bepasty Documentation

Release 0.4.0

The Bepasty Team (see AUTHORS file)

Jun 11, 2017

Contents

bepasty is like a pastebin for every kind of file (text, image, audio, video, documents, ...).

You can upload multiple files at once, simply by drag and drop.

CHAPTER 1

Contents

bepasty

bepasty is like a pastebin for all kinds of files (text, image, audio, video, documents, ..., binary).

The documentation is there: http://bepasty-server.readthedocs.org/en/latest/

Features

- Generic:
  - you can upload multiple files at once, simply by drag and drop
  - after upload, you get a unique link to a view of each file
  - on that view, we show actions you can do with the file, metadata of the file and, if possible, we also render the file contents
  - if you uploaded multiple files, you can create a pastebin with the list of all these files - with a single click!
  - Set an expiration date for your files
- Text files:
  - we highlight all text file types supported by pygments (a lot!)
  - we display line numbers
  - we link from line numbers to their anchors, so you can easily get a link to a specific line
- Image files:
  - we show the image (format support depends on browser)
- Audio and video files:
  - we show the html5 player for it (format support depends on browser)
- PDFs:

  - we support rendering PDFs in your browser (if your browser is able to)
- Storage: we use a storage backend api, currently we have backends for:
  - filesystem storage (just use a filesystem directory to store <uuid>.meta and <uuid>.data files)
  - currently there are no other storage implementations in master branch and releases. The "ceph cluster" storage implementation has issues and currently lives in branch "ceph-storage" until these issues are fixed.
- Keeping some control:
  - flexible permissions: create, read, delete, admin
  - assign permissions to users of login secrets
  - assign default permissions to not-logged-in users
  - you can purge files from storage by age, inactivity, size, type, ...
  - you can do consistency checks on the storage

Development

```
# Clone the official bepasty-server (or your fork, if you want to send PULL requests)
git clone https://github.com/bepasty/bepasty-server.git
cd bepasty-server
# Create a new virtualenv
virtualenv ~/bepasty
# Activate the virtualenv
source ~/bepasty/bin/activate
# This will use the current directory for the installed package
# Very useful during development! It will also autoreload when files are changed
pip install -e .
# Run the bepasty-server in debug mode. The server is reachable in http://127.0.0.
↪1:5000
bepasty-server --debug
```

Using bepasty's web interface

Logging in and Permissions

You may need to log in to get enough permissions required for the misc. functions of bepasty.

Your current permissions are shown on the web interface (in the navigation bar).

To log in, you need to know credentials - ask the admin who runs the site.

The site admin can assign permissions to login credentials (and also to the anonymous, not logged-in user):

- create: be able to create pastebins
- read: be able to read / download pastebins
- delete: be able to delete pastebins
- list: be able to list (discover) all pastebins
- admin: be able to lock/unlock files, do actions even if upload is not completed or item is locked

Be careful about who you give what permissions - especially “admin” and “list” are rather critical ones.

If you want good privacy, do not give “list” to anybody (except site administrator maybe).

If you want to do everything rather in the public, you may give “list” to users (or even use it by default for not-logged-in users).

“admin” likely should be given only to very trusted people, like site administrator.

Pasting text

Just paste it into that big upper text input box.

Content-Type: Below the box you can optionally choose the type of your content (if you don’t, it will become plain text). Just start typing some letters of the type, e.g. if you pasted some python code, type pyt and see how it offers you some choices based on that. Based on that type, we will highlight your text (using the Pygments library, which supports a lot of text formats).

File name: You can optionally give a filename for your paste. If someone later downloads it, the browser will use the filename you gave. If you don’t give a filename, bepasty will make something up.

Maximum lifetime: The file will be automatically deleted after this time is over

When finished, click on “Submit”. bepasty will save your text using a unique ID and redirect you to the URL where you can view or download your pasted text.

Uploading files

See that big box below the text input box - you can:

- click it to upload files via the file selection dialogue of your browser
- drag files from your desktop and drop them there

Note: some features require a modern browser, like a current Firefox or Chrome/Chromium with Javascript enabled.

It will show you a progress indication while the files upload.

After the files are uploaded, bepasty will show you a list of links to the individual views of each uploaded file. Make sure you keep all the links (open the links in new tabs / new windows) - they are the only way to access the files.

Additionally, bepasty prepared a file list for you (that has all the unique IDs of your uploaded files). If you create a list item by hitting the respective button, bepasty will store that list in yet another pastebin item, so you need to remember only that one URL. It’s also a nice way to communicate a collection of files as you only need to communicate that one URL (not each individual file’s URL).

Viewing / Downloading files

Just visit the file’s unique URL to view, download or delete it.

bepasty will show you metadata like:

- file name
- precise file size
- upload date/time (UTC)
- (last) download date/time (UTC) - viewing the data also counts as download
- expiration date, if set

- sha256 hash of the file contents

bepasty also supports directly displaying the data, for these content types:

- lists of files (if a list item was created at upload time)
- text files (highlighted depending on the content-type)
- PDFs (if you browser can render PDFs or has a plugin doing that)
- image files, like jpeg, png and svg
- audio/video files (using the html5 player widget, format support might depend on your browser and OS)
- for other file types, you need to download them and open them with the appropriate application

File hashes

If you're unfamiliar with hashes like SHA256, you might wonder what they are good for and why we show them.

A hash is something like a checksum or fingerprint of the file contents. We compute the hash while or directly after the file upload. If you have 2 files at different places and they have the same SHA256 hash, you can be pretty sure that they have completely identical contents.

If you are looking at a file you just uploaded yourself, it might be interesting to compare the size and hash with the values you see for your local file to make sure the upload worked correctly.

Same after you download a file: check whether the hash matches for the file on bepasty and your downloaded file.

If you transfer a file from location A via bepasty (B) to location C, you can also compare the file hashes at locations A and C to make sure the file was not modified or corrupted while being transferred.

Important stuff to remember

- if you get credentials from an admin, do not make them available to other persons except if explicitly allowed
- files may go away at any time, always remember that a pastebin site is only for short-term temporary storage (how long this is depends on the site's / site admin's policy and available disk space)

Using bepasty's REST-API

The Rest-API enables you to upload and download files, as well as retrieve informations about the file on the server.

Currently (Version 0.3) the REST API provides four API Endpoints:

```
GET  /apis/rest
GET  /apis/rest/items
POST /apis/rest/items
GET  /apis/rest/items/<itemname>
GET  /apis/rest/items/<itemname>/download
```

Retrieving information for uploading

API Interface:

```
GET /apis/rest
```

Example Response:

```
{
    MAX_ALLOWED_FILE_SIZE: 5000000000,
    MAX_BODY_SIZE: 1048576
}
```

This interface will give you important infos for uploading and downloading files to your bepasty server. By now only the MAX_BODY_SIZE will be delivered to you, as no more info is available.

MAX_BODY_SIZE The maximum size of a post request's body. This is limited by the webserver and other middleware. See the documentation for more information. This also gives you the maximum size for the chunked upload.

MAX_ALLOWED_FILE_SIZE The maximum allowed filesize that can be stored on the server. Files uploads bigger than this limit will be aborted and the file on the server will be deleted.

Uploading a file

API Interface:

```
POST /apis/rest/items
```

When uploading a file, chunked upload is mandatory. Check the MAX_BODY_SIZE for the maximum chunk size that can be sent to the server. The body of the post request contains the base64 encoded binary of the file to be uploaded.

POST Request by the client

Post Request Body Contains the base64 encoded binary of the file to be uploaded.

The following headers *can (cursive)* or **must (bold)** be delivered by every post request to the server:

Content-Range The content-range header follows the specification by the w3c (http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.16). It has to be provided consistently and can resume a aborted file upload, together with the transaction-ID.

Transaction-ID The transaction-ID will be provided by the server after the first upload chunk. After that first chunk, the transaction-id has to be provided by the client, to continue uploading the file.

Content-Type The content-type of the file uploaded to the server. If the content-type is not given, the server will guess the content-type by the filename. If this fails the content-type will be 'application/octet-stream'

Content-Length The content-length is mostly ignored by the server. It can be used to indicate the final file size. If your final file size is bigger than the maximum allowed size on the server, the upload will be aborted. The real filesize will be calculated by the server while uploading.

Content-Filename The content-filename header can be used to name the file on the server. If no content-filename is passed, the server will generate a name from scratch. Maximum filename size is 50 characters.

Maxlife-Unit The maxlife-unit can be used with the maxlife-value header to define a lifetime for the file that is uploaded. The unit has to be one of these:

```
['MINUTES', 'HOURS', 'DAYS', 'WEEKS', 'MONTHS', 'YEARS', 'FOREVER']
```

If this header is ommited the unit will be forever

Maxlife-Value The maxlife-value header defines the value of the maxlife-unit.

POST Response by the server

Transaction-ID Transaction-ID provided for continued upload in a chunked upload process.

Content-Disposition The URI of the newly uploaded file on the server. Will only be provided when upload is finished and successful.

Retrieving information about a file

API Interface:

```
GET /apis/rest/items/<itemname>
```

GET Request by the client

itemname The itemname of the file requested.

GET Response by the server

Example Response:

```
{
file-meta: {
    complete: true,
    filename: "Wallpaper Work.7z",
    hash: "dded24ba6f1d953bedb9d2745635a6f7462817061763b0d70f68b7952722f275",
    locked: false,
    size: 150225567,
    timestamp-download: 1414483078,
    timestamp-max-life: -1,
    timestamp-upload: 1414443534,
    type: "application/x-7z-compressed"
    },
uri: "/apis/rest/items/N24bFRZm"
}
```

URI The URI of the file on the server. Used to link to the download.

File-Meta

> ***Filename*** The Filename of the uploaded file.
>
> ***Size*** The calculated size of the file on the server.
>
> ***Timestamp-Upload*** The timestamp of the moment the file was uploaded.
>
> ***Timestamp-Download*** The timestamp of the last download.
>
> ***Timestamp-Max_life*** The lifetime timestamp of the file in seconds. -1 means to keep the file forever.
>
> ***Complete*** True if the file upload is completed. False if it isn't
>
> ***Locked*** Wether the file is locked or not.
>
> ***Hash*** The sha256 hash of the file uploaded. Calculated by the server.

Type Mimetype of the file uploaded. If no filetype is provided this will be set to 'application/octet-stream'.

Retrieving Item List

API Interface:

```
GET /apis/rest/items
```

GET Request by the client

No Parameters

GET Response by the server

Example Response:

```
{
  "N24bFRZm": {
    file-meta: {
        complete: true,
        filename: "Wallpaper Work.7z",
        hash: "dded24ba6f1d953bedb9d2745635a6f7462817061763b0d70f68b7952722f275",
        locked: false,
        size: 150225567,
        timestamp-download: 1414483078,
        timestamp-max-life: -1,
        timestamp-upload: 1414443534,
        type: "application/x-7z-compressed"
      },
      uri: "/apis/rest/items/N24bFRZm"
  }, ...
}
```

Parameters are the same as in *Retrieving information about a file*.

Downloading a file

API Interface:

```
GET /apis/rest/items/<itemname>/download
```

GET Response by the server

Example Response:: Content-Type: application/x-7z-compressed Content-Length: 150225568 Content-Disposition: attachment; filename="Wallpaper Work.7z" Content-Range: bytes 0-150225567/150225567

Opens up a stream and delivers the binary data directly. The above headers can be found in the HTTP Response.

Using bepasty with non-web clients

pastebinit

pastebinit is a popular pastebin client (included in debian, ubuntu and maybe elsewhere) that can be configured to work with bepasty:

Configuration

~/.pastebinit.xml:

```
<pastebinit>
    <pastebin>https://bepasty.example.org</pastebin>
    <format></format>
</pastebinit>
```

Notes:

- we set an empty default format so pastebinit will transmit this (and not its internal format default [which is "text" and completely useless for us as it is not a valid contenttype])

~/.pastebin.d/bepasty.conf:

```
[pastebin]
basename = bepasty.example.org
regexp = https://bepasty.example.org

[format]
content = text
title = filename
format = contenttype
page = page
password = token

[defaults]
page = +upload
```

Usage

Simplest:

```
echo "test" | pastebinit
```

More advanced:

```
# give title (filename), password, input file
pastebinit -t example.py -p yourpassword -i example.py

# read from stdin, give title (filename), give format (contenttype)
cat README | pastebinit -t README -f text/plain
```

Notes:

- we use -t ("title") to transmit the desired filename (we do not have a "title", but the filename that is used for downloading the pastebin is prominently displayed above the content, so can be considered as title also).

- bepasty guesses the contenttype from the filename given with -t. if you do not give a filename there or the contenttype is not guessable from it, you may need to give -f also (e.g. -f text/plain).
- if you give the contenttype, but not the filename, bepasty will make up a filename.
- you need to use -p if the bepasty instance you use requires you to log in before you can create pastebins.

Quickstart

Installing bepasty

You can install bepasty either from PyPi (latest release) or from the git repository (latest available code).

```
# from PyPi:
pip install bepasty-server

# from git repo
pip install -e git+https://github.com/bepasty/bepasty-server.git#egg=bepasty-server
```

Configuring bepasty

Before you can use bepasty, you need to carefully configure it (it won't work in default configuration and most of the configuration settings need your attention).

When setting up permissions and giving out login secrets, carefully think about whom you give which permissions, especially when setting up the `DEFAULT_PERMISSIONS` (which apply to not-logged-in users).

Here is the documentation straight from its config:

class `bepasty.config.`**`Config`**

This is the basic configuration class for bepasty.

`APP_BASE_PATH` = **None**

base URL path of app (if not served on root URL, but e.g. on http://example.org/bepasty). setting this to a non-None value will activate the PrefixMiddleware that splits PATH_INFO into SCRIPT_NAME (== APP_BASE_PATH) and the rest of PATH_INFO.

`DEFAULT_PERMISSIONS` = **''**

not-logged-in users get these permissions - usually they are either no permissions ('') or read-only ('read').

`MAX_ALLOWED_FILE_SIZE` = **5000000000**

The site admin can set some maximum allowed file size he wants to accept here. This is the maximum size an uploaded file may have.

`MAX_BODY_SIZE` = **1048576**

The maximum http request body size. This is an information given to rest api clients so they can adjust their chunk size accordingly.

This needs to be in sync with (or at least not beyond) the web server settings: apache: LimitRequestBody 1048576 # apache default is 0 (unlimited) nginx: client_max_body_size 1m; # nginx default (== 1048576)

`MAX_RENDER_SIZE` = **{'': 1000000, 'audio/': 1000000000, 'image/': 10000000, 'HIGHLIGHT_TYPES': 100000, 'video/'**

Setup maximum file sizes for specific content-types. If an item is beyond the limit set for its type, it will not be rendered, but just offered for download. Lookup within MAX_RENDER_SIZE is done by first-match and it is automatically sorted for longer content-type- prefixes first.

Format of entries: content-type-prefix: max_size

PERMANENT_SESSION = False
use a permanent session (True, cookie will expire after the given time, see below) or not (False, cookie will get removed when browser is closed)

PERMANENT_SESSION_LIFETIME = 2678400
lifetime of the permanent session (in seconds)

PERMISSIONS = {}
logged-in users may get more permissions. a user may have a login secret to log in and, depending on that secret, he/she will get the permissions configured here.

you can use same secret / same permissions for all privileged users or set up different secrets / different permissions for each user.

PERMISSIONS is a dict that maps secrets to permissions, use it like:

```
PERMISSIONS = {
    'myadminsecret': 'admin,list,create,read,delete',
    'uploadersecret': 'create,read',
}
```

SECRET_KEY = ''
server secret key needed for safe session cookies. you must set a very long, very random, very secret string here, otherwise bepasty will not work (and crash when trying to log in)!

SESSION_COOKIE_SECURE = True
transmit cookie only over https (if you use http, set this to False)

SITENAME = 'bepasty.example.org'
name of this site (put YOUR bepasty fqdn here)

STORAGE = 'filesystem'
Define storage backend, choose from:

- 'filesystem'

STORAGE_FILESYSTEM_DIRECTORY = '/tmp/'
Filesystem storage path

UPLOAD_LOCKED = False
Whether files are automatically locked after upload.

If you want to require admin approval and manual unlocking for each uploaded file, set this to True.

To create a local and non-default configuration, copy `bepasty/config.py` to e.g. `/srv/bepasty/bepasty.conf` first, remove the `class Config` and remove all indents in the file. The comments can be removed too, if you feel the need to. At last modify these two configs variables: then modify it:

```
# Note: no Config class required, just simple KEY = value lines:
SECRET_KEY = '........................'
STORAGE = 'filesystem'
STORAGE_FILESYSTEM_DIRECTORY = '/srv/bepasty/storage/'
# ...
```

Important notes:

- if you copied the file from the `bepasty/config.py` it will have a "class Config" in it and all the settings are inside that class. This is **not** what you need. Due to how flask config files work, you need to remove the class statement and outdent all the settings, so you just have global KEY = VALUE statements left on the top level of the config file.

- if you run over http (like for trying it locally / for development), you need to change the configuration to use SESSION_SECURE_COOKIE = False (otherwise you can not login as it won't transmit the cookie over unsecure http).

Starting bepasty server

You can run the bepasty server with your local configuration by pointing to it via the BEPASTY_CONFIG environment variable like this:

```
BEPASTY_CONFIG=/srv/bepasty/bepasty.conf bepasty-server
```

Important note:

- Use an absolute path as value for BEPASTY_CONFIG.

The builtin WSGI server is recommended only for development and non-production use.

For production, you should use a WSGI server like gunicorn, apache+mod-wsgi, nginx+uwsgi, etc.

```
gunicorn bepasty.wsgi
```

Invoking CLI commands

All bepasty commands expect either a –config <configfilename> argument or that the BEPASTY_CONFIG environment variable points to your configuration file.

The "object" command operates on objects stored in the storage. You can get infos about them ("info" subcommand), you can set some flags on them ("set"), you can remove all or some ("purge"), you can check the consistency ("consistency"), etc...

To get help about the object command, use:

```
bepasty-object --help
```

To get help about the object purge subcommand, use:

```
bepasty-object purge --help
```

To run the object purge subcommand (here: dry-run == do not remove anything, files >= 10MiB AND age >= 14 days), use something like:

```
bepasty-object purge --dry-run --size 10 --age 14 '*'
```

If you upgraded bepasty, you might need to upgrade the stored metadata to the current bepasty metadata schema:

```
bepasty-object migrate '*'
```

Note: the '*' needs to be quoted with single-quotes so the shell does not expand it. it tells the command to operate on all names in the storage (you could also give some specific names instead of '*').

Installation tutorial with Debian, NGinx and gunicorn

preliminary packages:

```
apt-get install build-essential nginx supervisor python-dev git-core python-pip␣
↪python-virtualenv
```

commands to run

```
# add user bepasty to system
adduser bepasty
# change to user bepasty
sudo su - bepasty
# clone repository from github
git clone https://github.com/bepasty/bepasty-server.git src
# create folder for storage
mkdir storage
# create folder for logs
mkdir logs
# create virtualenv
virtualenv .
# activate virtualenv
. bin/activate
cd src
# install python requirements for bepasty
pip install -r requirements.txt
# add gunicorn and gevent for hosting
pip install gunicorn gevent
```

config file for bepasty – `/home/bepasty/bepasty.conf`:

Copy `bepasty/config.py` to `/home/bepasty/bepasty.conf` first, remove the `class Config` and remove all indents in the file. The comments can be removed too, if you feel the need to. At last modify these two configs variables:

```
STORAGE = 'filesystem'
STORAGE_FILESYSTEM_DIRECTORY = '/home/bepasty/storage/'
```

add this content to `/home/bepasty/bin/gunicorn_bepasty`:

```
#!/bin/bash

NAME="bepasty"
HOME=/home/bepasty
SOCKFILE=$HOME/gunicorn.sock  # we will communicate using this unix socket
PIDFILE=$HOME/gunicorn.pid
NUM_WORKERS=3                 # how many worker processes should Gunicorn spawn
export BEPASTY_CONFIG=$HOME/bepasty.conf

source $HOME/bin/activate

cd $HOME/src

exec gunicorn bepasty.wsgi \
  --name $NAME \
  --workers $NUM_WORKERS \
  --log-level=info \
  --bind=unix:$SOCKFILE \
  --pid $PIDFILE \
  -k gevent
```

Make it executable: `chmod +x ~/bin/gunicorn_bepasty`

A nginx configuration i.e. in `/etc/nginx/conf.d/bepasty.conf`:

```
upstream pasty_server {
  server unix:/home/bepasty/gunicorn.sock fail_timeout=0;
}

server {
  listen 80;
  #listen [::]:80; #uncomment this if your server supports IPv6
  server_name paste.example.org;  # <-- add your domainname here

  access_log /home/bepasty/logs/nginx-access.log;
  error_log /home/bepasty/logs/nginx-error.log;

  client_max_body_size 32M;

  location / {
      proxy_set_header Host $http_host;
      proxy_pass http://pasty_server;
  }

  location /static/ {
      alias /home/bepasty/src/bepasty/static/;
  }
}
```

Now reload your nginx configuration: *service nginx reload*.

Supervisord config i.e. in `/etc/supervisor/conf.d/bepasty.conf`:

```
[program:bepasty]
command = /home/bepasty/bin/gunicorn_bepasty                    ; Command to start app
user = bepasty                                                  ; User to run as
stdout_logfile = /home/bepasty/logs/gunicorn_supervisor.log     ; Where to write log␣
↪messages
redirect_stderr = true                                          ; Save stderr in the␣
↪same log
```

Finally reload supervisor: *service supervisor reload*

ChangeLog

Release 0.5.0 (not released yet)

New features:

- run bepasty at non-root URLs, see APP_BASE_PATH in the config.

Release 0.4.0

New features:

- shorter, easy-to-read URLs / filenames (old uuid4 style URLs still supported, but not generated any more for new items)
- "list" permission separated from "admin" permission.

  - list: be able to list (discover) all pastebins
  - admin: be able to lock/unlock files, do actions even if upload is not completed or item is locked

  Make sure you update your PERMISSIONS configuration (you likely want to give "list" to the site administrator).

  By giving "list" (and/or "delete") permission to more users, you could operate your bepasty site in a rather public way (users seeing stuff from other users, maybe even being able to delete stuff they see).

Fixes:

- give configured limits to JS also, so stuff has not to be kept in sync manually, fixes #109
- highlighted text file views: set fixed width to line number column, fixes #108
- fixed crash for inline and download views when item was already deleted

Other changes:

- support Python 3.3+ additionally to 2.6+
- improved documentation, esp. about REST api
- improve sample configs

Release 0.3.0

New features:

- support http basic auth header (it just reads the password from there, the user name is ignored). this is useful for scripting, e.g. you can do now: $ curl -F 'file=@somefile;type=text/plain' http://user:password@localhost:5000/+upload
- you can give the filename for the list items now
- do not use paste.txt as default filename, but <uuid>.txt or <uuid>.bin (this is less pretty, but avoids collisions if you download multiple files)
- allow uploading of multiple files via the fileselector of the browser
- display download (view) timestamp
- sorting of file lists
- use iso-8859-1 if decoding with utf-8 fails
- let admin directly delete locked files, without having to unlock first
- new bepasty-object cli command
- added REST api for bepasty-client-cli
- MAX_RENDER_SIZE can be used to set up maximum sizes for items of misc. types, so bepasty e.g. won't try to render a 1 GB text file with highlighting.
- offer a "max. lifetime" when creating a pastebin
- if you link to some specific text line, it will highlight that line now
- add filename to the pastebin url (as anchor)

Removed features:

- removed ceph-storage implementation due to bugs, missing features and general lack of maintenance. it is still in the repo in branch ceph-storage, waiting to be merged back after these issues have been fixed: https://github.com/bepasty/bepasty-server/issues/13 https://github.com/bepasty/bepasty-server/issues/38

Fixes:

- security fix: when showing potentially dangerous text/* types, force the content-type to be text/plain and also turn the browser's sniffer off.
- security fix: prevent disclosure of locked item's metadata
- use POST for delete/lock actions
- application/x-pdf content-type items are offer for in-browser rendering, too
- fix typo in cli command bepasty-object set –incomplete (not: uncomplete)
- quite some UI / UX and other bug fixes
- filesystem storage: check if the configured directory is actually writeable

Other changes:

- using xstatic packages now for all 3rd party static files
- docs updated / enhanced

No release 0.2.0

We made quite quick progress due to many contributions from EuroPython 2014 sprint participants, so there was no 0.2.0 release and we directly jumped to 0.3.0.

Release 0.1.0

New features:

- add a textarea so one now actually can paste (not just upload)
- simple login/logout and permissions system - see PERMISSIONS in config.py.
- add lock/unlock functionality to web UI (admin)
- add "List all items" on web UI (admin)
- add link to online documentation
- support inline viewing of PDFs
- support Python 2.6
- after upload of multiple files, offer creation of list item
- file uploads can be aborted (partially uploaded file will get deleted)
- store upload timestamp into metadata
- Compute hash of chunked uploads in a background thread directly after upload has finished.
- new migrate cli subcommand to upgrade stored metadata (see –help for details)
- new purge cli subcommand (see –help for details). you can use this to purge by age (since upload), inactivity (since last download), size or (mime)type. BEWARE: giving no criteria (like age, size, ...) means: purge all. Giving multiple criteria means they all must apply for files to get purged (AND - if you need OR, just run the command multiple times).
- new consistency cli subcommand (see –help for details). you can check consistency of hash/size in metadata against what you have in storage. Optionally, you can compute hashes (if an empty hash was stored) or fix the metadata with the computed hash/size values. also, you can remove files with inconsistent hash / size.

Fixes:

- for chunked upload, a wrong hash was computed. Fixed.
- misc. cosmetic UI fixes / misc. minor bug fixes
- add project docs
- use monospace font for textarea
- now correctly positions to linenumber anchors

Release 0.0.1

- first pypi release. release early, release often! :)

The bepasty software Project

History

The initial version of the bepasty(-server) software was developed in 48h in the WSGI Wrestle 2013 contest by:

- Bastian Blank
- Christian Fischer

Project site

Source code repository, issue tracker (bugs, ideas about enhancements, todo, feedback, ...), link to documentation is all there:

https://github.com/bepasty/

Contributing

Feedback is welcome.

If you find some issue, have some idea or some patch, please submit them via the issue tracker.

Or even better: if you use git, fork our repo, make your changes and submit a pull request.

For small fixes, you can even just edit the files on github (github will then fork, change and submit a pull request automatically).

License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Authors

Bastian Blank <bastian@waldi.eu.org>

Christian Fischer <cfischer@nuspace.de>

Thomas Waldmann <tw@waldmann-edv.de>

Dennis Schmalacker <github@progde.de>

Ana Balica <ana.balica@gmail.com>

Daniel Gonzalez <gonvaled@gonvaled.com>

Valentin Pratz <vp.pratz@yahoo.de>

Darko Ronic <darko.ronic@gmail.com>

(add your name/email above this line if you contributed to this project)

Index