youtube-dl - Man Page

download videos from youtube.com or other video platforms

Examples (TL;DR)

- Download a video or playlist:youtube-dl https://www.youtube.com/watch?v=oHg5SJYRHA0

- List all formats that a video or playlist is available in:youtube-dl --list-formats https://www.youtube.com/watch?v=Mwa0_nE9H7A

- Download a video or playlist at a specific quality:youtube-dl --format "best[height<=480]" https://www.youtube.com/watch?v=oHg5SJYRHA0

- Download the audio from a video and convert it to an MP3:youtube-dl -x --audio-format mp3 url

- Download the best quality audio and video and merge them:youtube-dl -f bestvideo+bestaudio url

- Download video(s) as MP4 files with custom filenames:youtube-dl --format mp4 -o "%(title)s by %(uploader)s on %(upload_date)s in %(playlist)s.%(ext)s" url

- List available subtitles:youtube-dl --list-subs https://www.youtube.com/watch?v=Mwa0_nE9H7A

- Download a particular language's subtitles along with the video:youtube-dl --sub-lang en --write-sub https://www.youtube.com/watch?v=Mwa0_nE9H7A

Synopsis

youtube-dl [Options] URL [URL...]

Description

youtube-dl is a command-line program to download videos from YouTube.com and a few more sites. It requires the Python interpreter, version 2.6, 2.7, or 3.2+, and it is not platform specific. It should work on your Unix box, on Windows or on macOS. It is released to the public domain, which means you can modify it, redistribute it or use it however you like.

Options

-h, --help

      Print this help text and exit

--version

      Print program version and exit

-U, --update

      Update this program to latest version. Make sure that you have sufficient permissions (run with sudo if needed)

-i, --ignore-errors

Continue on download errors, for example to skip unavailable videos in a playlist

--abort-on-error

Abort downloading of further videos (in the playlist or the command line) if an error occurs

--dump-user-agent

Display the current browser identification

--list-extractors

List all supported extractors

--extractor-descriptions

Output descriptions of all supported extractors

--force-generic-extractor

Force extraction to use the generic extractor

--default-search *PREFIX*

Use this prefix for unqualified URLs. For example "gvsearch2:" downloads two videos from google videos for youtube-dl "large apple". Use the value "auto" to let youtube-dl guess ("auto_warning" to emit a warning when guessing). "error" just throws an error. The default value "fixup_error" repairs broken URLs, but emits an error if this is not possible instead of searching.

--ignore-config

Do not read configuration files. When given in the global configuration file /etc/youtube-dl.conf: Do not read the user configuration in ~/.config/youtube- dl/config (%APPDATA%/youtube-dl/config.txt on Windows)

--config-location *PATH*

Location of the configuration file; either the path to the config or its containing directory.

--flat-playlist

Do not extract the videos of a playlist, only list them.

--mark-watched

Mark videos watched (YouTube only)

--no-mark-watched

Do not mark videos watched (YouTube only)

--no-color

Do not emit color codes in output

Network Options

--proxy *URL*

Use the specified HTTP/HTTPS/SOCKS proxy. To enable SOCKS proxy, specify a proper scheme. For example socks5://127.0.0.1:1080/. Pass in an empty string (--proxy "") for direct connection

--socket-timeout *SECONDS*

Time to wait before giving up, in seconds

--source-address *IP*

Client-side IP address to bind to

-4, **--force-ipv4**

Make all connections via IPv4

-6, **--force-ipv6**

Make all connections via IPv6

Geo Restriction

--geo-verification-proxy *URL*

Use this proxy to verify the IP address for some geo-restricted sites. The default proxy specified by --proxy (or none, if the option is not present) is used for the actual downloading.

--geo-bypass

Bypass geographic restriction via faking X-Forwarded-For HTTP header

--no-geo-bypass

Do not bypass geographic restriction via faking X-Forwarded-For HTTP header

--geo-bypass-country *CODE*

Force bypass geographic restriction with explicitly provided two-letter ISO 3166-2 country code

--geo-bypass-ip-block *IP_BLOCK*

Force bypass geographic restriction with explicitly provided IP block in CIDR notation

Video Selection

--playlist-start *NUMBER*

Playlist video to start at (default is 1)

--playlist-end *NUMBER*

Playlist video to end at (default is last)

--playlist-items *ITEM_SPEC*

Playlist video items to download. Specify indices of the videos in the playlist separated by commas like: "**--playlist-items** 1,2,5,8" if you want to download videos indexed 1, 2, 5, 8 in the playlist. You can specify range: "**--playlist-items** 1-3,7,10-13", it will download the videos at index 1, 2, 3, 7, 10, 11, 12 and 13.

--match-title *REGEX*

Download only matching titles (regex or caseless sub-string)

--reject-title *REGEX*

Skip download for matching titles (regex or caseless sub-string)

--max-downloads *NUMBER*

Abort after downloading NUMBER files

--min-filesize *SIZE*

Do not download any videos smaller than SIZE (e.g. 50k or 44.6m)

--max-filesize *SIZE*

Do not download any videos larger than SIZE (e.g. 50k or 44.6m)

--date *DATE*

Download only videos uploaded in this date

--datebefore *DATE*

Download only videos uploaded on or before this date (i.e. inclusive)

--dateafter *DATE*

Download only videos uploaded on or after this date (i.e. inclusive)

--min-views *COUNT*

Do not download any videos with less than COUNT views

--max-views COUNT

Do not download any videos with more than COUNT views

--match-filter FILTER

Generic video filter. Specify any key (see the "Output Template" for a list of available keys) to match if the key is present, !key to check if the key is not present, key > NUMBER (like "comment_count > 12", also works with >=, <, <=, !=, =) to compare against a number, key = 'LITERAL' (like "uploader = 'Mike Smith'", also works with !=) to match against a string literal and & to require multiple matches. Values which are not known are excluded unless you put a question mark (?) after the operator. For example, to only match videos that have been liked more than 100 times and disliked less than 50 times (or the dislike functionality is not available at the given service), but who also have a description, use --match-filter "like_count > 100 & dislike_count <? 50 & description" .

--no-playlist

Download only the video, if the URL refers to a video and a playlist.

--yes-playlist

Download the playlist, if the URL refers to a video and a playlist.

--age-limit YEARS

Download only videos suitable for the given age

--download-archive FILE

Download only videos not listed in the archive file. Record the IDs of all downloaded videos in it.

--include-ads

Download advertisements as well (experimental)

Download Options

-r, **--limit-rate** RATE

    Maximum download rate in bytes per second (e.g. 50K or 4.2M)

-R, **--retries** RETRIES

    Number of retries (default is 10), or "infinite".

--fragment-retries RETRIES

    Number of retries for a fragment (default is 10), or "infinite" (DASH, hlsnative and ISM)

--skip-unavailable-fragments

    Skip unavailable fragments (DASH, hlsnative and ISM)

--abort-on-unavailable-fragment

    Abort downloading when some fragment is not available

--keep-fragments

Keep downloaded fragments on disk after downloading is finished; fragments are erased by default

--buffer-size *SIZE*

Size of download buffer (e.g. 1024 or 16K) (default is 1024)

--no-resize-buffer

Do not automatically adjust the buffer size. By default, the buffer size is automatically resized from an initial value of SIZE.

--http-chunk-size *SIZE*

Size of a chunk for chunk-based HTTP downloading (e.g. 10485760 or 10M) (default is disabled). May be useful for bypassing bandwidth throttling imposed by a webserver (experimental)

--playlist-reverse

Download playlist videos in reverse order

--playlist-random

Download playlist videos in random order

--xattr-set-filesize

Set file xattribute ytdl.filesize with expected file size

--hls-prefer-native

Use the native HLS downloader instead of ffmpeg

--hls-prefer-ffmpeg

Use ffmpeg instead of the native HLS downloader

--hls-use-mpegts

Use the mpegts container for HLS videos, allowing to play the video while downloading (some players may not be able to play it)

--external-downloader *COMMAND*

Use the specified external downloader. Currently supports aria2c,avconv,axel,curl,ffmpeg,httpie,wget

--external-downloader-args *ARGS*

Give these arguments to the external downloader

Filesystem Options

-a, --batch-file *FILE*

File containing URLs to download ('-' for stdin), one URL per line. Lines starting with '#', ';' or ']' are considered as comments and ignored.

--id

Use only video ID in file name

-o, **--output** *TEMPLATE*

Output filename template, see the "Output Template" for all the info

--autonumber-start *NUMBER*

Specify the start value for %(autonumber)s (default is 1)

--restrict-filenames

Restrict filenames to only ASCII characters, and avoid "&" and spaces in filenames

-w, **--no-overwrites**

Do not overwrite files

-c, **--continue**

Force resume of partially downloaded files. By default, youtube-dl will resume downloads if possible.

--no-continue

Do not resume partially downloaded files (restart from beginning)

--no-part

Do not use .part files - write directly into output file

--no-mtime

Do not use the Last-modified header to set the file modification time

--write-description

Write video description to a .description file

--write-info-json

Write video metadata to a .info.json file

--write-annotations

Write video annotations to a .annotations.xml file

--load-info-json *FILE*

JSON file containing the video information (created with the "**--write-info-json**" option)

--cookies *FILE*

File to read cookies from and dump cookie jar in

--cache-dir *DIR*

Location in the filesystem where youtube-dl can store some downloaded information permanently. By default $XDG_CACHE_HOME/youtube-dl or ~/.cache/youtube-dl . At the moment, only YouTube player files (for videos with obfuscated signatures) are cached, but that may change.

--no-cache-dir

Disable filesystem caching

--rm-cache-dir

      Delete all filesystem cache files

Thumbnail images

--write-thumbnail

      Write thumbnail image to disk

--write-all-thumbnails

      Write all thumbnail image formats to disk

--list-thumbnails

      Simulate and list all available thumbnail formats

Verbosity / Simulation Options

-q, **--quiet**

      Activate quiet mode

--no-warnings

      Ignore warnings

-s, **--simulate**

      Do not download the video and do not write anything to disk

--skip-download

      Do not download the video

-g, **--get-url**

      Simulate, quiet but print URL

-e, **--get-title**

      Simulate, quiet but print title

--get-id

      Simulate, quiet but print id

--get-thumbnail

      Simulate, quiet but print thumbnail URL

--get-description

      Simulate, quiet but print video description

--get-duration

      Simulate, quiet but print video length

--get-filename

      Simulate, quiet but print output filename

--get-format

Simulate, quiet but print output format

-j, --dump-json

Simulate, quiet but print JSON information. See the "[Output Template](#)" for a description of available keys.

-J, --dump-single-json

Simulate, quiet but print JSON information for each command-line argument. If the URL refers to a playlist, dump the whole playlist information in a single line.

--print-json

Be quiet and print the video information as JSON (video is still being downloaded).

--newline

Output progress bar as new lines

--no-progress

Do not print progress bar

--console-title

Display progress in console titlebar

-v, --verbose

Print various debugging information

--dump-pages

Print downloaded pages encoded using base64 to debug problems (very verbose)

--write-pages

Write downloaded intermediary pages to files in the current directory to debug problems

--print-traffic

Display sent and read HTTP traffic

-C, --call-home

Contact the youtube-dl server for debugging

--no-call-home

Do NOT contact the youtube-dl server for debugging

Workarounds

--encoding *ENCODING*

Force the specified encoding (experimental)

--no-check-certificate

Suppress HTTPS certificate validation

--prefer-insecure

Use an unencrypted connection to retrieve information about the video. (Currently supported only for YouTube)

--user-agent *UA*

Specify a custom user agent

--referer *URL*

Specify a custom referer, use if the video access is restricted to one domain

--add-header *FIELD:VALUE*

Specify a custom HTTP header and its value, separated by a colon ':'. You can use this option multiple times

--bidi-workaround

Work around terminals that lack bidirectional text support. Requires bidiv or fribidi executable in PATH

--sleep-interval *SECONDS*

Number of seconds to sleep before each download when used alone or a lower bound of a range for randomized sleep before each download (minimum possible number of seconds to sleep) when used along with --max-sleep-interval.

--max-sleep-interval *SECONDS*

Upper bound of a range for randomized sleep before each download (maximum possible number of seconds to sleep). Must only be used along with --min-sleep-interval.

Video Format Options

-f, **--format** *FORMAT*

Video format code, see the "Format Selection" for all the info

--all-formats

Download all available video formats

--prefer-free-formats

Prefer free video formats unless a specific one is requested

-F, **--list-formats**

List all available formats of requested videos

--youtube-skip-dash-manifest

Do not download the DASH manifests and related data on YouTube videos

--merge-output-format *FORMAT*

If a merge is required (e.g. bestvideo+bestaudio), output to given container format. One of mkv, mp4, ogg, webm, flv. Ignored if no merge is required

Subtitle Options

--write-sub

      Write subtitle file

--write-auto-sub

      Write automatically generated subtitle file (YouTube only)

--all-subs

      Download all the available subtitles of the video

--list-subs

      List all available subtitles for the video

--sub-format *FORMAT*

      Subtitle format, accepts formats preference, for example: "srt" or "ass/srt/best"

--sub-lang *LANGS*

      Languages of the subtitles to download (optional) separated by commas, use --list- subs for available language tags

Authentication Options

-u, **--username** *USERNAME*

      Login with this account ID

-p, **--password** *PASSWORD*

      Account password. If this option is left out, youtube-dl will ask interactively.

-2, **--twofactor** *TWOFACTOR*

      Two-factor authentication code

-n, **--netrc**

      Use .netrc authentication data

--video-password *PASSWORD*

      Video password (vimeo, smotri, youku)

Adobe Pass Options

--ap-mso *MSO*

      Adobe Pass multiple-system operator (TV provider) identifier, use **--ap-list-mso** for a list of available MSOs

--ap-username *USERNAME*

      Multiple-system operator account login

--ap-password *PASSWORD*

Multiple-system operator account password. If this option is left out, youtube-dl will ask interactively.

--ap-list-mso

List all supported multiple-system operators

Post-processing Options

-x, **--extract-audio**

Convert video files to audio-only files (requires ffmpeg or avconv and ffprobe or avprobe)

--audio-format *FORMAT*

Specify audio format: "best", "aac", "flac", "mp3", "m4a", "opus", "vorbis", or "wav";

"best" by default; No effect without **-x**

--audio-quality *QUALITY*

Specify ffmpeg/avconv audio quality, insert a value between 0 (better) and 9 (worse) for VBR or a specific bitrate like 128K (default 5)

--recode-video *FORMAT*

Encode the video to another format if necessary (currently supported: mp4|flv|ogg|webm|mkv|avi)

--postprocessor-args *ARGS*

Give these arguments to the postprocessor

-k, **--keep-video**

Keep the video file on disk after the post- processing; the video is erased by default

--no-post-overwrites

Do not overwrite post-processed files; the post-processed files are overwritten by default

--embed-subs

Embed subtitles in the video (only for mp4, webm and mkv videos)

--embed-thumbnail

Embed thumbnail in the audio as cover art

--add-metadata

Write metadata to the video file

--metadata-from-title *FORMAT*

Parse additional metadata like song title / artist from the video title. The format syntax is the same as **--output**. Regular expression with named capture groups may also be used. The parsed parameters replace existing values. Example: --metadata-from- title

"%(artist)s - %(title)s" matches a title like "Coldplay - Paradise". Example (regex): --metadata-from-title "(?P.+?) - (?P

.+)"

--xattrs

Write metadata to the video file's xattrs (using dublin core and xdg standards)

--fixup *POLICY*

Automatically correct known faults of the file. One of never (do nothing), warn (only emit a warning), detect_or_warn (the default; fix file if we can, warn otherwise)

--prefer-avconv

Prefer avconv over ffmpeg for running the postprocessors

--prefer-ffmpeg

Prefer ffmpeg over avconv for running the postprocessors (default)

--ffmpeg-location *PATH*

Location of the ffmpeg/avconv binary; either the path to the binary or its containing directory.

--exec *CMD*

Execute a command on the file after downloading, similar to find's -exec syntax.

Example: --exec 'adb push {} /sdcard/Music/ && rm {}'

--convert-subs *FORMAT*

Convert the subtitles to other format (currently supported: srt|ass|vtt|lrc)

Configuration

You can configure youtube-dl by placing any supported command line option to a configuration file. On Linux and macOS, the system wide configuration file is located at /etc/youtube-dl.conf and the user wide configuration file at ~/.config/youtube-dl/config. On Windows, the user wide configuration file locations are %APPDATA%\youtube-dl\config.txt or C:\Users\<user name>\youtube-dl.conf. Note that by default configuration file may not exist so you may need to create it yourself.
For example, with the following configuration file youtube-dl will always extract the audio, not copy the mtime, use a proxy and save all videos under Movies directory in your home directory:

Lines starting with # are comments

Always extract audio

-x

Do not copy the mtime

--no-mtime

Use this proxy

--proxy 127.0.0.1:3128

Save all videos under Movies directory in your home directory

-o ~/Movies/%(title)s.%(ext)s

Note that options in configuration file are just the same options aka switches used in regular command line calls thus there **must be no whitespace** after - or --, e.g. -o or --proxy but

not - o or -- proxy.

You can use --ignore-config if you want to disable the configuration file for a particular youtube-dl run.

You can also use --config-location if you want to use custom configuration file for a particular youtube-dl run.

Authentication with .netrc file

You may also want to configure automatic credentials storage for extractors that support authentication (by providing login and password with --username and --password) in order not to pass credentials as command line arguments on every youtube-dl execution and prevent tracking plain text passwords in the shell command history. You can achieve this using a .netrc file (https://stackoverflow.com/tags/.netrc/info) on a per extractor basis. For that you will need to create a .netrc file in your $HOME and restrict permissions to read/write by only you:

touch $HOME/.netrc

chmod a-rwx,u+rw $HOME/.netrc

After that you can add credentials for an extractor in the following format, where *extractor* is the name of the extractor in lowercase:

machine <extractor> login <login> password <password>

For example:

machine youtube login myaccount@gmail.com password my_youtube_password

machine twitch login my_twitch_account_name password my_twitch_password

To activate authentication with the .netrc file you should pass --netrc to youtube-dl or place it in the configuration file.

On Windows you may also need to setup the %HOME% environment variable manually. For example:

set HOME=%USERPROFILE%

Output Template

The -o option allows users to indicate a template for the output file names.
tl;dr: navigate me to examples.
The basic usage is not to set any template arguments when downloading a single file, like in youtube-dl -o funny_video.flv "https://some/video". However, it may contain special sequences that will be replaced when downloading each video. The special sequences may be formatted according to python string formatting operations (https://docs.python.org/2/library/stdtypes.html#string-formatting). For example, %(NAME)s or %(NAME)05d. To clarify, that is a percent symbol followed by a name in parentheses, followed by formatting operations. Allowed names along with sequence type are:

- id (string): Video identifier
- title (string): Video title
- url (string): Video URL
- ext (string): Video filename extension
- alt_title (string): A secondary title of the video
- display_id (string): An alternative identifier for the video
- uploader (string): Full name of the video uploader
- license (string): License name the video is licensed under
- creator (string): The creator of the video
- release_date (string): The date (YYYYMMDD) when the video was released
- timestamp (numeric): UNIX timestamp of the moment the video became available
- upload_date (string): Video upload date (YYYYMMDD)
- uploader_id (string): Nickname or id of the video uploader
- channel (string): Full name of the channel the video is uploaded on
- channel_id (string): Id of the channel
- location (string): Physical location where the video was filmed
- duration (numeric): Length of the video in seconds
- view_count (numeric): How many users have watched the video on the platform
- like_count (numeric): Number of positive ratings of the video
- dislike_count (numeric): Number of negative ratings of the video
- repost_count (numeric): Number of reposts of the video

- average_rating (numeric): Average rating give by users, the scale used depends on the webpage
- comment_count (numeric): Number of comments on the video
- age_limit (numeric): Age restriction for the video (years)
- is_live (boolean): Whether this video is a live stream or a fixed-length video
- start_time (numeric): Time in seconds where the reproduction should start, as specified in the URL
- end_time (numeric): Time in seconds where the reproduction should end, as specified in the URL
- format (string): A human-readable description of the format
- format_id (string): Format code specified by --format
- format_note (string): Additional info about the format
- width (numeric): Width of the video
- height (numeric): Height of the video
- resolution (string): Textual description of width and height
- tbr (numeric): Average bitrate of audio and video in KBit/s
- abr (numeric): Average audio bitrate in KBit/s
- acodec (string): Name of the audio codec in use
- asr (numeric): Audio sampling rate in Hertz
- vbr (numeric): Average video bitrate in KBit/s
- fps (numeric): Frame rate
- vcodec (string): Name of the video codec in use
- container (string): Name of the container format
- filesize (numeric): The number of bytes, if known in advance
- filesize_approx (numeric): An estimate for the number of bytes
- protocol (string): The protocol that will be used for the actual download
- extractor (string): Name of the extractor
- extractor_key (string): Key name of the extractor
- epoch (numeric): Unix epoch when creating the file
- autonumber (numeric): Five-digit number that will be increased with each download, starting at zero
- playlist (string): Name or id of the playlist that contains the video
- playlist_index (numeric): Index of the video in the playlist padded with leading zeros according to the total length of the playlist
- playlist_id (string): Playlist identifier
- playlist_title (string): Playlist title

- playlist_uploader (string): Full name of the playlist uploader
- playlist_uploader_id (string): Nickname or id of the playlist uploader

Available for the video that belongs to some logical chapter or section:

- chapter (string): Name or title of the chapter the video belongs to
- chapter_number (numeric): Number of the chapter the video belongs to
- chapter_id (string): Id of the chapter the video belongs to

Available for the video that is an episode of some series or programme:

- series (string): Title of the series or programme the video episode belongs to
- season (string): Title of the season the video episode belongs to
- season_number (numeric): Number of the season the video episode belongs to
- season_id (string): Id of the season the video episode belongs to
- episode (string): Title of the video episode
- episode_number (numeric): Number of the video episode within a season
- episode_id (string): Id of the video episode

Available for the media that is a track or a part of a music album:

- track (string): Title of the track
- track_number (numeric): Number of the track within an album or a disc
- track_id (string): Id of the track
- artist (string): Artist(s) of the track
- genre (string): Genre(s) of the track
- album (string): Title of the album the track belongs to
- album_type (string): Type of the album
- album_artist (string): List of all artists appeared on the album
- disc_number (numeric): Number of the disc or other physical medium the track belongs to
- release_year (numeric): Year (YYYY) when the album was released

Each aforementioned sequence when referenced in an output template will be replaced by the actual value corresponding to the sequence name. Note that some of the sequences are not guaranteed to be present since they depend on the metadata obtained by a particular extractor. Such sequences will be replaced with NA.

For example for -o %(title)s-%(id)s.%(ext)s and an mp4 video with title youtube-dl test video and id BaW_jenozKcj, this will result in a youtube-dl test video-BaW_jenozKcj.mp4 file created in the current directory.

For numeric sequences you can use numeric related formatting, for example, %(view_count)05d will result in a string with view count padded with zeros up to 5 characters, like in 00042.

Output templates can also contain arbitrary hierarchical path, e.g. -o '%(playlist)s/%(playlist_index)s - %(title)s.%(ext)s' which will result in downloading each video in a directory corresponding to this path template. Any missing directory will be automatically created for you.

To use percent literals in an output template use %%. To output to stdout use -o -.

The current default template is %(title)s-%(id)s.%(ext)s.

In some cases, you don't want special characters such as 中, spaces, or &, such as when transferring the downloaded filename to a Windows system or the filename through an 8bit-unsafe channel. In these cases, add the --restrict-filenames flag to get a shorter title:

Output template and Windows batch files

If you are using an output template inside a Windows batch file then you must escape plain percent characters (%) by doubling, so that -o "%(title)s-%(id)s.%(ext)s" should become -o "%%(title)s-%%(id)s.%%(ext)s". However you should not touch %'s that are not plain characters, e.g. environment variables for expansion should stay intact: -o "C:\%HOMEPATH%\Desktop\%%(title)s.%%(ext)s".

Output template examples

Note that on Windows you may need to use double quotes instead of single.
$ youtube-dl --get-filename -o '%(title)s.%(ext)s' BaW_jenozKc

youtube-dl test video ''_ä↔⅄.mp4   # All kinds of weird characters

$ youtube-dl --get-filename -o '%(title)s.%(ext)s' BaW_jenozKc --restrict-filenames

youtube-dl_test_video_.mp4   # A simple file name

# Download YouTube playlist videos in separate directory indexed by video order in a playlist
$ youtube-dl -o '%(playlist)s/%(playlist_index)s - %(title)s.%(ext)s' https://www.youtube.com/playlist?list=PLwiyx1dc3P2JR9N8gQaQN_BCvlSlap7re

Download all playlists of YouTube channel/user keeping each playlist in separate directory:

$ youtube-dl -

o '%(uploader)s/%(playlist)s/%(playlist_index)s - %(title)s.%(ext)s' https://www.youtube.com/user/TheLinuxFoundation/playlists

Download Udemy course keeping each chapter in separate directory under MyVideos directory in your home

$ youtube-dl -u user -p password -

o '~/MyVideos/%(playlist)s/%(chapter_number)s - %(chapter)s/%(title)s.%(ext)s' https://www.udemy.com/java-tutorial/

Download entire series season keeping each series and each season in separate directory under C:/My Videos

$ youtube-dl -

o "C:/MyVideos/%(series)s/%(season_number)s - %(season)s/%(episode_number)s - %(episode)s.%(ext)s" https://videomore.ru/kino_v_detalayah/5_sezon/367617

Stream the video being downloaded to stdout

$ youtube-dl -o - BaW_jenozKc

Format Selection

By default youtube-dl tries to download the best available quality, i.e. if you want the best quality you **don't need** to pass any special options, youtube-dl will guess it for you by **default**. But sometimes you may want to download in a different format, for example when you are on a slow or intermittent connection. The key mechanism for achieving this is so-called *format selection* based on which you can explicitly specify desired format, select formats based on some criterion or criteria, setup precedence and much more.

The general syntax for format selection is --format FORMAT or shorter -

f FORMAT where FORMAT is a *selector expression*, i.e. an expression that describes format or formats you would like to download.

tl;dr: navigate me to examples.

The simplest case is requesting a specific format, for example with -f 22 you can download the format with format code equal to 22. You can get the list of available format codes for particular video using --list-formats or -F. Note that these format codes are extractor specific.

You can also use a file extension (currently 3gp, aac, flv, m4a, mp3, mp4, ogg, wav, webm are supported) to download the best quality format of a particular file extension served as a single file, e.g. -f webm will download the best quality format with the webm extension served as a single file.

You can also use special names to select particular edge case formats:

- best: Select the best quality format represented by a single file with video and audio.
- worst: Select the worst quality format represented by a single file with video and audio.
- bestvideo: Select the best quality video-only format (e.g. DASH video). May not be available.
- worstvideo: Select the worst quality video-only format. May not be available.
- bestaudio: Select the best quality audio only-format. May not be available.
- worstaudio: Select the worst quality audio only-format. May not be available.

For example, to download the worst quality video-only format you can use -f worstvideo.

If you want to download multiple videos and they don't have the same formats available, you can specify the order of preference using slashes. Note that slash is left-associative, i.e. formats on the left hand side are preferred, for example -f 22/17/18 will download format 22 if it's available, otherwise it will download format 17 if it's available, otherwise it will download format 18 if it's available, otherwise it will complain that no suitable formats are available for download.

If you want to download several formats of the same video use a comma as a separator, e.g. -f 22,17,18 will download all these three formats, of course if they are available. Or a more sophisticated example combined with the precedence feature: -f 136/137/mp4/bestvideo,140/m4a/bestaudio.

You can also filter the video formats by putting a condition in brackets, as in -f "best[height=720]" (or -f "[filesize>10M]").

The following numeric meta fields can be used with comparisons <, <=, >, >=, = (equals), != (not equals):

- filesize: The number of bytes, if known in advance
- width: Width of the video, if known
- height: Height of the video, if known
- tbr: Average bitrate of audio and video in KBit/s
- abr: Average audio bitrate in KBit/s
- vbr: Average video bitrate in KBit/s
- asr: Audio sampling rate in Hertz
- fps: Frame rate

Also filtering work for comparisons = (equals), ^= (starts with), $= (ends with), *= (contains) and following string meta fields:

- ext: File extension
- acodec: Name of the audio codec in use
- vcodec: Name of the video codec in use
- container: Name of the container format
- protocol: The protocol that will be used for the actual download, lower-case

  (http, https, rtsp, rtmp, rtmpe, mms, f4m, ism, http_dash_segments, m3u8, or m3u8_native)
- format_id: A short description of the format

Any string comparison may be prefixed with negation ! in order to produce an opposite comparison, e.g. !*= (does not contain).

Note that none of the aforementioned meta fields are guaranteed to be present since this solely depends on the metadata obtained by particular extractor, i.e. the metadata offered by the video hoster.

Formats for which the value is not known are excluded unless you put a question mark (?) after the operator. You can combine format filters, so -f "[height <=? 720][tbr>500]" selects up to 720p videos (or videos where the height is not known) with a bitrate of at least 500 KBit/s.

You can merge the video and audio of two formats into a single file using -f <video-format>+<audio-format> (requires ffmpeg or avconv installed), for example -f bestvideo+bestaudio will download the best video-only format, the best audio-only format and mux them together with ffmpeg/avconv.

Format selectors can also be grouped using parentheses, for example if you want to download the best mp4 and webm formats with a height lower than 480 you can use -f '(mp4,webm)[height<480]'.

Since the end of April 2015 and version 2015.04.26, youtube-dl uses -f bestvideo+bestaudio/best as the default format selection (see #5447 (https://github.com/ytdl-org/youtube-dl/issues/5447), #5456 (https://github.com/ytdl-org/youtube-dl/issues/5456)). If ffmpeg or avconv are installed this results in downloading bestvideo and bestaudio separately and muxing them together into a single file giving the best overall quality available. Otherwise it falls back to best and results in downloading the best available quality served as a single file. best is also needed for videos that don't come from YouTube because they don't provide the audio and video in two different files. If you want to only download some DASH formats (for example if you are not interested in getting videos with a resolution higher than 1080p), you can add -f bestvideo[height<=?1080]+bestaudio/best to your configuration file. Note that if you use youtube-dl to stream to stdout (and most likely to pipe it to your media player then), i.e. you explicitly specify output template as -o -, youtube-dl still uses -f best format selection in order to start

content delivery immediately to your player and not to wait until bestvideo and bestaudio are downloaded and muxed.

If you want to preserve the old format selection behavior (prior to youtube-dl 2015.04.26), i.e. you want to download the best available quality media served as a single file, you should explicitly specify your choice with -f best. You may want to add it to the configuration file in order not to type it every time you run youtube-dl.

Format selection examples

Note that on Windows you may need to use double quotes instead of single.

```
# Download best mp4 format available or any other best if no mp4 available
```

```
$ youtube-dl -f 'bestvideo[ext=mp4]+bestaudio[ext=m4a]/best[ext=mp4]/best'
```

```
# Download best format available but no better than 480p
```

```
$ youtube-dl -f 'bestvideo[height<=480]+bestaudio/best[height<=480]'
```

```
# Download best video only format but no bigger than 50 MB
```

```
$ youtube-dl -f 'best[filesize<50M]'
```

```
# Download best format available via direct link over HTTP/HTTPS protocol
```

```
$ youtube-dl -f '(bestvideo+bestaudio/best)[protocol^=http]'
```

```
# Download the best video format and the best audio format without merging them
```

```
$ youtube-dl -f 'bestvideo,bestaudio' -o '%(title)s.f%(format_id)s.%(ext)s'
```

Note that in the last example, an output template is recommended as bestvideo and bestaudio may have the same file name.

Video Selection

Videos can be filtered by their upload date using the options --date, --datebefore or --dateafter. They accept dates in two formats:

- Absolute dates: Dates in the format YYYYMMDD.
- Relative dates: Dates in the format (now|today)[+-][0-9](day|week|month|year)(s)?

Examples:

```
# Download only the videos uploaded in the last 6 months
```

$ youtube-dl --dateafter now-6months

# Download only the videos uploaded on January 1, 1970
$ youtube-dl --date 19700101

$ # Download only the videos uploaded in the 200x decade
$ youtube-dl --dateafter 20000101 --datebefore 20091231

FAQ

How do I update youtube-dl?

If you've followed our manual installation instructions (https://ytdl-org.github.io/youtube-dl/download.html), you can simply run youtube-dl -U (or, on Linux, sudo youtube-dl -U).

If you have used pip, a simple sudo pip install -U youtube-dl is sufficient to update.

If you have installed youtube-dl using a package manager like *apt-get* or *yum*, use the standard system update mechanism to update. Note that distribution packages are often outdated. As a rule of thumb, youtube-dl releases at least once a month, and often weekly or even daily. Simply go to https://yt-dl.org to find out the current version. Unfortunately, there is nothing we youtube-dl developers can do if your distribution serves a really outdated version. You can (and should) complain to your distribution in their bugtracker or support forum.
As a last resort, you can also uninstall the version installed by your package manager and follow our manual installation instructions. For that, remove the distribution's package, with a line like

sudo apt-get remove -y youtube-dl

Afterwards, simply follow our manual installation instructions (https://ytdl-org.github.io/youtube-dl/download.html):

sudo wget https://yt-dl.org/downloads/latest/youtube-dl -O /usr/local/bin/youtube-dl

sudo chmod a+rx /usr/local/bin/youtube-dl

hash -r

Again, from then on you'll be able to update with sudo youtube-dl -U.

youtube-dl is extremely slow to start on Windows

Add a file exclusion for youtube-dl.exe in Windows Defender settings.

I'm getting an error

Unable to extract OpenGraph title on YouTube playlists

YouTube changed their playlist format in March 2014 and later on, so you'll need at least youtube-dl 2014.07.25 to download all YouTube videos.
If you have installed youtube-dl with a package manager, pip, setup.py or a tarball, please use that to update. Note that Ubuntu packages do not seem to get updated anymore. Since we are not affiliated with Ubuntu, there is little we can do. Feel free to report bugs (https://bugs.launchpad.net/ubuntu/+source/youtube-dl/+filebug) to the Ubuntu packaging people (mailto:ubuntu-motu@lists.ubuntu.com?subject=outdated%20version%20of%20youtube-dl) - all they have to do is update the package to a somewhat recent version. See above for a way to update.

I'm getting an error when trying to use output template

error: using output template conflicts with using title, video ID or auto number

Make sure you are not using -o with any of these options -t, --title, --id, -A or --auto-number set in command line or in a configuration file. Remove the latter if any.

Do I always have to pass -citw?

By default, youtube-dl intends to have the best options (incidentally, if you have a convincing case that these should be different, please file an issue where you explain that (https://yt-dl.org/bug)). Therefore, it is unnecessary and sometimes harmful to copy long option strings from webpages. In particular, the only option out of -citw that is regularly useful is -i.

Can you please put the -b option back?

Most people asking this question are not aware that youtube-dl now defaults to downloading the highest available quality as reported by YouTube, which will be 1080p or 720p in some cases, so you no longer need the -b option. For some specific videos, maybe YouTube does not report them to be available in a specific high quality format you're interested in. In that case, simply request it with the -f option and youtube-dl will try to download it.

I get HTTP error 402 when trying to download a video. What's

this?
Apparently YouTube requires you to pass a CAPTCHA test if you download too much. We're considering to provide a way to let you solve the CAPTCHA (https://github.com/ytdl-org/youtube-dl/issues/154), but at the moment, your best course of action is pointing a web browser to the youtube URL, solving the CAPTCHA, and restart youtube-dl.

Do I need any other programs?

youtube-dl works fine on its own on most sites. However, if you want to convert video/audio, you'll need avconv (https://libav.org/) or ffmpeg (https://www.ffmpeg.org/). On some sites -

most notably YouTube - videos can be retrieved in a higher quality format without sound. youtube-dl will detect whether avconv/ffmpeg is present and automatically pick the best option. Videos or video formats streamed via RTMP protocol can only be downloaded when rtmpdump (https://rtmpdump.mplayerhq.hu/) is installed. Downloading MMS and RTSP videos requires either mplayer (https://mplayerhq.hu/) or mpv (https://mpv.io/) to be installed.

I have downloaded a video but how can I play it?

Once the video is fully downloaded, use any video player, such as mpv (https://mpv.io/), vlc (https://www.videolan.org/) or mplayer (https://www.mplayerhq.hu/).

I extracted a video URL with -g, but it does not play on

another machine / in my web browser.
It depends a lot on the service. In many cases, requests for the video (to download/play it) must come from the same IP address and with the same cookies and/or HTTP headers. Use the --cookies option to write the required cookies into a file, and advise your downloader to read cookies from that file. Some sites also require a common user agent to be used, use --dump-user-agent to see the one in use by youtube-dl. You can also get necessary cookies and HTTP headers from JSON output obtained with --dump-json.
It may be beneficial to use IPv6; in some cases, the restrictions are only applied to IPv4. Some services (sometimes only for a subset of videos) do not restrict the video URL by IP address, cookie, or user-agent, but these are the exception rather than the rule.
Please bear in mind that some URL protocols are **not** supported by browsers out of the box, including RTMP. If you are using -g, your own downloader must support these as well.
If you want to play the video on a machine that is not running youtube-dl, you can relay the video content from the machine that runs youtube-dl. You can use -o - to let youtube-dl stream a video to stdout, or simply allow the player to download the files written by youtube-dl in turn.

ERROR: no fmt_url_map or conn information found in video info

YouTube has switched to a new video info format in July 2011 which is not supported by old versions of youtube-dl. See above for how to update youtube-dl.

ERROR: unable to download video

YouTube requires an additional signature since September 2012 which is not supported by old versions of youtube-dl. See above for how to update youtube-dl.

Video URL contains an ampersand and I'm getting some strange

output [1] 2839 or 'v' is not recognized as an internal or external command

That's actually the output from your shell. Since ampersand is one of the special shell characters it's interpreted by the shell preventing you from passing the whole URL to youtube-dl. To disable your shell from interpreting the ampersands (or any other special characters) you have to either put the whole URL in quotes or escape them with a backslash (which approach will work depends on your shell).

For example if your URL is https://www.youtube.com/watch?t=4&v=BaW_jenozKc you should end up with following command:

youtube-dl 'https://www.youtube.com/watch?t=4&v=BaW_jenozKc'

or

youtube-dl https://www.youtube.com/watch?t=4\&v=BaW_jenozKc

For Windows you have to use the double quotes:

youtube-dl "https://www.youtube.com/watch?t=4&v=BaW_jenozKc"

ExtractorError: Could not find JS function u'OF'

In February 2015, the new YouTube player contained a character sequence in a string that was misinterpreted by old versions of youtube-dl. See above for how to update youtube-dl.

HTTP Error 429: Too Many Requests or 402: Payment Required

These two error codes indicate that the service is blocking your IP address because of overuse. Usually this is a soft block meaning that you can gain access again after solving CAPTCHA. Just open a browser and solve a CAPTCHA the service suggests you and after that pass cookies to youtube-dl. Note that if your machine has multiple external IPs then you should also pass exactly the same IP you've used for solving CAPTCHA with --source-address. Also you may need to pass a User-Agent HTTP header of your browser with --user-agent.

If this is not the case (no CAPTCHA suggested to solve by the service) then you can contact the service and ask them to unblock your IP address, or - if you have acquired a whitelisted IP address already - use the --proxy or --source-address options to select another IP address.

SyntaxError: Non-ASCII character

The error

File "youtube-dl", line 2

SyntaxError: Non-ASCII character '\x93' ...

means you're using an outdated version of Python. Please update to Python 2.6 or 2.7.

What is this binary file? Where has the code gone?

Since June 2012 (#342 (https://github.com/ytdl-org/youtube-dl/issues/342)) youtube-dl is packed as an executable zipfile, simply unzip it (might need renaming to youtube-dl.zip first on some

systems) or clone the git repository, as laid out above. If you modify the code, you can run it by executing the __main__.py file. To recompile the executable, run make youtube-dl.

The exe throws an error due to missing MSVCR100.dll

To run the exe you need to install first the Microsoft Visual C++ 2010 Redistributable Package (x86) (https://www.microsoft.com/en-US/download/details.aspx?id=5555).

On Windows, how should I set up ffmpeg and youtube-dl? Where should

I put the exe files?
If you put youtube-dl and ffmpeg in the same directory that you're running the command from, it will work, but that's rather cumbersome.
To make a different directory work - either for ffmpeg, or for youtube-dl, or for both - simply create the directory (say, C:\bin, or C:\Users\<User name>\bin), put all the executables directly in there, and then set your PATH environment variable (https://www.java.com/en/download/help/path.xml) to include that directory.
From then on, after restarting your shell, you will be able to access both youtube-dl and ffmpeg (and youtube-dl will be able to find ffmpeg) by simply typing youtube-dl or ffmpeg, no matter what directory you're in.

How do I put downloads into a specific folder?

Use the -o to specify an output template, for example -o "/home/user/videos/%(title)s-%(id)s.%(ext)s". If you want this for all of your downloads, put the option into your configuration file.

How do I download a video starting with a -?

Either prepend https://www.youtube.com/watch?v= or separate the ID from the options with --:

youtube-dl -- -wNyEUrxzFU

youtube-dl "https://www.youtube.com/watch?v=-wNyEUrxzFU"

How do I pass cookies to youtube-dl?

Use the --cookies option, for example --cookies /path/to/cookies/file.txt.
In order to extract cookies from browser use any conforming browser extension for exporting cookies. For example, cookies.txt (https://chrome.google.com/webstore/detail/cookiestxt/njabckikapfpffapmjgojcnbfjonfjfg) (for Chrome) or cookies.txt (https://addons.mozilla.org/en-US/firefox/addon/cookies-txt/) (for Firefox).
Note that the cookies file must be in Mozilla/Netscape format and the first line of the cookies file must be either # HTTP Cookie File or # Netscape HTTP Cookie File. Make sure you have correct

newline format (https://en.wikipedia.org/wiki/Newline) in the cookies file and convert newlines if necessary to correspond with your OS, namely CRLF (\r\n) for Windows and LF (\n) for Unix and Unix-like systems (Linux, macOS, etc.). HTTP Error 400: Bad Request when using --cookies is a good sign of invalid newline format.

Passing cookies to youtube-dl is a good way to workaround login when a particular extractor does not implement it explicitly. Another use case is working around CAPTCHA (https://en.wikipedia.org/wiki/CAPTCHA) some websites require you to solve in particular cases in order to get access (e.g. YouTube, CloudFlare).

How do I stream directly to media player?

You will first need to tell youtube-dl to stream media to stdout with -o -, and also tell your media player to read from stdin (it must be capable of this for streaming) and then pipe former to latter. For example, streaming to vlc (https://www.videolan.org/) can be achieved with:

youtube-dl -o - "https://www.youtube.com/watch?v=BaW_jenozKcj" | vlc -

How do I download only new videos from a playlist?

Use download-archive feature. With this feature you should initially download the complete playlist with --download-archive /path/to/download/archive/file.txt that will record identifiers of all the videos in a special file. Each subsequent run with the same --download-archive will download only new videos and skip all videos that have been downloaded before. Note that only successful downloads are recorded in the file.

For example, at first,

youtube-dl --download-archive archive.txt "https://www.youtube.com/playlist?list=PLwiyx1dc3P2JR9N8gQaQN_BCvlSlap7re"

will download the complete PLwiyx1dc3P2JR9N8gQaQN_BCvlSlap7re playlist and create a file archive.txt. Each subsequent run will only download new videos if any:

youtube-dl --download-archive archive.txt "https://www.youtube.com/playlist?list=PLwiyx1dc3P2JR9N8gQaQN_BCvlSlap7re"

Should I add --hls-prefer-native into my config?

When youtube-dl detects an HLS video, it can download it either with the built-in downloader or ffmpeg. Since many HLS streams are slightly invalid and ffmpeg/youtube-dl each handle some invalid cases better than the other, there is an option to switch the downloader if needed.

When youtube-dl knows that one particular downloader works better for a given website, that downloader will be picked. Otherwise, youtube-dl will pick the best downloader for general compatibility, which at the moment happens to be ffmpeg. This choice may change in future versions of youtube-dl, with improvements of the built-in downloader and/or ffmpeg.

In particular, the generic extractor (used when your website is not in the list of supported sites by youtube-dl (https://ytdl-org.github.io/youtube-dl/supportedsites.html) cannot mandate one specific downloader.

If you put either --hls-prefer-native or --hls-prefer-ffmpeg into your configuration, a different subset of videos will fail to download correctly. Instead, it is much better to file an issue (https://yt-dl.org/bug) or a pull request which details why the native or the ffmpeg HLS downloader is a better choice for your use case.

Can you add support for this anime video site, or site which shows

current movies for free?

As a matter of policy (as well as legality), youtube-dl does not include support for services that specialize in infringing copyright. As a rule of thumb, if you cannot easily find a video that the service is quite obviously allowed to distribute (i.e. that has been uploaded by the creator, the creator's distributor, or is published under a free license), the service is probably unfit for inclusion to youtube-dl.

A note on the service that they don't host the infringing content, but just link to those who do, is evidence that the service should **not** be included into youtube-dl. The same goes for any DMCA note when the whole front page of the service is filled with videos they are not allowed to distribute. A "fair use" note is equally unconvincing if the service shows copyright-protected videos in full without authorization.

Support requests for services that **do** purchase the rights to distribute their content are perfectly fine though. If in doubt, you can simply include a source that mentions the legitimate purchase of content.

How can I speed up work on my issue?

(Also known as: Help, my important issue not being solved!) The youtube-dl core developer team is quite small. While we do our best to solve as many issues as possible, sometimes that can take quite a while. To speed up your issue, here's what you can do:

First of all, please do report the issue at our issue tracker (https://yt-dl.org/bugs). That allows us to coordinate all efforts by users and developers, and serves as a unified point. Unfortunately, the youtube-dl project has grown too large to use personal email as an effective communication channel.

Please read the bug reporting instructions below. A lot of bugs lack all the necessary information. If you can, offer proxy, VPN, or shell access to the youtube-dl developers. If you are able to, test the issue from multiple computers in multiple countries to exclude local censorship or misconfiguration issues.

If nobody is interested in solving your issue, you are welcome to take matters into your own hands and submit a pull request (or coerce/pay somebody else to do so).

Feel free to bump the issue from time to time by writing a small comment ("Issue is still present in youtube-dl version ...from France, but fixed from Belgium"), but please not more than once a month. Please do not declare your issue as important or urgent.

How can I detect whether a given URL is supported by youtube-dl?

For one, have a look at the list of supported sites (docs/supportedsites.md). Note that it can sometimes happen that the site changes its URL scheme (say, from https://example.com/video/1234567 to https://example.com/v/1234567) and youtube-dl reports an URL of a service in that list as unsupported. In that case, simply report a bug.

It is *not* possible to detect whether a URL is supported or not. That's because youtube-dl contains a generic extractor which matches **all** URLs. You may be tempted to disable, exclude, or remove the generic extractor, but the generic extractor not only allows users to extract videos from lots of websites that embed a video from another service, but may also be used to extract video from a service that it's hosting itself. Therefore, we neither recommend nor support disabling, excluding, or removing the generic extractor.

If you want to find out whether a given URL is supported, simply call youtube-dl with it. If you get no videos back, chances are the URL is either not referring to a video or unsupported. You can find out which by examining the output (if you run youtube-dl on the console) or catching an UnsupportedError exception if you run it from a Python program.

Why do I need to go through that much red tape when filing bugs?

Before we had the issue template, despite our extensive bug reporting instructions, about 80% of the issue reports we got were useless, for instance because people used ancient versions hundreds of releases old, because of simple syntactic errors (not in youtube-dl but in general shell usage), because the problem was already reported multiple times before, because people did not actually read an error message, even if it said "please install ffmpeg", because people did not mention the URL they were trying to download and many more simple, easy-to-avoid problems, many of whom were totally unrelated to youtube-dl.

youtube-dl is an open-source project manned by too few volunteers, so we'd rather spend time fixing bugs where we are certain none of those simple problems apply, and where we can be reasonably confident to be able to reproduce the issue without asking the reporter repeatedly. As such, the output of youtube-dl -v YOUR_URL_HERE is really all that's required to file an issue. The issue template also guides you through some basic steps you can do, such as checking that your version of youtube-dl is current.

Developer Instructions

Most users do not need to build youtube-dl and can download the builds (https://ytdl-org.github.io/youtube-dl/download.html) or get them from their distribution.

To run youtube-dl as a developer, you don't need to build anything either. Simply execute

python -m youtube_dl

To run the test, simply invoke your favorite test runner, or execute a test file directly; any of the following work:

python -m unittest discover

python test/test_download.py

nosetests

See item 6 of new extractor tutorial for how to run extractor specific test cases.
If you want to create a build of youtube-dl yourself, you'll need

- python
- make (only GNU make is supported)
- pandoc
- zip
- nosetests

Adding support for a new site

If you want to add support for a new site, first of all **make sure** this site is **not dedicated to copyright infringement (README.md#can-you-add-support-for-this-anime-video-site-or-site-which-shows-current-movies-for-free)**. youtube-dl does **not support** such sites thus pull requests adding support for them **will be rejected**.
After you have ensured this site is distributing its content legally, you can follow this quick list (assuming your service is called yourextractor):

1. Fork this repository (https://github.com/ytdl-org/youtube-dl/fork)
2. Check out the source code with:
    git clone git@github.com:YOUR_GITHUB_USERNAME/youtube-dl.git
3. Start a new git branch with
4. cd youtube-dl

    git checkout -b yourextractor
5. Start with this simple template and save it to youtube_dl/extractor/yourextractor.py:
6. # coding: utf-8
7. from __future__ import unicode_literals
8.
9. from .common import InfoExtractor
10.
11.
12. class YourExtractorIE(InfoExtractor):
13.   _VALID_URL = r'https?://(?:www\.)?yourextractor\.com/watch/(?P<id>[0-9]+)'
14.   _TEST = {
15.   'url': 'https://yourextractor.com/watch/42',
16.   'md5': 'TODO: md5 sum of the first 10241 bytes of the video file (use --test)',
17.   'info_dict': {

```
18.         'id': '42',
19.         'ext': 'mp4',
20.         'title': 'Video title goes here',
21.         'thumbnail': r're:^https?://.*\.jpg$',
22.         # TODO more properties, either as:
23.         # * A value
24.         # * MD5 checksum; start the string with md5:
25.         # * A regular expression; start the string with re:
26.         # * Any Python type (for example int or float)
27.     }
28. }
29.
30. def _real_extract(self, url):
31.     video_id = self._match_id(url)
32.     webpage = self._download_webpage(url, video_id)
33.
34.     # TODO more code goes here, for example ...
35.     title = self._html_search_regex(r'<h1>(.+?)</h1>', webpage, 'title')
36.
37.     return {
38.         'id': video_id,
39.         'title': title,
40.         'description': self._og_search_description(webpage),
41.         'uploader': self._search_regex(r'<div[^>]+id="uploader"[^>]*>([^<]+)<', webpage, 'uploader', fatal=False),
42.         # TODO more properties (see youtube_dl/extractor/common.py)
    }
```

43. Add an import in youtube_dl/extractor/extractors.py (https://github.com/ytdl-org/youtube-dl/blob/master/youtube_dl/extractor/extractors.py).

44. Run python test/test_download.py TestDownload.test_YourExtractor. This *should fail* at first, but you can continually re-run it until you're done. If you decide to add more than one test, then rename _TEST to _TESTS and make it into a list of dictionaries. The tests will then be

named TestDownload.test_YourExtractor, TestDownload.test_YourExtractor_1, TestDownload.test_Your Extractor_2, etc. Note that tests with only_matching key in test's dict are not counted in.

45. Have a look at youtube_dl/extractor/common.py (https://github.com/ytdl-org/youtube-dl/blob/master/youtube_dl/extractor/common.py) for possible helper methods and a detailed description of what your extractor should and may return (https://github.com/ytdl-org/youtube-dl/blob/7f41a598b3fba1bcab2817de64a08941200aa3c8/youtube_dl/extractor/common.py#L94-L303). Add tests and code for as many as you want.

46. Make sure your code follows youtube-dl coding conventions and check the code with flake8 (http://flake8.pycqa.org/en/latest/index.html#quickstart):

 $ flake8 youtube_dl/extractor/yourextractor.py

47. Make sure your code works under all Python (https://www.python.org/) versions claimed supported by youtube-dl, namely 2.6, 2.7, and 3.2+.

48. When the tests pass, add (https://git-scm.com/docs/git-add) the new files and commit (https://git-scm.com/docs/git-commit) them and push (https://git-scm.com/docs/git-push) the result, like this:

49. $ git add youtube_dl/extractor/extractors.py

50. $ git add youtube_dl/extractor/yourextractor.py

51. $ git commit -m [yourextractor] Add new extractor'

 $ git push origin yourextractor

52. Finally, create a pull request (https://help.github.com/articles/creating-a-pull-request). We'll then review and merge it.

In any case, thank you very much for your contributions!

youtube-dl coding conventions

This section introduces a guide lines for writing idiomatic, robust and future-proof extractor code.
Extractors are very fragile by nature since they depend on the layout of the source data provided by 3rd party media hosters out of your control and this layout tends to change. As an extractor implementer your task is not only to write code that will extract media links and metadata correctly but also to minimize dependency on the source's layout and even to make the code foresee potential future changes and be ready for that. This is important because it will allow the extractor not to break on minor layout changes thus keeping old youtube-dl versions working. Even though this breakage issue is easily fixed by emitting a new version of youtube-dl with a fix incorporated, all the previous versions become broken in all repositories and distros' packages that may not be so prompt in fetching the update from us. Needless to say, some non rolling release distros may never receive an update at all.

Mandatory and optional metafields

For extraction to work youtube-dl relies on metadata your extractor extracts and provides to youtube-dl expressed by an information dictionary (https://github.com/ytdl-org/youtube-dl/blob/7f41a598b3fba1bcab2817de64a08941200aa3c8/youtube_dl/extractor/common.py#L94-L303) or simply *info dict*. Only the following meta fields in the *info dict* are considered mandatory for a successful extraction process by youtube-dl:

* id (media identifier)
* title (media title)
* url (media download URL) or formats

In fact only the last option is technically mandatory (i.e. if you can't figure out the download location of the media the extraction does not make any sense). But by convention youtube-dl also treats id and title as mandatory. Thus the aforementioned metafields are the critical data that the extraction does not make any sense without and if any of them fail to be extracted then the extractor is considered completely broken.
Any field (https://github.com/ytdl-org/youtube-dl/blob/7f41a598b3fba1bcab2817de64a08941200aa3c8/youtube_dl/extractor/common.py#L188-L303) apart from the aforementioned ones are considered **optional**. That means that extraction should be **tolerant** to situations when sources for these fields can potentially be unavailable (even if they are always available at the moment) and **future-proof** in order not to break the extraction of general purpose mandatory fields.

Example

Say you have some source dictionary meta that you've fetched as JSON with HTTP request and it has a key summary:

meta = self._download_json(url, video_id)

Assume at this point meta's layout is:

```
{
    ...
    "summary": "some fancy summary text",
    ...
}
```

Assume you want to extract summary and put it into the resulting info dict as description.

Since description is an optional meta field you should be ready that this key may be missing from the meta dict, so that you should extract it like:

description = meta.get('summary') # correct

and not like:

description = meta['summary'] # incorrect

The latter will break extraction process with KeyError if summary disappears from meta at some later time but with the former approach extraction will just go ahead with description set to None which is perfectly fine (remember None is equivalent to the absence of data). Similarly, you should pass fatal=False when extracting optional data from a webpage with _search_regex, _html_search_regex or similar methods, for instance:

```
description = self._search_regex(
    r'<span[^>]+id="title"[^>]*>([^<]+)<',
    webpage, 'description', fatal=False)
```

With fatal set to False if _search_regex fails to extract description it will emit a warning and continue extraction.

You can also pass default=<some fallback value>, for example:

```
description = self._search_regex(
    r'<span[^>]+id="title"[^>]*>([^<]+)<',
    webpage, 'description', default=None)
```

On failure this code will silently continue the extraction with description set to None. That is useful for metafields that may or may not be present.

Provide fallbacks

When extracting metadata try to do so from multiple sources. For example if title is present in several places, try extracting from at least some of them. This makes it more future-proof in case some of the sources become unavailable.

Example

Say meta from the previous example has a title and you are about to extract it. Since title is a mandatory meta field you should end up with something like:

```
title = meta['title']
```

If title disappears from meta in future due to some changes on the hoster's side the extraction would fail since title is mandatory. That's expected.

Assume that you have some another source you can extract title from, for example og:title HTML meta of a webpage. In this case you can provide a fallback scenario:

```
title = meta.get('title') or self._og_search_title(webpage)
```

This code will try to extract from meta first and if it fails it will try extracting og:title from a webpage.

Regular expressions

Don't capture groups you don't use

Capturing group must be an indication that it's used somewhere in the code. Any group that is not used must be non capturing.

Example

Don't capture id attribute name here since you can't use it for anything anyway.
Correct:
r'(?:id|ID)=(?P<id>\d+)'
Incorrect:
r'(id|ID)=(?P<id>\d+)'

Make regular expressions relaxed and flexible

When using regular expressions try to write them fuzzy, relaxed and flexible, skipping insignificant parts that are more likely to change, allowing both single and double quotes for quoted values and so on.

Example

Say you need to extract title from the following HTML code:

<span style="position: absolute; left: 910px; width: 90px; float: right; z-

index: 9999;" class="title">some fancy title
The code for that task should look similar to:
title = self._search_regex(

  r'<span[^>]+class="title"[^>]*>([^<]+)', webpage, 'title')
Or even better:
title = self._search_regex(

  r'<span[^>]+class=(["\'])title\1[^>]*>(?P<title>[^<]+)',

  webpage, 'title', group='title')
Note how you tolerate potential changes in the style attribute's value or switch from using double

quotes to single for class attribute:
The code definitely should not look like:
title = self._search_regex(

  r'<span style="position: absolute; left: 910px; width: 90px; float: right; z-

index: 9999;" class="title">(.*?)',

```
    webpage, 'title', group='title')
```

Long lines policy

There is a soft limit to keep lines of code under 80 characters long. This means it should be respected if possible and if it does not make readability and code maintenance worse.
For example, you should **never** split long string literals like URLs or some other often copied entities over multiple lines to fit this limit:
Correct:

```
'https://www.youtube.com/watch?v=FqZTN594JQw&list=PLMYEtVRpaqY00V9W81Cwmzp6N6vZqfUKD4'
```

Incorrect:

```
'https://www.youtube.com/watch?v=FqZTN594JQw&list='
```

```
'PLMYEtVRpaqY00V9W81Cwmzp6N6vZqfUKD4'
```

Inline values

Extracting variables is acceptable for reducing code duplication and improving readability of complex expressions. However, you should avoid extracting variables used only once and moving them to opposite parts of the extractor file, which makes reading the linear flow difficult.

Example

Correct:

```
title = self._html_search_regex(r'<title>([^<]+)</title>', webpage, 'title')
```

Incorrect:

```
TITLE_RE = r'<title>([^<]+)</title>'
```

```
# ...some lines of code...
```

```
title = self._html_search_regex(TITLE_RE, webpage, 'title')
```

Collapse fallbacks

Multiple fallback values can quickly become unwieldy. Collapse multiple fallback values into a single expression via a list of patterns.

Example

Good:

```
description = self._html_search_meta(
    ['og:description', 'description', 'twitter:description'],
    webpage, 'description', default=None)
```

Unwieldy:
description = (

    self._og_search_description(webpage, default=None)

    or self._html_search_meta('description', webpage, default=None)

    or self._html_search_meta('twitter:description', webpage, default=None))

Methods supporting list of patterns
are: _search_regex, _html_search_regex, _og_search_property, _html_search_meta.

Trailing parentheses

Always move trailing parentheses after the last argument.

Example

Correct:

    lambda x: x['ResultSet']['Result'][0]['VideoUrlSet']['VideoUrl'],

    list)

Incorrect:

    lambda x: x['ResultSet']['Result'][0]['VideoUrlSet']['VideoUrl'],

    list,

)

Use convenience conversion and parsing functions

Wrap all extracted numeric data into safe functions
from youtube_dl/utils.py (https://github.com/ytdl-org/youtube-dl/blob/master/youtube_dl/utils.py): int_or_none, float_or_none. Use them for string to number conversions as well.

Use url_or_none for safe URL processing.

Use try_get for safe metadata extraction from parsed JSON.

Use unified_strdate for uniform upload_date or any YYYYMMDD meta field

extraction, unified_timestamp for

uniform timestamp extraction, parse_filesize for filesize extraction, parse_count for count meta fields

extraction, parse_resolution, parse_duration for duration extraction, parse_age_limit for age_limit extraction.

Explore youtube_dl/utils.py (https://github.com/ytdl-org/youtube-dl/blob/master/youtube_dl/utils.py) for more useful convenience functions.

More examples

Safely extract optional description from parsed JSON

description = try_get(response, lambda x: x['result']['video'][0]['summary'], compat_str)

Safely extract more optional metadata

video = try_get(response, lambda x: x['result']['video'][0], dict) or {}

description = video.get('summary')

duration = float_or_none(video.get('durationMs'), scale=1000)

view_count = int_or_none(video.get('views'))

Embedding Youtube-Dl

youtube-dl makes the best effort to be a good command-line program, and thus should be callable from any programming language. If you encounter any problems parsing its output, feel free to create a report (https://github.com/ytdl-org/youtube-dl/issues/new).
From a Python program, you can embed youtube-dl in a more powerful fashion, like this:

from __future__ import unicode_literals

import youtube_dl

ydl_opts = {}

with youtube_dl.YoutubeDL(ydl_opts) as ydl:

   ydl.download(['https://www.youtube.com/watch?v=BaW_jenozKc'])

Most likely, you'll want to use various options. For a list of options available, have a look at youtube_dl/YoutubeDL.py (https://github.com/ytdl-org/youtube-dl/blob/3e4cedf9e8cd3157df2457df7274d0c842421945/youtube_dl/YoutubeDL.py#L137-L312).
For a start, if you want to intercept youtube-dl's output, set a logger object.

Here's a more complete example of a program that outputs only errors (and a short message after the download is finished), and downloads/converts the video to an mp3 file:

from __future__ import unicode_literals

import youtube_dl

class MyLogger(object):

   def debug(self, msg):

```python
            pass

    def warning(self, msg):
        pass

    def error(self, msg):
        print(msg)


def my_hook(d):
    if d['status'] == 'finished':
        print('Done downloading, now converting ...')


ydl_opts = {
    'format': 'bestaudio/best',
    'postprocessors': [{
        'key': 'FFmpegExtractAudio',
        'preferredcodec': 'mp3',
        'preferredquality': '192',
    }],
    'logger': MyLogger(),
    'progress_hooks': [my_hook],
}
with youtube_dl.YoutubeDL(ydl_opts) as ydl:
    ydl.download(['https://www.youtube.com/watch?v=BaW_jenozKc'])
```

Bugs

Bugs and suggestions should be reported at: <https://github.com/ytdl-org/youtube-dl/issues>. Unless you were prompted to or there is another pertinent reason (e.g. GitHub fails to accept the bug report), please do not send bug reports via personal email. For discussions, join us in the IRC channel #youtube-dl (irc://chat.freenode.net/#youtube-dl) on freenode (webchat (https://webchat.freenode.net/?randomnick=1&channels=youtube-dl)).

Please include the full output of youtube-dl when run with -v, i.e. **add -v** flag to **your command line**, copy the **whole** output and post it in the issue body wrapped in ``` for better formatting. It should look similar to this:

$ youtube-dl -v <your command line>

[debug] System config: []

[debug] User config: []

[debug] Command-line args: [u'-v', u'https://www.youtube.com/watch?v=BaW_jenozKcj']

[debug] Encodings: locale cp1251, fs mbcs, out cp866, pref cp1251

[debug] youtube-dl version 2015.12.06

[debug] Git HEAD: 135392e

[debug] Python version 2.6.6 - Windows-2003Server-5.2.3790-SP2

[debug] exe versions: ffmpeg N-75573-g1d0487f, ffprobe N-75573-g1d0487f, rtmpdump 2.4

[debug] Proxy map: {}

...

Do not post screenshots of verbose logs; only plain text is acceptable.
The output (including the first lines) contains important debugging information. Issues without the full output are often not reproducible and therefore do not get solved in short order, if ever. Please re-read your issue once again to avoid a couple of common mistakes (you can and should use this as a checklist):

Is the description of the issue itself sufficient?

We often get issue reports that we cannot really decipher. While in most cases we eventually get the required information after asking back multiple times, this poses an unnecessary drain on our resources. Many contributors, including myself, are also not native speakers, so we may misread some parts.
So please elaborate on what feature you are requesting, or what bug you want to be fixed. Make sure that it's obvious

- What the problem is
- How it could be fixed
- How your proposed solution would look like

If your report is shorter than two lines, it is almost certainly missing some of these, which makes it hard for us to respond to it. We're often too polite to close the issue outright, but the missing info makes misinterpretation likely. As a committer myself, I often get frustrated by these issues, since the only possible way for me to move forward on them is to ask for clarification over and over.
For bug reports, this means that your report should contain the *complete* output of youtube-dl when called with the -v flag. The error message you get for (most) bugs even says so, but you would not believe how many of our bug reports do not contain this information.

If your server has multiple IPs or you suspect censorship, adding --call-home may be a good idea to get more diagnostics. If the error is ERROR: Unable to extract ... and you cannot reproduce it from multiple countries, add --dump-pages (warning: this will yield a rather large output, redirect it to the file log.txt by adding >log.txt 2>&1 to your command-line) or upload the .dump files you get when you add --write-pages somewhere (https://gist.github.com/).
Site support requests must contain an example URL. An example URL is a URL you might want to download, like https://www.youtube.com/watch?v=BaW_jenozKc. There should be an obvious video present. Except under very special circumstances, the main page of a video service (e.g. https://www.youtube.com/) is *not* an example URL.

Are you using the latest version?

Before reporting any issue, type youtube-dl -U. This should report that you're up-to-date. About 20% of the reports we receive are already fixed, but people are using outdated versions. This goes for feature requests as well.