# Ginpar

***Release v0.8.3***

**Nov 08, 2019**

# Contents

Ginpar is a static content generator for interactive P5.js sketches, awkwardly named after **Generative Interactive Parametrisable Canvases**.

By separating the primary development and the parametric experimentation, it allows you to stop thinking about code once your pieces have reached an acceptable level of complexity, freeing you to experiment in a GUI for the browser.

Features:

- Simple API to define the controllable variables in your sketch.
- Easy to adapt existing sketches.
- Easy replicability of each final result.
- Index page to list all your sketches.

CHAPTER 1

# Documentation

If you want to start using Ginpar to build your own website, this links will guide you.

We recommend starting with the introduction and then move up to the API docs.

## 1.1 Introduction

This is a quick introductory documentation for the Ginpar static content generator.

Ginpar works similarly to other engines such as Jekyll, Hugo, or Pelican, but with a narrower and deeper set of functionalities, since it's very specific in its task.

The two main objectives of Ginpar are:

- Allowing artists to stop thinking about code when experimenting with the parameters that control the results of the artwork, achieving a **quicker feedback loop.**
- Making interactive websites to share the artist's work, letting users play with the same GUI.

The basic structure of a Ginpar project consists of a `config.yaml` file, and a `sketches` directory.

It's easy to adapt your existing sketches to work with Ginpar. In fact, the only **necessary** step is to add `.parent("artwork-container")` to the `createCanvas()` call.

But to fully take advantage of Ginpar —that is, the interactive sketch with custom parameters—, you need to create a `data.yaml` file. Check *Adapting existing sketches* and *Specifying the parameters*.

**To be fully sure of how easy is to use Ginpar, check this** example **together with its** source code. You can ignore `requirements.txt`, `runtime.txt`, `netlify.toml`, `README.md`, and `.gitignore`.

### 1.1.1 Prerequisites

For now Ginpar only runs using Python >= 3.6. Future versions will add compatibility with Python 2.

### 1.1.2 Installation

The easiest way to install the latest version of Ginpar is using pip:

```
$ pip install ginpar
```

To make sure it's installed and working, run:

```
$ ginpar --version
```

### 1.1.3 Quickstart

Ginpar has an example project ready for installation, it contains the default project structure and a sketch example.

If you're new to static content generators, this may be the best way to start.

```
$ ginpar quickstart
```

This will create the following directory structure:

```
.
├── config.yaml
└── sketches/
    ├── circles/
    │   ├── sketch.js
    │   └── data.yaml
    └── ...
```

To build the project and start a server, run:

```
$ cd quickstart
$ ginpar serve
```

And the project will be live in localhost:8080

Now, you can start to modify the contents of `config.yaml` and `sketches/`.

Next, you should read *Creating new sketches*, or *Serving & Building*.

### 1.1.4 Initialization

Alternatively, if you want to start a new project without importing anything extra, run:

```
$ ginpar init
```

This will prompt you for the values to build your configuration file and then create the project using those values.

With this command, you may configure things like the destination and source directories (`public` and `sketches` by default).

Check *ginpar init* or run `ginpar init --help` for more information.

### 1.1.5 Creating new sketches

Ginpar has a handy command to start new projects with some configuration already set:

```
$ ginpar new [SKETCH]
```

This will create a new sketch inside your predefined source directory. You must set the name of the sketch when running the command.

Check *ginpar new* or run `ginpar new --help` for more information.

Now, you must be *specifying the parameters*.

### 1.1.6 Adapting existing sketches

For Ginpar to build the interactive page, you'll need to add some modifications to your sketch code.

#### Adding it to the list of sketches

First, make your sketch detectable by Ginpar:

1. Create a directory `my-sketch/` inside `sketches/`.
2. Copy your existent sketch script inside `my-sketch` and rename it to `sketch.js`.
3. Create a `data.yaml` file.

You should end with a structure like this:

```
.
└── sketches/
    └── my-sketch/
        ├── sketch.js
        └── data.yaml
```

#### Making your sketch compatible with Ginpar

In your `createCanvas` instruction, add `.parent("artwork-container")`.

Now, you must be *specifying the parameters*.

### 1.1.7 Specifying the parameters

Each sketch is a directory that contains two files: `sketch.js` and `data.yaml`. The `data.yaml` file is where the parameters specification takes place.

To create a parameters list, add this to your data file:

```yaml
---
date: 2019-11-04
# ... other data
# ...

# Key that contains a list of parameters
params:

  # The name of the parameter must be the key of the element
  # It must match a variable in your sketch.js file
  - MY_VARIABLE:

      # Ginpar parameters definition keys. All optional.
```

(continues on next page)

(continued from previous page)

```
    # For a full list check the API
    randomizable: True
    name: My displayed variable name

    # HTML valid attributes
    attrs:
      type: number
      value: 30
      step: 1
      min: 0
      max: 100
```

Once parsed, Ginpar will produce:

- **A form containing each of the items in the `parameters` list:**

```
<form>
  <div class="form-field">
      <label for="my-variable">
          My displayed variable name
      </label>
      <input name="my-variable"
             id="my-variable"
             type="number"
             value="30"
             step="1">
    </div>
  <!-- More form-fields. One for each params element. --->
</form>
```

- **A JS code fragment to update each of the parameters using the form values:**

```
function updateVars() {
  MY_VARIABLE = document.getElementById("my-variable").value;
  // More variable updates. One for each params element.
}
```

If the type of the input is a `number`, Ginpar will parse it before assigning it to the variable.

---

To use this parameters inside your sketch, just use the same name you used as key:

```
console.log(MY_VARIABLE)
// ==> 30
```

### 1.1.8 Serving & Building

Ginpar has two different commands to build your site:

```
$ ginpar build
```

Will build your site into the `build_directory` path, which by default is `public`.

```
$ ginpar serve
```

Will build your site and start a new server on `localhost:8080`. You can specify the port with `--port`.

Check *ginpar serve* and *ginpar build*, or run `ginpar serve --help`, `ginpar build --help` to see the full list of options and arguments available.

## 1.2 Guide

If you want to take full advantage of the Ginpar engine, you may want to read more than just the *Introduction*.

These guides may come handy if you want to modify Gart or develop your own theme and generation flow.

### 1.2.1 Site configuration

This is the API documentation for the configuration file of a Ginpar site: `config.yaml`.

This file contains all the metadata for your site, such as your name, the name of the site, the repository, social links, etc.

The following data fields are the ones used by Ginpar. However, you can design a custom theme or template that makes uses of extra fields.

#### author

*String, optional* [**Not defined by default**]

The name of the author of the site. This will be used to set Copyright messages and meta tags.

#### sitename

*String, optional* [**Not defined by default**]

The name of the site. This will be used to set Copyright messages and meta tags.

#### description

*String, optional* [**Not defined by default**]

The description of the site. This will be used as a meta tag, and will appear in the index of the site.

#### url

*String, optional* [**Not defined by default**]

The URL you'll be using to redirect to the site's content.

#### theme

*String, required* [**davidomarf/gart**]

If the theme you want to use is a GitHub repository, set this value to `AUTHOR/REPO`.

If it's a git repository in a different server, add the `.git` address.

Alternatively, you can add a string that matches the name of one directory inside the `themes/` folder for a locally designed theme.

The default value is `davidomarf/ginpar`

#### content_path

*String, required* [**sketches**]

The directory that contains the project sketches. This path is referenced when you run *ginpar build* and *ginpar serve*.

#### build_path

*String, required* [**public**]

The directory Ginpar will use to build the site.

#### scripts

*List, Optional* [**Not defined by default**]

This is a list of scripts and the url to fetch them. You can later reference the items of this list to include them in individual sketches.

The structure is this:

```
scripts:
    p5: https://p5-url
    lib: https://my-lib-url
```

#### sketch_defaults

*Object, Optional* [**Not defined by default**]

Here you'll add the same content you'd otherwise manually add to every sketch in your project.

For example, if you'd like all your sketches to **not allow a global_seed**, you'd need to add this to your config.yaml file:

```
sketch_defaults:
    global_seed: False
```

Now, all your sketches will automatically have the value `global_seed: False`. However, you can manually replace that value for a single sketch.

For all the available values, check *Sketch data*.

### 1.2.2 Sketch data

This is the API documentation for the data files of single sketches: `data.yaml`.

Data files contain instructions to build the pages for individual sketches, such as the sketch parameters, source code obfuscation, sketch hashing, etc.

All these values are converted into a Python dictionary and then associated with the sketch. Ginpar will use this dictionary to render `sketch templates`.

Check Jinja to learn how this templating works.

The following keys are the ones that:

- Ginpar uses to determine build flow, or
- Gart (the default theme) uses in its templates to render final pages.

The only indispensable key is `params`. Every other key it's optional.

**If you want to use a configuration value for all your sketches, you may want to read** *sketch_defaults*.

### date

**Date, Required [Date.today()]**

Used to sort the sketches in the index. The format is any valid date format, but the suggested is `YYYY-MM-DD`: 2019-11-04.

The default value is th current day, however, this is only assigned when you create the sketch using `ginpar new [SKETCH]`.

### params

**List, Required**

This is the most important and the only required key for the data file. In `params`, you specify the sketch parameters and their attributes.

The key of every element **must match the variable name in your sketch.js**.

```
params:
    - YEAR:
        attrs:
            type: number
            value: 2019
            step: 1
    - RATIO:
        attrs:
            type: number
            value: 0.2
            step: 0.05
            min: 0
            max: 1
```

```
console.log(NAME, YEAR, RATIO)
// ==> "Ginpar", 2019, 0.2
```

For most of the variables, those attributes will suffice.

Ginpar will automatically remove low dashes and capitalize the parameter name, however, you can also specify the name to display in the form:

```
params:
    - YEAR:
        name: Current year
        attrs:
            # ...
```

For a complete list of the fields you can specify for the `params` list, check *Params API*.

#### global_seed

*Boolean, Optional* [**True**]

When **True**, Ginpar will add a button to generate new seeds, and will create a file name for the saved image using `{NAME}-{RANDOM_SEED}-{NOISE_SEED}`.

#### scripts

*List, Optional* [**site.scripts**]

By default, Ginpar will include all the scripts you specify in the `config.yaml`. If you only wona to include a subset of these, you create a list of the scripts to include.

```
# in config.yaml
scripts:
    p5: https://my-p5-url
    d3: https://my-d3-url
    extra: https://extra

# in data.yaml
scripts:
    - p5
    - d3
```

The elements of *data.scripts* must exist as keys in your `config.yaml` file.

### 1.2.3 Params API

This is the API documentation for the `params` field inside a single sketch data file `data.yaml`. Check *Sketch data*

Data files contain instructions to build the pages for individual sketches, such as the sketch parameters, source code obfuscation, sketch hashing, etc.

All these values are converted into a Python dictionary and then associated with the sketch. Ginpar will use this dictionary to render `sketch templates`.

Ginpar will automatically generate a `name` when processing the parameter info. This value is generated by removing *snake_case* from the string and capitalizing:

```
MY_VERY_VERBOSE_VARIABLE ==> My very verbose variable
```

However, if you'd like to use a different name instead of the generated one, you can specify it in its values.

#### name

*Boolean, Optional* [**Not defined by default**]

Ginpar will automatically generate a `name` when processing the parameter info. This value is generated by removing *snake_case* from the string and capitalizing:

```
MY_VERY_VERBOSE_VARIABLE ==> My very verbose variable
```

However, if you'd like to use a different name instead of the generated one, you can specify it in its values.

#### attrs

*List, Required*

All the *key-value* pairs inside the `attrs` field will be added to the input tag as attributes.

In `attrs` you can (and should) specify all the input tag attributes you'd like to include in your input.

The only required field is `value`, however, Ginpar will produce a better output if you specify `type`, `value`, `step`, `min`, `max`.

## 1.3 CLI Commands

This is a list of the available CLI commands for Ginpar.

This page contains the same information you'd get if you run

```
$ ginpar --help
```

and then,

```
$ ginpar COMMAND --help
```

for every command available.

### 1.3.1 ginpar

Ginpar is a static content generator for interactive P5.js sketches, awkwardly named after Generative Interactive Parametrisable Canvases.

```
ginpar [OPTIONS] COMMAND [ARGS]...
```

#### Options

**--version**
Show the version and exit.

#### Commands

**build**
Build the project content into PATH.

**init**
Initialize a new project in PATH.

**new**
Create a new SKETCH.

**quickstart**
Import a working sample project.

**serve**
Start a new server in localhost:PORT.

### 1.3.2 ginpar build

Build the project content into PATH.

*ginpar build* will read your configuration file, fetch all the sketches inside your <config.content_path>, and build your static site inside PATH, which defaults to <config.build_path>, or public if it doesn't exist.

This operation will wipe all the content from PATH in each run, so you must not make modifications you expect to preserve.

```
ginpar build [OPTIONS]
```

#### Options

**-p, --path** <path>
The PATH where the site will be built. [ <config.build_path>, public ] This path is relative to the current directory. When no option is provided Ginpar will read the <config.build_path> from the configuration file.

### 1.3.3 ginpar init

Initialize a new project in PATH.

*ginpar init* will prompt you for a series of values that will be used to generate the configuration and file structure of your project.

```
ginpar init [OPTIONS]
```

#### Options

**-f, --force**
If Ginpar finds an existing directory with the same name of the project being initialized, it'll force its removal. Only do this if you're completely sure you want to do it.

**-q, --quick**
Skip the prompts and use the default values for the configuration file. You can still modify the variables later by manually updating your configuration file.

### 1.3.4 ginpar new

Create a new SKETCH.

*ginpar new* will create a new sketch structure inside your <config.content_path>.

You must specify the name of the sketch.

If there's an existing sketch with the same name, it'll throw an error and ask for a different name.

```
ginpar new [OPTIONS] SKETCH
```

#### Arguments

**SKETCH**
Required argument

### 1.3.5 ginpar quickstart

Import a working sample project.

*ginpar quickstart* will download the contents of the sample project, hosted at github: davidomarf/ginpar-quickstart in the current directory.

```
ginpar quickstart [OPTIONS]
```

#### Options

**-f, --force**
If Ginpar finds an existing directory with the same name of the sample content, it'll force its removal. Only do this if you're completely sure you want to do it.

### 1.3.6 ginpar serve

Start a new server in localhost:PORT.

*ginpar serve* will trigger *ginpar build*, and start a new server inside <config.build_path>.

Every time you modify a file that is part of the project's source code the site gets built again.

```
ginpar serve [OPTIONS]
```

#### Options

**-p, --port** <port>
Port of the server

**-w, --watch**
By default, the server will only watch for changes in the source path, but if this flag is received, it'll watch all the project directories.

## 1.4 Developer docs

### 1.4.1 ginpar package

#### Subpackages

#### ginpar.utils package

#### Submodules

#### ginpar.utils.echo module

Echo different categories of messages.

Use either `click.echo` or `click.secho` with predefined custom foreground colors to echo different categories of messages like success, errors, or warnings.

ginpar.utils.echo.**alert**(*m*)
Warnings and alerts that may change Ginpar behavior. Yellow foreground.

ginpar.utils.echo.**echo**(*m*)
General messages with no emphasis. Blue foreground.

ginpar.utils.echo.**error**(*m*)
Failure messages that interrumpted a certain operation. Red foreground.

ginpar.utils.echo.**info**(*m*)
Information messages with tips or suggestions. Blue foreground.

ginpar.utils.echo.**success**(*m*)
Success messages after operations or tasks. Green foreground.

### ginpar.utils.files module

File management for common Ginpar operations.

This module implements the default ways of managing file creation, deletion, copying, and moving.

ginpar.utils.files.**check_existence**(*path*)
Check if the received path exists as either a file or a directory.

**Parameters**

**path** [str] Can be a valid path or not.

ginpar.utils.files.**copy_folder**(*fr*, *to*, *force=False*)
Attempt (and optionally force) the copy of a folder.

**Parameters**

**fr** [str] From. Path of the folder to be copied.

**to** [str] To. Path of the to be created.

**force** [bool] Will remove an existing directory with the same name of `to`, if it exists.

ginpar.utils.files.**create_file**(*file*, *content*)
Attempt the creation of a new file with custom content.

**Parameters**

**file** [str] Name of the file to be created (path must be included).

**content** [str] Content to write into the file.

ginpar.utils.files.**create_folder**(*folder*, *force=False*)
Attempt (and optionally force) the creation of a new folder.

**Parameters**

**folder** [str] Name of the folder to be created (path must be included).

**force** [bool] Will remove an existing directory with the same name, if it exists.

ginpar.utils.files.**try_remove**(*path*)
Attempt the removal of a path, either if it is a file or a directory.

**Parameters**

**path** [str] Must be an existing path.

#### Notes

To check for the existence of the path, previous to calling `try_remove`, use **'check_existence'_**.

### ginpar.utils.git module

Git repository management inside Ginpar for themes and quickstart.

`ginpar.utils.git.`**`clone_repo`**(*repo*, *path*)
Clone the contents of a repository in a custom path

**Parameters**

**repo** [str] GitHub repository as in "{USER}/{REPO}"

**path** [str] Path to clone the repository to

`ginpar.utils.git.`**`delete_git_files`**(*path*)
Delete the git files to only keep the relevant files

**Parameters**

**path** [str] Path to look for git files

### ginpar.utils.strings module

String filters to convert between cases.

The list of filters in here are added to the Jinja2 environment, so they may come handy when designing a custom theme.

`ginpar.utils.strings.`**`camel_to_space`**(*s*)
Replace low dashes with spaces.

**Parameters**

**s** [str] String that may contain "_" characters.

`ginpar.utils.strings.`**`space_to_kebab`**(*s*)
Replace spaces with dashes.

**Parameters**

**s** [str] String that may contain " " characters.

`ginpar.utils.strings.`**`unkebab`**(*s*)
Replace dashes with spaces.

**Parameters**

**s** [str] String that may contain "-" characters.

### Module contents

### Submodules

### ginpar.build module

Build command for Ginpar projects.

This module implements the building command for the ginpar static content generator.

`build` will read the configuration file in search for `build_path` and `source_path`. If not defined, *build* will use `"public"` and `"sketches"`, respectively.

### Examples

To build your project according to your specifications in *config.yaml*:

```
ginpar build
```

To build targeting a custom path *_site/*:

```
ginpar build --path="_site"
```

### Notes

You cannot specify the content path. It is either `config.content_path` or `"sketches"`.

`ginpar.build.`**`build`**(*path*)
Main function of the module. This is what *ginpar build* calls.

**Parameters**

**build_path** [str] Path of the build.

`ginpar.build.`**`convert_information`**(*sketch*)
Convert the *["data"]* field of a sketch into a Python dictionary.

**Parameters**

**sketch** [dict] It contains the sketch information. Must contain *["data"]*, and *["data"]* must be a YAML-valid string.

**Returns**

**dictionary** *sketch* but with the updated *["data"]* field.

`ginpar.build.`**`copy_theme`**(*build_path*, *theme*)
Copy the theme static content into the build static directory.

**Parameters**

**build_path** [str] Path of the build.

**theme** [str] Name of the theme to install.

`ginpar.build.`**`create_publishing_directory`**(*build_path*)
Remove existing directories with the same name, and create again.

**Parameters**

**build_path** [str] Path of the build.

`ginpar.build.`**`dict_to_attrs`**(*d*)
Filter to convert a python dictionary into a HTML attributes.

For each (key, value) pair inside the dict, a `key=value` string will be created.

**Parameters**

**d** [dict] Dictionary containing the key value attributes.

**Returns**

**str** String containing all the attributes of the dictionary separated with spaces.

ginpar.build.**get_sketches**(*content_path*)
Obtain the list of **valid** sketches inside *path*.

Valid sketches are directories containing at least two files: *sketch.js* and *data.yaml*.

This function will create a list of sketch objects containing *name*, *script*, and *data*.

**Parameters**

**content_path** [str] The path containing the sketches to fetch.

**Returns**

**list** Individual elements contain *{"name", "script", "data"}*.

ginpar.build.**param_to_dict**(*param*)
Receive a parameter entry and convert it into a Python dictionary

A parameter entry looks like `[{ VAR : CONTENTS_OF_VAR }]`. This function turns that entry into a dictionary with keys `{var, id, name, attrs}`.

The `attrs` key must exist inside `COTENTS_OF_VAR`.

**Parameters**

**param** [dict] A parameter as specified inside the `data.yaml` file of the sketch.

**Returns**

**dict** A properly formatted dictionary to use inside Ginpar with keys `{var, id, attrs, name}`.

ginpar.build.**read_config**(*path*)
Create a dictionary out of the YAML file received

**Parameters**

**path** [str] Path of the YAML file.

ginpar.build.**render_index**(*build_path*, *sketches*, *site*, *page_template*)
Render the index using the list of sketches and site configuration

The index is rendered using a Jinja2 template inside the theme *templates/* directory.

The index template must receive *sketches*, containing a list of the sketches names; and *site*, containing the site configuration from `config.yaml`.

**Parameters**

**build_path** [str] Path of the build.

**sketches** [list] Contains all the sketches in the project. Must contain at leas *["name"]*.

**site** [dict] Contains the site information, such as *sitename* and *author*.

**page_template** [Jinja2.Template] Jinja2 template to render the sketch.

ginpar.build.**render_sketch_page**(*build_path*, *sketch*, *site*, *page_template*, *input_templates*)
Render a sketch page

This generates the page for a single sketch. This will convert the *sketch["data"]* into a form that will control the variables of the script.

When *sketch["data"]* doesn't define fields that may be used at the moment of the form generation, Ginpar will instead look up for those fields in *site["sketch_defaults"]*.

When both *sketch["data"]* and *site["sketch_defaults"]* don't define those fields, Ginpar will use the default values.

Ginpar default values for sketch data

**Parameters**

**build_path** [str] Path of the build.

**sketch** [dict] Sketch information. Must contain *["data"]* and *["name"]*

**site** [dict] Site configuration.

**page_template** [Jinja2.Template] Jinja2 template to render the sketch.

### ginpar.cli module

Definition of the CLI commands for Ginpar.

This module defines the different commands available for the ginpar static content generator.

### Examples

To get the list of available commands and options run:

```
ginpar
```

### ginpar.generators module

ginpar.generators.**dict_to_attrs**(*d*)

Filter to convert a python dictionary into a HTML attributes.

For each (key, value) pair inside the dict, a `key=value` string will be created.

**Parameters**

**d** [dict] Dictionary containing the key value attributes.

**Returns**

**str** String containing all the attributes of the dictionary separated with spaces.

ginpar.generators.**makeValueGetter**(*global_seed*, *attrs*)

### ginpar.init module

Init command for Ginpar projects.

This module implements the initialization command for the ginpar static content generator.

*init* will prompt for a series of values to write the site configuration file.

### Examples

To initialize a project in a standard way to specify the configuration values:

```
ginpar init
```

To skip the prompts and initialize the project with the default values:

```
ginpar init --quick
ginpar init --q
```

To force the initialization in case there is a directory with the same name of the project to initialize:

```
ginpar init --force
ginpar init -f
```

ginpar.init.**init**(*force*, *quick*)

Main function of the module. This is what *ginpar init* calls.

**Parameters**

**force** [bool] Remove conflicting files when true.

**quick** [bool] Skip prompts when true.

ginpar.init.**prompt_site_config**(*quick*)

Echo the prompts and create the configuration dict.

Echo the instructions and configuration fields, store each input, and create a dictionary containing those values.

**Parameters**

**quick** [bool] Returns the default values immediatle if True.

**Returns**

**dict** Used to generate the site configuration file.

## ginpar.new module

New sketch creation command for Ginpar projects.

This module implements the sketch creation command for the ginpar static content generator.

*new* will read the configuration file in search for *source_path* and create a new directory in there with the specified name, which by default is *new-sketch-{n}*.

This directory will contain the two required files with a boilerplate code.

### Examples

To create a new sketch *rectangle*:

```
ginpar new rectangle
```

To start a new sketch with the default name *new-sketch-{n}*:

```
ginpar new
```

ginpar.new.**new**(*sketch*)
Main function of the module. This is what *ginpar new* calls.

**Parameters**

**sketch** [str] Name of the sketch to create

ginpar.new.**read_config**(*path*)
Create a dictionary out of the YAML file received

**Parameters**

**path** [str] Path of the YAML file.

### ginpar.quickstart module

Quickstart command for Ginpar projects.

This module implements the quickstart command for the ginpar static content generator.

*quickstart* will download the contents of the sample repository hosted at davidomarf/ginpar-quickstart into *./quickstart*.

This is aimed to provide an easier and faster way to start working in a Ginpar project for people who isn't familiar with static content generators.

### Example

To create *./quickstart* and copy the contents of the sample repository:

```
ginpar quickstart
```

If there's another directory named *quickstart*, you can force this command, removing the existing directory:

```
ginpar quickstart --force
ginpar quickstart -f
```

ginpar.quickstart.**quickstart**(*force*)
Main function of the module. This is what *ginpar quickstart* calls.

**Parameters**

**force** [bool] Remove conflicting files when true.

### ginpar.serve module

Serve command for Ginpar projects.

This module implements the server starting command for the ginpar static content generator.

*serve* will start a live-reloading server in a specified port.

### Examples

To start a new server in the default port *8080*:

```
ginpar serve
```

To start a new server in a custom port:

```
ginpar serve --port=3000
ginpar serve -p=3000
```

ginpar.serve.**read_config**(*path*)
Create a dictionary out of the YAML file received

**Parameters**

**path** [str] Path of the YAML file.

ginpar.serve.**serve**(*port*, *watch*)
Main function of the module. This is what *ginpar serve* calls.

**Parameters**

**port** [int] The port of the server

#### Module contents

ginpar.**main**()

### 1.4.2 tests package

#### Submodules

#### tests.test_build module

#### tests.test_init module

#### tests.test_new module

#### tests.test_quickstart module

#### Module contents

CHAPTER 2

# Other links

- genindex
- modindex
- search

# Python Module Index

## g



## t

# Index

## Symbols



## A



## B



## C



## D



## E



## G



## I