB. Requires temporary file space to store the audio to be normalised.

To create a normalised copy of an audio file,

```
sox infile outfile norm
```

can be used, though note that if ‘infile’ has a simple encoding (e.g. PCM), then

```
sox infile outfile vol `sox infile -n stat -v 2>&1`
```

(on systems that support this construct) might be quicker to execute (though perhaps not to type!) as it doesn’t require a temporary file.

For a more complex example, suppose that ‘effect1’ performs some unknown or unpredictable attenuation and that ‘effect2’ requires up to 10dB of headroom, then

```
sox infile outfile effect1 norm -10 effect2 norm
```

gives both effect2 and the output file the highest possible signal levels.

Normally, audio is normalised based on the level of the channel with the highest peak level, which means that whilst all channels are adjusted, only one channel attains the normalised level. If the **-i** option is given, then each channel is treated individually and will attain the normalised level.

In most cases, **norm -3** should be the maximum level used at the output file (to leave headroom for playback-resampling, etc.). See also the discussions of clipping and Replay Gain in **sox(1)**.

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.

pad { *length*[*@position*] }

Pad the audio with silence, at the beginning, the end, or any specified points through the audio. Both *length* and *position* can specify a time or, if appended with an ‘s’, a number of samples. *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 than 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 padding 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, specify either the end position or specify a zero-length pad at the start.

pan *direction*

Pan the audio from one channel to another. This is done by changing the volume of the input channels so that it fades out on one channel and fades-in on another. If the number of input channels is different then the number of output channels then this effect tries to intelligently handle this. For instance, if the input contains 1 channel and the output contains 2 channels, then it will create the missing channel itself. The *direction* is a value from -1 to 1. -1 represents far left and 1 represents far right. Numbers in between will start the pan effect without totally muting the opposite channel.

phaser *gain-in gain-out delay decay speed* [-s|-t]

Add a phasing effect to the audio. See [3] for a detailed description of phasing.

delay/decay/speed gives the delay in milliseconds and the decay (relative to *gain-in*) with a modulation speed in Hz. The modulation is either sinusoidal (-s) —preferable for multiple instruments, or triangular (-t) —gives single instruments a sharper phasing effect. The decay should be less than 0.5 to avoid feedback, and usually no less than 0.1. *Gain-out* is the volume of the output.

For example:

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

Gentler:

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

A popular sound:

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

More severe:

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

rate [-q|-l|-m|-h|-v] [RATE[k]]

Change the audio sampling rate (i.e. resample the audio) using a quality level as follows:

	<i>Quality</i>	<i>BW %</i>	<i>Rej dB</i>	<i>Typical Use</i>
-q	quick & dirty	n/a	≈30 @ Fs/4	playback on ancient hardware
-l	low	80	100	playback on old hardware
-m	medium	99	100	audio playback
-h	high	99	125	16-bit mastering (use with dither)
-v	very high	99	175	24-bit mastering

where **BW %** is the percentage of the audio band that is preserved (based on the 3dB point) during sample rate conversion, and **Rej dB** is the level of noise rejection. The default quality level is 'high' (**-h**). The **-q** algorithm uses cubic interpolation; the others use linear-phase bandwidth-limited interpolation.

This effect is invoked automatically if SoX's **-r** option specifies a rate that is different to that of the input file(s). Alternatively, this effect may be invoked with the output rate parameter *RATE* and SoX's **-r** option need not be given. For example, the following two commands are equivalent:

```
sox input.au -r 48k output.au bass -3
sox input.au output.au bass -3 rate 48k
```

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

See also **resample**, **polyphase** and **rabbit** for other sample-rate changing effects.

remix [-a|-m|-p] <out-spec>

out-spec = *in-spec*{,*in-spec*} | **0**

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*: 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 input.au output.au 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 input.au output.au remix 1-3,7 3
```

creates a 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 input.au output.au remix -
```

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

By default, where an output channel is mixed from multiple (n) input channels, each input channel will be 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). This is one of the letters **p**, **i**, or **v**, followed by a volume number, the meaning of which depends on the given letter and is defined as follows:

Letter	Volume number	Notes
p	power adjust in dB	0 = no change
i	power adjust in dB	As 'p', but invert the audio
v	voltage multiplier	1 = no change, 0.5 \approx 6dB attenuation, 2 \approx 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 may be in other out-specs). The **-a** (automatic) option however, can be given to retain the automatic scaling in this case. For example,

```
sox input.au output.au remix 1,2 3,4v0.8
```

results in channel level multipliers of 0.5,0.5 1,0.8, whereas

```
sox input.au output.au remix -a 1,2 3,4v0.8
```

results in channel level multipliers of 0.5,0.5 0.5,0.8.

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

```
sox input.au output.au remix -m 1,2 3,4v0.8
```

results in channel level multipliers of 1,1 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.au* is stereo, then

```
sox input.au output.au remix 1,2i
```

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

If the **-p** option is given, then any automatic $1/n$ scaling is replaced by $1/\sqrt{n}$ ('power') scaling; this gives a louder mix but one that might occasionally clip.

* * *

One typical 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). Where more than a few channels are involved, a script such as the following is useful:

```
#!/bin/sh                                     # This is a Bourne shell script
chans=`soxi -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 "$1" "$out" remix $chans
    chans=`expr $chans - 1`
done
```

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

See also **mixer** and **swap** for similar effects.

repeat *count*

Repeat the entire audio *count* times. 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|--wet-only] [*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 will usually make it sound more natural. See [3] for a detailed description of reverberation.

Note that this effect increases both the volume and the length of the audio, so to prevent clipping in these domains, a typical invocation might be:

```
play dry.au gain -3 pad 0 3 reverb
```

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

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

Removes silence from the beginning, middle, or end of the audio. Silence is anything below a specified threshold.

The *above-periods* value is used to indicate if audio should be trimmed at the beginning of the audio. A value of zero indicates no silence should be trimmed from the beginning. When specifying an non-zero *above-periods*, it trims audio up until it finds non-silence. Normally, when trimming silence from beginning of audio the *above-periods* will be 1 but it can be increased to higher values to trim all audio up to a specific count of non-silence periods. For example, if you had an audio file with two songs that each contained 2 seconds of silence before the song, you could specify an *above-period* of 2 to strip out both silence periods and the first song.

When *above-periods* is non-zero, you must also specify a *duration* and *threshold*. *Duration* indicates the amount of time that non-silence must be detected before it stops trimming audio. By increasing the duration, burst of noise can be treated as silence and trimmed off.

Threshold is used to indicate what sample value you should treat as 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 account for background noise.

When optionally trimming silence from the end of the audio, you specify a *below-periods* count. In this case, *below-period* means to remove all audio after silence is detected. Normally, this will be a value 1 of but it can be increased to skip over periods of silence that are wanted. For example, if you have a song with 2 seconds of silence in the middle and 2 second at the end, you could set below-period to a value of 2 to skip over the silence in the middle of the audio.

For *below-periods*, *duration* specifies a period of silence that must exist before audio is not copied any more. By specifying a higher duration, silence that is wanted can be left in the audio. For example, if you have a song with an expected 1 second of silence in the middle and 2 seconds of silence at the end, a duration of 2 seconds could be used to skip over the middle silence.

Unfortunately, you must know the length of the silence at the end of your audio file to trim off silence reliably. A work around is to use the **silence** effect in combination with the **reverse** effect. By first reversing the audio, you can use the *above-periods* to reliably trim all audio from what looks like the front of the file. Then reverse the file again to get back to normal.

To remove silence from the middle of a file, specify a *below-periods* that is negative. This value is then treated as a positive value and is also used to indicate 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 option **-l** indicates that *below-periods duration* length of 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 *period* counts are in units of samples. *Duration* counts may be in the format of hh:mm:ss.frac, or the exact count of samples. *Threshold* numbers may be suffixed with **d** to indicate the value is in decibels, or **%** to indicate a percentage of maximum value of the sample value (**0%** specifies pure digital silence).

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

```
rec parameters filename other-effects silence 1 5 2%
```

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 up, less than 1 slows down, or, if appended with the letter ‘c’, the number of cents (i.e. 100ths of a semitone) by which the pitch (and tempo) should be adjusted: greater than 0 increases, less than 0 decreases.

By default, the speed change is performed by resampling with the **rate** effect 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.

spectrogram [options]

Create a spectrogram of the audio. This effect is optional; type **sox --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. Z-axis values are represented by the colour (or intensity) of the pixels in the X-Y plane.

This effect supports only one channel; for multi-channel input files, use either SoX’s **-c 1** option with the output file (to obtain a spectrogram on the mix-down), or the **remix n** effect to select a particular channel. Be aware though, that both of these methods affect the audio in the effects chain.

-x num X-axis pixels/second, default 100. This controls the width 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 quantisation reasons; if so, SoX will report the actual number used (viewable when **--verbose** is in effect).

The maximum width of the spectrogram is 999 pixels; if the audio length and the given **-x** number are such that this would be exceeded, then the spectrogram (and the effects chain) will be truncated. To move the spectrogram to a point later in the audio stream, first invoke the **trim** effect; e.g.

```
sox audio.ogg -n trim 1:00 spectrogram
```

starts the spectrogram at 1 minute through the audio.

-y num Y-axis resolution (1 – 4), default 2. This controls the height of the spectrogram; *num* can be from 1 (low frequency resolution) to 4 (high frequency resolution). For values greater than 2, the resulting image may be too tall to display on the screen; if so, a graphic manipulation package (such as **ImageMagick**(1)) can be used to re-size the image.

To increase the frequency resolution without increasing the height of the spectrogram, the

rate effect may be invoked to reduce the sampling rate of the signal before invoking **spectrogram**; e.g.

```
sox audio.ogg -r 4k -n rate spectrogram
```

allows detailed analysis of frequencies up to 2kHz (half the sampling rate).

-z num Z-axis (colour) 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.

-q num Sets the Z-axis quantisation, i.e. the number of different colours (or intensities) in which to render Z-axis values. A small number (e.g. 4) will give a ‘poster’-like effect making it easier to discern magnitude bands of similar level. Smaller numbers also usually result in smaller PNG files. The number given specifies the number of colours to use inside the Z-axis range; two colours are reserved to represent out-of-range values.

-w name

Window: Hann (default), Hamming, Bartlett, Rectangular or Kaiser. The spectrogram is produced using the Discrete Fourier Transform (DFT) algorithm. A significant parameter to this algorithm is the choice of ‘window function’. By default, SoX uses the Hann window which has good all-round frequency-resolution and dynamic-range properties. For better frequency resolution (but lower dynamic-range), select a Hamming window; for higher dynamic-range (but poorer frequency-resolution), select a Kaiser window. Bartlett and Rectangular windows are also available. Selecting a window other than Hann will usually require a corresponding **-z** setting.

-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 colour).

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

-p num Permute the colours in a colour or hybrid palette. The *num* parameter (from 1 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.

-t text Set the image title—text to display above the spectrogram.

-c text Set the image comment—text to display below and to the left of the spectrogram.

-o text Name of the spectrogram output PNG file, default ‘spectrogram.png’.

For example, let’s see what the spectrogram of a swept triangular wave looks like:

```
sox -n -n synth 6 tri 10k:14k spectrogram -z 100 -w k
```

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

splice { *position*[,*excess*[,*leeway*]] }

Splice together audio sections. 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.

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) of audio 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).

For example, a long song begins with two verses which start (as determined e.g. by using the **play** command with the **trim** (*start*) effect) at times 0:30.125 and 1:03.432. The following commands cut out the first verse:

```
sox too-long.au part1.au trim 0 30.130
```

(5 ms excess, after the first verse starts)

```
sox long.au part2.au trim 1:03.422
```

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

```
sox part1.au part2.au just-right.au splice 30.130
```

Provided your arithmetic is good enough, multiple splices can be performed with a single **splice** invocation. For example:

```
#!/bin/sh
# Audio Copy and Paste Over
# acpo infile copy-start copy-stop paste-over-start outfile
# All times measured in samples.
rate=`soxi -r "$1" `
e=`expr $rate '*' 5 / 1000` # Using default excess
l=$e                        # and leeway.
sox "$1" piece.au trim `expr $2 - $e - $l`s \
    `expr $3 - $2 + $e + $l + $e`s
sox "$1" part1.au trim 0 `expr $4 + $e`s
sox "$1" part2.au trim `expr $4 + $3 - $2 - $e - $l`s
sox part1.au piece.au part2.au "$5" splice \
    `expr $4 + $e`s \
    `expr $4 + $e + $3 - $2 + $e + $l + $e`s
```

In the above Bourne shell script, 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 an number of seconds, and *leeway* should be set to zero.

stat [**-s** *n*] [**-rms**] [**-freq**] [**-v**] [**-d**]

Do a statistical check on the input file, and print results on the standard error file. Audio is passed unmodified through the SoX processing chain.

The ‘Volume Adjustment:’ field in the statistics gives you the parameter to the **-v** *number* which will make the audio as loud as possible without clipping. Note: See the discussion on **Clipping** in **sox(1)** for reasons why it is rarely a good idea to actually do this.

The option **-v** will print out the ‘Volume Adjustment:’ field’s value only and return. This could be of use in scripts to auto convert the volume.

The **-s** option is used to scale the input data by a given factor. The default value of *n* is the maximum value of a signed long integer (7fffffff in hexadecimal). Internal effects always work with signed long PCM data and so the value should relate to this fact.

The **-rms** option will convert all output average values to ‘root mean square’ format.

The **-freq** option calculates the input’s power spectrum and prints it to standard error.

There is also an optional parameter **-d** that will print out a hex dump of the audio from the internal buffer that is in 32-bit signed PCM data. This is mainly only of use in tracking down endian problems that creep in to SoX on cross-platform versions.

swap [*l* *2* | *1* *2* *3* *4*]

Swap channels in multi-channel audio files. Optionally, you may specify the channel order you would like the output in. This defaults to output channel 2 and then 1 for stereo and 2, 1, 4, 3 for quad-channels. An interesting feature is that you may duplicate a given channel by overwriting another. This is done by repeating an output channel on the command-line. For example, **swap 2 2** will overwrite channel 1 with channel 2; creating a stereo file with both channels containing the same audio.

See also the **remix** effect.

synth [*len*] {[*type*] [*combine*] [[*%*]*freq*[*k*]][:|+|/|-*%*]*freq2*[*k*]] [*off*] [*ph*] [*p1*] [*p2*] [*p3*]}

This effect can be used to generate fixed or swept frequency audio tones with various wave shapes, or to generate wide-band noise of various ‘colours’. Multiple synth effects can be cascaded to produce more complex waveforms; at each stage it is possible to choose whether the generated waveform will be mixed with, or modulated onto the output from the previous stage. Audio for each channel in a multi-channel audio file can be synthesised independently.

Though this effect is used to generate audio, an input file must still be given, the characteristics of which will be used to set the synthesised 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 name **-n**) is often given instead (and the length specified as a parameter to **synth** or by another given effect that can has an associated length).

For example, the following produces a 3 second, 48kHz, audio file containing a sine-wave swept from 300 to 3300 Hz:

```
sox -n output.au synth 3 sine 300-3300
```

and this produces an 8 kHz version:

```
sox -r 8000 -n output.au synth 3 sine 300-3300
```

Multiple channels can be synthesised by specifying the set of parameters shown between braces multiple times; the following puts the swept tone in the left channel and adds ‘brown’ noise in the right:

```
sox -n output.au synth 3 sine 300-3300 brownnoise
```

The following example shows how two synth effects can be cascaded to create a more complex waveform:

```
sox -n output.au synth 0.5 sine 200-500 \  
    synth 0.5 sine fmod 700-100
```

Frequencies can also be given as a number of musical semitones relative to ‘middle A’ (440 Hz) by prefixing a ‘%’ character; for example, the following could be used to help tune a guitar’s ‘E’ strings:

```
play -n synth sine %-17
```

N.B. This effect generates audio at maximum volume (0dBFS), which means that there is a high chance of clipping when using the audio subsequently, so in most cases, you will want to follow this effect with the **gain** effect to prevent this from happening. (See also **Clipping** in **sox(1)**.)

A detailed description of each **synth** parameter follows:

len is the length of audio to synthesise expressed as a time or as a number of samples; 0=input-length, default=0.

The format for specifying lengths in time is hh:mm:ss.frac. The format for specifying sample counts is the number of samples with the letter 's' appended to it.

type is one of sine, square, triangle, sawtooth, trapezium, exp, [white]noise, pinknoise, brown-noise; default=sine

combine is one of create, mix, amod (amplitude modulation), fmod (frequency modulation); default=create

freq/freq2 are the frequencies at the beginning/end of synthesis in Hz or, if preceded with '%', semitones relative to A (440 Hz); for both, default=%0. If *freq2* is given, then *len* must also have been given and the generated tone will be swept between the given frequencies. The two given frequencies must be separated by one of the characters ':', '+', '/', or '-'. This character is used to specify the sweep function as follows:

- : Linear: the tone will change by a fixed number of hertz per second.
- + Square: a second-order function is used to change the tone.
- / Exponential: the tone will change by a fixed number of semitones per second.
- Exponential: as '/', but initial phase always zero, and stepped (less smooth) frequency changes.

Not used for noise.

off is the bias (DC-offset) of the signal in percent; default=0.

ph is the phase shift in percentage of 1 cycle; default=0. Not used for noise.

p1 is the percentage of each cycle that is 'on' (square), or 'rising' (triangle, exp, trapezium); default=50 (square, triangle, exp), default=10 (trapezium).

p2 (trapezium): the percentage through each cycle at which 'falling' begins; default=50. exp: the amplitude in percent; default=100.

p3 (trapezium): the percentage through each cycle at which 'falling' ends; default=60.

tempo [-q] *factor* [*segment* [*search* [*overlap*]]]

Change the audio tempo (but not its pitch) using a 'WSOLA' 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 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, giving a quicker, but possibly lower quality, result.

factor gives the ratio of new tempo to the old tempo, so e.g. 1.1 speeds up the tempo by 10%, and 0.9 slows it down by 10%.

The optional *segment* parameter selects the algorithm's segment size in milliseconds. The default value is 82 and is typically suited to making small changes to the tempo of music; for larger changes (e.g. a factor of 2), 50 ms may give a better result. When changing the tempo of speech, a segment size of around 30 ms often works well.

The optional *search* parameter gives the audio length in milliseconds (default 14) over which the algorithm will search for overlapping points. Larger values use more processing time and do not necessarily produce better results.

The optional *overlap* parameter gives the segment overlap length in milliseconds (default 12).

See also **stretch** for a similar effect, **speed** for an effect that changes tempo and key together, and **key** for an effect that changes key without changing tempo.

treble *gain* [*frequency*[**k**] [*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 amplitude modulation) effect to the audio. The tremolo frequency in Hz is given by *speed*, and the depth as a percentage by *depth* (default 40).

Note: This effect is a special case of the **synth** effect.

trim *start* [*length*]

Trim can trim off unwanted audio from the beginning and end of the audio. Audio is not sent to the output stream until the *start* location is reached.

The optional *length* parameter tells the number of samples to output after the *start* sample and is used to trim off the back side of the audio. Using a value of 0 for the *start* parameter will allow trimming off the back side only.

Both options can be specified using either an amount of time or an exact count of samples. The format for specifying lengths in time is hh:mm:ss.frac. A start value of 1:30.5 will not start until 1 minute, thirty and ½ seconds into the audio. The format for specifying sample counts is the number of samples with the letter ‘s’ appended to it. A value of 8000s will wait until 8000 samples are read before starting to process audio.

vol *gain* [*type* [*limitergain*]]

Apply an amplification or an attenuation to the audio signal. Unlike the **-v** option (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, and several times if necessary, during the processing chain.

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), then *gain* is an amplitude (i.e. voltage or linear) ratio, if **power**, then a power (i.e. wattage or voltage-squared) ratio, and if **dB**, then 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 *limitergain* 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 will cause no limiter to be used. In verbose mode, this effect will display the percentage of the audio that needed to be limited.

See also **compand** for a dynamic-range compression/expansion/limiting effect.

Deprecated Effects

The following effects have been renamed or have their functionality included in another effect; they continue to work in this version of SoX but may be removed in future.

pitch *shift* [*width* *interpolate* *fade*]

Change the audio pitch (but not its duration). This effect is equivalent to the **key** effect with *search* set to zero, so its results are comparatively poor; it is retained for backwards compatibility only.

Change by cross-fading shifted samples. *shift* is given in cents. Use a positive value to shift to treble, negative value to shift to bass. Default shift is 0. *width* of window is in ms. Default width is 20ms. Try 30ms to lower pitch, and 10ms to raise pitch. *interpolate* option, can be **cubic** or **linear**. Default is **cubic**. The *fade* option, can be **cos**, **hamming**, **linear** or **trapezoid**; the default is **cos**.

polyphase [-w **nut**|**ham**] [-width *n*] [-cut-off *c*]

Change the sampling rate using ‘polyphase interpolation’, a DSP algorithm. **polyphase** copes with only certain rational fraction resampling ratios, and, compared with the **rate** effect, is generally slow, memory intensive, and has poorer stop-band rejection.

If the -w parameter is **nut**, then a Nuttall (~90 dB stop-band) window will be used; **ham** selects a Hamming (~43 dB stop-band) window. The default is Nuttall.

The -width parameter specifies the (approximate) width of the filter. The default is 1024 samples, which produces reasonable results.

The -cut-off value (*c*) specifies the filter cut-off frequency in terms of fraction of frequency bandwidth, also know as the Nyquist frequency. See the **resample** effect for further information on Nyquist frequency. If up-sampling, then this is the fraction of the original signal that should go through. If down-sampling, this is the fraction of the signal left after down-sampling. The default is 0.95.

See also **rate**, **rabbit** and **resample** for other sample-rate changing effects.

rabbit [-c0|-c1|-c2|-c3|-c4]

Change the sampling rate using libsamplerate, also known as ‘Secret Rabbit Code’. This effect is optional and, due to licence issues, is not included in all versions of SoX. Compared with the **rate** effect, **rabbit** is very slow.

See <http://www.mega-nerd.com/SRC> for details of the algorithms. Algorithms 0 through 2 are progressively faster and lower quality versions of the sinc algorithm; the default is -c0. Algorithm 3 is zero-order hold, and 4 is linear interpolation.

See also **rate**, **polyphase** and **resample** for other sample-rate changing effects, and see **resample** for more discussion of resampling.

resample [-qs|-q|-ql] [rolloff [*beta*]]

Change the sampling rate using simulated analog filtration. Compared with the **rate** effect, **resample** is slow, and has poorer stop-band rejection. Only the low quality option works with all resampling ratios.

By default, linear interpolation of the filter coefficients is used, with a window width about 45 samples at the lower of the two rates. This gives an accuracy of about 16 bits, but insufficient stop-band rejection in the case that you want to have roll-off greater than about 0.8 of the Nyquist frequency.

The -q* options will change the default values for roll-off and beta as well as use quadratic interpolation of filter coefficients, resulting in about 24 bits precision. The -qs, -q, or -ql options specify increased accuracy at the cost of lower execution speed. It is optional to specify roll-off and beta parameters when using the -q* options.

Following is a table of the reasonable defaults which are built-in to SoX:

Option	Window	Roll-off	Beta	Interpolation
(none)	45	0.80	16	linear
-qs	45	0.80	16	quadratic
-q	75	0.875	16	quadratic
-ql	149	0.94	16	quadratic

-qs, -q, or -ql use window lengths of 45, 75, or 149 samples, respectively, at the lower sample-

rate of the two files. This means progressively sharper stop-band rejection, at proportionally slower execution times.

rolloff refers to the cut-off frequency of the low pass filter and is given in terms of the Nyquist frequency for the lower sample rate. *rolloff* therefore should be something between 0 and 1, in practise 0.8–0.95. The defaults are indicated above.

The *Nyquist frequency* is equal to half the sample rate. Logically, this is because the A/D converter needs at least 2 samples to detect 1 cycle at the Nyquist frequency. Frequencies higher than the Nyquist will actually appear as lower frequencies to the A/D converter and is called aliasing. Normally, A/D converts run the signal through a lowpass filter first to avoid these problems.

Similar problems will happen in software when reducing the sample rate of an audio file (frequencies above the new Nyquist frequency can be aliased to lower frequencies). Therefore, a good resample effect will remove all frequency information above the new Nyquist frequency.

The *rolloff* refers to how close to the Nyquist frequency this cut-off is, with closer being better. When increasing the sample rate of an audio file you would not expect to have any frequencies exist that are past the original Nyquist frequency. Because of resampling properties, it is common to have aliasing artifacts created above the old Nyquist frequency. In that case the *rolloff* refers to how close to the original Nyquist frequency to use a highpass filter to remove these artifacts, with closer also being better.

The *beta*, if unspecified, defaults to 16. This selects a Kaiser window. You can select a Nuttall window by specifying anything ≤ 2 here. For more discussion of *beta*, look under the **filter** effect.

Default parameters are, as indicated above, Kaiser window of length 45, roll-off 0.80, *beta* 16, linear interpolation.

Note: **-qs** is only slightly slower, but more accurate for 16-bit or higher precision.

Note: In many cases of up-sampling, no interpolation is needed, as exact filter coefficients can be computed in a reasonable amount of space. To be precise, this is done when both $\text{input-rate} < \text{output-rate}$, and $\text{output-rate} \div \text{gcd}(\text{input-rate}, \text{output-rate}) \leq 511$.

See also **rate**, **polyphase** and **rabbit** for other sample-rate changing effects. There is a detailed analysis of **resample** and **polyphase** at <http://leute.server.de/wilde/resample.html>; see **rabbit** for a pointer to its own documentation.

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

Change the audio duration (but not its pitch). This effect is equivalent to the **tempo** effect with (*factor* inverted and) *search* set to zero, so its results are comparatively poor; it is retained for backwards compatibility only.

factor of stretching: >1 lengthen, <1 shorten duration. *window* size is in ms. Default is 20ms. The *fade* option, can be 'lin'. *shift* ratio, in $[0\ 1]$. Default depends on stretch factor. 1 to shorten, 0.8 to lengthen. The *fading* ratio, in $[0\ 0.5]$. The amount of a fade's default depends on *factor* and *shift*.

SEE ALSO

sox(1), **soxi**(1), **soxformat**(7), **libsox**(3),

The SoX web page at <http://sox.sourceforge.net>

SoX scripting examples at <http://sox.sourceforge.net/Docs/Scripts>

References

- [1] R. Bristow-Johnson, *Cookbook formulae for audio EQ biquad filter coefficients*, <http://musicdsp.org/files/Audio-EQ-Cookbook.txt>
- [2] Wikipedia, *Q-factor*, http://en.wikipedia.org/wiki/Q_factor
- [3] Scott Lehman, *Effects Explained*, <http://harmony-central.com/Effects/effects-explained.html>

- [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>
- [7] Steve Harris, *LADSPA plugins*, <http://plugin.org.uk>

AUTHORS

Chris Bagwell (cbagwell@users.sourceforge.net). Other authors and contributors are listed in the AUTHORS file that is distributed with the source code.