tzk

Release 0.3.0

Soren Bjornstad

Aug 19, 2022

CONTENTS

tzk (pronounced /t'zik/, tuh-ZEEK) is a build tool and utility CLI for Soren Bjornstad's Zettelkasten edition of TiddlyWiki.

CHAPTER

ONE

INSTALLING TZK

1.1 tzk itself

On most systems, tzk may be installed directly from pip at a command line:

```
$ pip install tzk
```

If you don't have Python 3.6 or greater on your computer, you'll need to install it first. If you aren't sure how to do that, here's an exhaustive guide to installing Python on all major operating systems. Once you've gotten Python installed, you'll be able to use `pip` to install tzk as described above.

To check your work, run `tzk --version`; you should see a version number rather than an error, something like:

```
$ tzk --version
1.0.0
```

1.2 Dependencies

In order to set up your Zettelkasten, you'll also need `npm` and `git`. You can check if they're installed like this:

```
$ npm --version
7.20.6
$ git --version
git version 2.32.0
```

Your versions will likely be a little different by the time you read this. As long as you get a version number rather than an error, you're good; tzk does not use any features of either tool that require bleeding-edge versions.

If you don't have **NPM**, follow step 1 of the Node.js installation instructions in the TiddlyWiki documentation. You can skip all the remaining steps – tzk takes care of that part for you.

If you don't have **Git**, follow the steps in the Installing Git section of Pro Git.

1.3 Checking your work

Run `tzk preflight` to double-check that everything is correctly installed.

CHAPTER

TWO

CONFIGURING TZK

2.1 Basic setup

1. Change into a directory you'd like to use as your Zettelkasten repository. The directory should be empty, so you'll probably want to create a new one, e.g.:

```
$ mkdir my_zettelkasten
$ cd my_zettelkasten
```

2. Run `tzk init`. This will create a tzk configuration file, install TiddlyWiki to this folder, and set up a Git repository.
3. When `init` has completed successfully, open the `tzk_config.py` in your favorite text editor. Read the comments and make any changes you would like. See the *Builders* section of this documentation for more information about builders – but you'll most likely want to get started with your wiki now and worry about builds once you actually have some content to build!
4. Run `tzk listen` and confirm that you can access your wiki.

2.2 Committing

Many people find that carefully designing atomic Git commits when editing a TiddlyWiki is difficult and not all that useful, so the `tzk commit` command is made available to quickly stage, commit, and (if you wish) push all changes in the repository in one go.

If you want to push your changes to some remote location, such as a GitHub repository, add a new Git remote (e.g., `git remote add origin https://github.com/you/YourRepository`) and set the `commit_remote` option in your tzk config to the remote name (here, `origin`). You can selectively skip pushing for a particular commit with the `--local` switch to `tzk commit`.

Note: If you want to push a wiki that contains only some of your content to GitHub in a form that others can browse, don't try to set it up here – use a *`publish_wiki_to_github()`* builder at the end of the `public` build product. See *Builders* for more information.

2.3 Environment

If you'd like to be able to run `tzk` from any directory, rather than having to change into the directory of your tzk repository, set the `TZK_DIRECTORY` environment variable on your system to its full path. If the current directory contains a `tzk_config.py` file, the current directory will still be preferred to the `TZK_DIRECTORY` directory.

Note: `TZK_DIRECTORY` is not honored when calling `tzk init`. Otherwise tzk would prioritize the `TZK_DIRECTORY` over the current directory since the current directory doesn't contain a config file yet, and it would be impossible to initialize a second tzk repository.

CHAPTER

THREE

BUILDERS

Builders are small executable chunks that can be linked together to form a useful build process. Products are built by applying builders in sequence. Please see the existing `products` dictionary and associated comments in the *config file* for how these are specified.

3.1 Builders included with tzk

You can use the following builders in your configuration file out of the box:

`tzk.builders.`**`check_for_kill_phrases`**(*kill_phrase_file: str = None*) → None

Fail the build if any of a series of regexes matches a tiddler's source in the temp wiki.

The temp wiki should be created first using the *`export_public_tiddlers()`* builder.

The kill phrases are Python-format regular expressions and may be configured within the wiki, currently in $:/sib/gui/KillPhrases.

Parameters **`kill_phrase_file`** – The path from the source wiki's root directory to the config tiddler containing kill phrases. In the default Zettelkasten edition, this is "tiddlers/$__config_zettelkasten_Build_KillPhrases.tid"; if you change the way paths are determined, you can give a different path here.

`tzk.builders.`**`compile_html_file`**(*wiki_name: str = 'index.html', output_folder: str = 'output/public_site/', overwrite: bool = True, externalize_attachments: bool = False, attachment_filter: str = '[is[image]]', canonical_uri_template: str = '$:/core/templates/canonical-uri-external-image'*) → None

Compile a single HTML file from the temp wiki.

Before compiling an HTML file, you should create a temp wiki using the *`export_public_tiddlers()`* builder, then run any other build steps you want to use to make changes to the wiki being built. Once you compile the HTML file, it will be copied out to an output location outside the temp folder, and the content can no longer be changed through tzk.

Parameters

- **`wiki_name`** – The filename of the single-file wiki to create. Default `index.html`.
- **`output_folder`** – The path to the folder where the single-file wiki and any externalized attachments will be placed, relative to the private wiki's root directory. Default `output/public_site`.
- **`overwrite`** – If the `output_folder` already exists, should we overwrite any files in it with the same name? Default True.

- **externalize_attachments** – If True, update tiddlers that match the `attachment_filter` parameter to point to external versions of their content. Only useful if you have previously run the `save_attachments_externally` builder. Default False.
- **attachment_filter** – If externalizing attachments, which tiddlers should be externalized? Default `[is[image]]`.
- **canonical_uri_template** – What TiddlyWiki template should be used to determine the new content of the `_canonical_uri` field? Default `$:/core/templates/canonical-uri-external-image`. If you're not sure what this is, don't change it.

`tzk.builders.`**delete_tiddlers**(*tiddlers: Sequence[str]*) → None
Delete selected tiddlers from the output.

This is hopefully self-explanatory.

Parameters **tiddlers** – A list of filenames of tiddlers to delete.

`tzk.builders.`**editionify**(*target_folder: str*, *description: str*) → None
Copy the output folder to a target location and set its edition description.

This generates a TiddlyWiki edition based on the temporary output. By copying it into an appropriate location or setting the environment variable `TIDDLYWIKI_EDITION_PATH` to the parent directory of the edition's folder, it's possible to quickly generate new TiddlyWikis based on the edition "template" (`tiddlywiki --init EDITION_NAME`, where the `EDITION_NAME` is the name of the folder).

Parameters

- **target_folder** – The folder to copy the output folder to. This folder *will be deleted* if it already exists prior to the copy.
- **description** – The description of this edition to use in the new edition's `tiddlywiki.info` file.

`tzk.builders.`**export_public_tiddlers**(*export_filter: str*) → None
Export specified tiddlers to a new wiki in the temporary build folder.

new_output_folder() must be run prior to this builder.

Parameters **export_filter** – A TiddlyWiki filter describing the tiddlers to be selected for inclusion in the new wiki.

`tzk.builders.`**new_output_folder**()
Create a new temporary folder to hold intermediate steps of the product being built.

The path to this temporary folder will be stored in the `public_wiki_folder` key of the `builders.build_state` dictionary. Future build steps can access the work in progress here. A cleaner is registered to delete this folder when all steps complete, so any finished product should be copied out by a later build step once it is complete.

`tzk.builders.`**publish_wiki_to_github**(*output_folder: str = 'output/public_site/'*, *commit_message: str = 'publish checkpoint'*, *remote: str = 'origin'*, *refspec: str = 'master'*, *push=True*) → None
Publish the built wiki to GitHub.

Parameters

- **output_folder** – Path to a folder containing the Git repository you'd like to publish, relative to your private wiki's root directory. This folder should contain the version of your

wiki built by the *compile_html_file()* builder, either directly or in a subfolder somewhere. You need to clone or create the repository yourself and set up the remotes as appropriate before running this step. Default `output/public_site`.

- **commit_message** – Message to use when committing the newly built wiki. Note that all changes in the repository will be committed. Default `publish checkpoint`.
- **remote** – The repository remote to push changes to. If you don't know what that is, the default of `origin` is probably correct.
- **refspec** – The local branch or refspec to push. Default `master`.
- **push** – If set to False, don't push after committing the changes (mostly useful for testing purposes). Default True.

tzk.builders.**replace_private_people**(*initialer: Callable[[str], str] = None*, *replace_link_text: bool = False*) → None

Replace the names of people who are not marked Public with their initials.

If you have lots of PAO (People, Animals, and Organizations) in your Zettelkasten and many of them are personal friends, you might prefer not to make everything you said about them easily searchable on the internet. This is more challenging than simply not marking their tiddlers public, since there will also probably be links to their tiddlers in other tiddlers – and those links contain their full names, if you put them in the title.

This builder replaces all links, bracketed or WikiCamelCase, to the names of all people *not* tagged Public with the initials suggested by their CamelCase titles (e.g., MsJaneDoe becomes J.D.). The links point to the tiddler `PrivatePerson`, which explains this process.

Parameters

- **initialer** – If you don't like the way that initials are generated from tiddler filenames by default, you can customize it by passing a callable that takes one string argument (a tiddler filename without the full path, e.g., `MsJaneDoe.tid`) and returns a string to be considered the "initials" of that person.
- **replace_link_text** – If you have links in the form `So then [[John said|MrJohnDoe]] something about this`, then enabling this option ensures that the link is fully replaced with `So then [[J.D.|PrivatePerson]] something about this`. This means that when using this feature, having the link text also be meaningful after redaction is important.

Warning: Using this link replacement feature does not redact everything, just the link (and the link text with *replace_link_text* enabled). So *do not* rely on it for redacting everything. Making a tiddler public still needs consideration and tooling is there to help, not to replace your own judgment.

tzk.builders.**require_branch**(*branchname: str*) → None

Require a specific Git branch to be checked out.

If the branch isn't checked out, the build will fail immediately. This may be helpful if you want to be sure you aren't accidentally building a wiki from tentative changes or an old version.

Parameters **branchname** – The name of the branch that must be checked out.

tzk.builders.**require_clean_working_tree**() → None

Require the working tree of the Git repository to be clean.

If there are any unstaged changes to existing files or staged changes, the build will fail immediately.

For the standard build process, it is not necessary for the working tree to be clean. However, if you use any custom build steps that compare history, or you simply want to ensure that you always have a recent checkpoint

in your local version whenever you publish another version, this may be a useful requirement.

tzk.builders.**save_attachments_externally**(*attachment_filter: str = '[is[image]]'*, *extimage_folder: str = 'extimage'*) → None
Save embedded files in the temp wiki into an external folder.

The temp wiki should be created first using the *export_public_tiddlers()* builder.

Note that this builder **does not finish externalizing images**. It saves the images outside the wiki, but it does not change the `_canonical_uri` and `text` fields on each image tiddler to point to this new location. For the latter step, use the `externalize_attachments`, `attachment_filter`, and `canonical_uri_template` parameters to the *compile_html_file()* step.

Parameters

- **attachment_filter** – The tiddlers to be saved to the external folder; by default, `[is[image]]`.
- **extimage_folder** – The name of the external folder to save to. This must be the default of `extimage` to work with the default `canonical_uri_template` in the *compile_html_file()* builder, but you can use a different name and a different canonical URI template if you prefer.

tzk.builders.**say_hi**(*username: str*) → None
Say hi to the specified user.

This function is intended for testing. It will normally succeed and print the provided *username*, but if you give "Jeff" as the username, the step will fail, and if you give "General Failure" as the username, it will cause an unhandled exception.

Parameters **username** – The name of the person to say hi to.

tzk.builders.**set_tiddler_values**(*text: Optional[Dict[str, str]] = None*, ***kwargs: Dict[str, str]*) → None
Set fields of selected config or other tiddlers to arbitrary new values.

This can be used to make customizations that can't easily be done with feature flags or other wikitext solutions within the wiki – for instance, changing the subtitle or what buttons are visible. It's also used to implement feature flags in the first place by changing the `$:/config/sib/CurrentEditionPublicity` tiddler to `public`, so at minimum, the build of a public wiki should use:

```
builders.set_tiddler_values({
    '$__config_sib_CurrentEditionPublicity.tid': 'public',
})
```

Any number of arguments can be provided. The name of each argument is the name of the field to edit. One positional argument may be used with no name; this argument is assumed to be mappings for replacing the 'text' field.

tzk.builders.**shell**(*shell_command: str*) → None
Run an arbitrary shell command.

The builder will fail if a return code other than 0 is returned, otherwise it will succeed. Output will be printed to stdout.

Parameters **shell_command** – A string to be passed to your system's shell.

3.2 Builder helper functions

These helper functions, also defined in `tzk.builders`, are intended for use with any custom builders you create.

`tzk.builders.`**`info`**(*message: str*) → None
: Print information about this build step to the console.

`tzk.builders.`**`stop`**(*message: str*) → None
: Stop the build due to an error condition.

`@tzk.builders.`**`tzk_builder`**
: Decorator which makes a function lazy-evaluable: that is, when it's initially called, it returns a zero-argument lambda with the arguments initially passed wrapped up in it. Calling that lambda has the effect of executing the builder.

  We use this in tzk to allow the user to use function calls in her config to define the build steps, while not requiring her to write a bunch of ugly and confusing lambda:'s in the config. The functions that will be called are prepared during the config and executed later.

3.3 Custom builders

If the existing builders don't cover something you're hoping to do to build a product, you can write your own builder as a Python function directly within your config file.

As an example, let's suppose that we want to publish our wiki to an S3 bucket fronted by CloudFront on Amazon Web Services. We can work with AWS using the `aws` CLI, if we've set that up on our computer (we could also use the `boto3` Python library, which is cleaner but slightly longer and requires us to muck around with `pip`, so we'll do it through the CLI in this example). We first write a builder function decorated with *`builders.tzk_builder()`* in our `tzk_config.py`, anywhere above the `products` dictionary:

```
from pathlib import Path
import subprocess

@builders.tzk_builder
def publish_to_aws(target_uri: str, cloudfront_distribution_id: str):
    "Publish output to Amazon S3"
    source_folder = Path(builders.build_state['public_wiki_folder']) / "output"
    # Sync the output folder to S3, deleting any files that have been removed.
    subprocess.call(("aws", "s3", "sync", "--delete", source_folder, target_uri))
    # Clear the CDN cache so the new version is available immediately.
    subprocess.call(("aws", "cloudfront", "create-invalidation",
                     "--distribution-id", cloudfront_distribution_id, "--paths", "/*"))
```

`builders.build_state` is a dictionary that is preserved across all build steps. The *`new_output_folder()`* builder populates this `public_wiki_folder` attribute early in the default build process, so that it contains the path to the temporary wiki that build steps happen within.

The first line of the docstring, in this case `"Publish output to Amazon S3"`, is used as the description of the step in output, with any period at the end removed.

Then we add a call to this builder within the list of steps for this product, with whatever parameters we like:

```
products = {
    'public': [
        [...]
        builders.compile_html_file(externalize_attachments=True),
```

(continues on next page)

(continued from previous page)

```
        publish_to_aws("s3://my_target_uri", "MY_DISTRIBUTION_ID"),
    ],
}
```

Since we've parameterized this builder, we can easily use it multiple times if we want, for instance within different products. Note that we say just `publish_to_aws`, not `builders.publish_to_aws`, since this builder is located directly within the config file rather than in the external `tzk.builders` module that comes with tzk.

Tip: If you do something in a builder that needs to be cleaned up later, like creating a temporary file, assign a cleanup function to the `cleaner` attribute of your builder:

```
def aws_cleanup():
    # nothing really needs to be cleaned up, but if it did we'd do it here
    pass
publish_to_aws.cleaner = aws_cleanup
```

Cleanup functions will run after all steps are done, regardless of whether the build succeeds or fails.

3.4 Shell commands

If a custom builder seems like overkill or you're not familiar with Python, you can also run simple shell commands using the *`shell()`* builder.

Our AWS example would look like this:

```
products = {
    'public': [
        [...]
        builders.compile_html_file(externalize_attachments=True,
                                   output_folder="output/public_site/"),
        builders.shell("aws s3 sync --delete output/public_site s3://my_target_uri"),
        builders.shell("aws cloudfront create-invalidation --distribution-id MY_
↪DISTRIBUTION_ID --paths '/*'"),
    ],
}
```

Notice the need to include quotes within the string in *`shell()`*; the same quoting rules as when running shell commands directly apply. Also notice that we had to access the compiled HTML file from `output/public_site`, since we can no longer refer to the `build_state` dictionary to learn where the temporary output folder is. Paths are relative to the private wiki's root directory (the directory containing the `tiddlywiki.info` file) while builders are running.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

INDEX