Hugo Workshop

Chemnitzer Linux-Tage 2025

Jan Ulrich Hasecke

23.03.2025

Installation

Hugo besteht aus einer ausführbaren Datei, die lediglich in den Ausführungspfad kopiert werden muss. Die aktuelle Version von Hugo findet sich jeweils hier:
`https://github.com/gohugoio/hugo/releases`.

Linux

- `hugo_0.145.0_Linux-64bit.tar.gz`
- `hugo_0.145.0_linux-amd64.deb`
- `hugo_0.145.0_linux-amd64.tar.gz`
- `hugo_extended_0.145.0_Linux-64bit.tar.gz`
- `hugo_extended_0.145.0_linux-amd64.deb`
- `hugo_extended_0.145.0_linux-amd64.tar.gz`

Tar-File

Die Tar-Files können entpackt werden. Die ausführbare Datei wird in einen Ordner verschoben, in dem Linux das Programm findet, zum Beispiel im Heimverzeichnis nach `~/bin/`.

Debian-Paket

Bei Debian-basierten Distributionen kann das Debian-Paket installiert werden.

```
$ sudo dpkg -i hugo_0.145.0_linux-amd64.deb
```

Anlegen eines Hugo-Projekts

Ein neues Hugo-Projekt kann folgendermaßen angelegt werden.

```
$ hugo new site clt25
```

Hugo quittiert diesen Befehl mit einer längeren Meldung.

```
Congratulations! Your new Hugo site was created in /home/juh/clt25.

Just a few more steps...

1. Change the current directory to /home/juh/clt25.
2. Create or install a theme:
   - Create a new theme with the command "hugo new theme <THEMENAME>"
   - Or, install a theme from https://themes.gohugo.io/
3. Edit hugo.toml, setting the "theme" property to the theme name.
4. Create new content with the command "hugo new content <SECTIONNAME>/<FILENAME>.<FOR-
MAT>".
5. Start the embedded web server with the command "hugo server --buildDrafts".
```

```
See documentation at https://gohugo.io/.
```

Was wurde erzeugt?

```
$ clt25 tree
.
 archetypes
    default.md
 assets
 content
 data
 hugo.toml
 i18n
 layouts
 static
 themes

9 directories, 2 files
```

Anlegen eines neuen Artikels

```
$ hugo new content posts/tag1/index.md
```

Was wurde erzeugt?

```
+++
title = 'Tag1'
date = 2024-08-12T12:37:20+02:00
draft = true
+++
```

Woher kommen diese Informationen?

Sie stammen aus der Datei `archetypes/default.md`.

```
+++
title = '{{ replace .File.ContentBaseName "-" " " | title }}'
date = {{ .Date }}
draft = true
+++
```

Der Vorspann einer Markdown-Datei im TOML-Format.
Wir ändern das in der Datei in YAML.

```
---
title: '{{ replace .File.ContentBaseName "-" " " | title }}'
date: {{ .Date }}
draft: true
---
```

Ab sofort wird der Vorspann einer Markdown-Datei in YAML generiert.

Artikel in Markdown schreiben

Wir wollen etwas in dem Artikel schreiben.
`content/posts/tag1/index.md`:

```
---
title: 'Tag 1'
date: 2024-08-12T12:48:31+02:00
description: "Am ersten Tag sind wir gleich zum Strand gegangen."
draft: true
---

Aliquam erat volutpat.
Nunc eleifend leo vitae magna.
In id erat non orci commodo lobortis.
Proin neque massa, cursus ut, gravida ut, lobortis eget, lacus.
Sed diam.
```

Markdown lässt sich umfangreich konfigurieren: `https://gohugo.io/configuration/markup/`

Vorlagen erstellen

Leider wird der erzeugte Artikel nicht angezeigt, wenn wir Hugo im Servermodus starten. Wir sehen weder eine Startseite unter `https://localhost:1313` noch den Artikel unter der vermuteten URL `https://localhost:1313/posts/tag1`

```
$ hugo server --buildDrafts
```

Wir müssen HTML-Vorlagen erstellen.

Hauptvorlage `baseof.html`

`layouts/_default/baseof.html`:

```
<html>
  <head>
    <title>{{ .Title }}</title>
  </head>
  <body>
    {{ block "main" . }}{{ end }}
  </body>
</html>
```

Das ist der äußere Rahmen von allen Seiten, die in der Website erzeugt werden. Die Seiten unterscheiden sich, weil beim Erzeugen der Seiten der Block "main" jeweils individuell gefüllt wird.

Startseitenvorlage `home.html`

Beginnen wir mit der Startseite. Für die Startseite gibt es ein Namensschema. Man kann sie `home.html` nennen.
`layouts/_default/home.html`:

```
{{ define "main" }}
<h1>{{ .Title }}</h1>
{{ .Content }}
{{ end }}
```

Damit wird nun wenigstens die Startseite angezeigt. Sie enthält nur den Titel der Website als Überschrift.

Vorlage für Artikel `single.html`

Wie können wir nun den Artikel "Tag1" anzeigen? Er müsste unter der URL `https://localhost:1313/posts/tag1` zu finden sein.
Hugo unterscheidet grundsätzlich zwischen Vorlagen für einzelne Artikel und Vorlagen, die mehrere Artikel auflisten, also Übersichten oder Inhaltsverzeichnisse.

Das Namensschema für Einzelseiten ist `single.html`.
`layouts/_default/single.html`:

```
{{ define "main" }}
<h1>{{ .Title }}</h1>
<strong>{{ .Description }}</strong>
{{ .Content }}
{{ end }}
```

Übersichten, Listen `list.html`

Das Namensschema für Listen lautet `list.html`.
`layouts/_default/list.html`:

```
{{ define "main" }}
<h1>{{ .Title }}</h1>
<strong>{{ .Description }}</strong>
{{ .Content }}

<h2>Übersicht</h2>
<ul>
```

```
{{ range .Pages }}
<li><a href="{{ .Permalink }}">{{ .Title }}</a></li>
{{ end }}
</ul>
{{ end }}
```

Wir haben nun eine Startseite, eine Übersicht über unsere Posts, und wir können einzelne Artikel darstellen.
Hugo hat noch sehr viele andere Vorlagen: `https://gohugo.io/templates/`

Inhalte und Vorlagen aufeinander abstimmen

Auf der Übersichtsseite für die "Posts" gibt es zwar Leerstellen für eine Beschreibung ".Description" und einen Inhalt ".Content", aber es wird dort nichts angezeigt. Wie können wir diese Lücken füllen.
Wir erstellen im Order `content/posts/` eine Datei mit dem Namen `_index.md`. Der Unterstrich vor dem Namen signalisiert Hugo, dass dieser Inhalt mit einer Listenvorlagen generiert werden soll. Wir machen das über die Kommandozeile, weil der Vorspann dann schon vorausgefüllt wird.

```
$ hugo new content posts/_index.md
```

`content/posts/_index.md`:

```
---
title: 'Posts'
date: 2024-08-14T10:15:43+02:00
draft: true
---
```

Wir ergänzen die Inhalte:

```
---
```

```
title: 'Urlaubs-Blog'
description: 'Wir bloggen über unseren Urlaub, damit alle neidisch werden.'
date: 2024-08-14T10:15:43+02:00
draft: true
---

Aliquam erat volutpat.
Nunc eleifend leo vitae magna.
In id erat non orci commodo lobortis.
Proin neque massa, cursus ut, gravida ut, lobortis eget, lacus.
```

Veröffentlichen

Artikel im Entwurfsmodus (`draft: true`) werden nicht generiert. Beim Aufruf von `hugo` im Servermodus erhalten wir eine 404-Meldung.

```
$ hugo server
```

Sie werden erst generiert, wenn sie aus dem Entwurfsmodus herausgenommen werden (`draft: false`).
Um mit dieser Funktion zu experimentieren, brauchen wir folgende Befehle.

```
$ hugo server --cleanDestinationDir
$ hugo server --buildDrafts --cleanDestinationDir
```

Hugo speichert Dateien in dem Ordner `public`. Das ist in der Voreinstellung das `DestinationDir`. Mit der Option `--cleanDestionationDir` wird dieses Verzeichnis vor dem Bauen gesäubert.
Alternativ kann man Hugo mit der Option `--renderToMemory` starten. Dann speichert Hugo die Dateien nicht im Ordner `public`.

```
$ hugo server --renderToMemory
```

Hugo speichert die generierten Datei in einem Cache. Da auch dort generierte Dateien nicht gelöscht werden, kann man die Option `--ignoreCache` oder `--gc=` benutzen. Dann sollte sich ein Verhalten wie bei `--cleanDestionationDir` einstellen.

```
$ hugo server --renderToMemory --igorneCache
```

```
$ hugo server --renderToMemory --gc
```

Um Änderungen bei der Veröffentlichung zu sehen, ist es jedoch manchmal notwendig, die Seite im Browser neu zu laden. In ganz hartnäckigen Fällen hilft es hugo neuzustarten.
Wenn man Hugo mit der Option `--buildDrafts` startet, werden alle Artikel angezeigt. Es ist empfehlenswert, Artikel im Entwurfsmodus zu kennzeichnen.
Wir ergänzen dazu etwas in der Listenvorlage.
`layouts/_default/list`:

```
<h3>{{ .Title }}{{ if .Draft }}== ENTWURF=={{ end }}</h3>
```

Ordnerbeziehung `content-layout`

Wir haben im Ordner `layouts/_default` Vorlagen für verschiedene Zwecke erstellt. Die Single-Vorlage im Default-Ordner wird auf alle Artikel angewendet und die List-Vorlage auf alle Listen. Hugo unterscheidet Artikel und Listen durch den Namen der Content-Datei. Lautet der Name `index.md` soll ein einzelner Artikel erstellt werden. Lautet er `_index.md` wird die Listenvorlage benutzt.
Hugo kann jedoch sehr viel mehr. So ist es zum Beispiel möglich, für Inhalte, die sich in Unterordnern des Ordners `content` befinden, jeweils eigene Vorlagen zu schreiben. Man kann so für bestimmte Inhalte Layout und Aussehen der Webseite anpassen.

Beispielhafte Übersicht

Ordner `content`	Ordner `layouts`
`posts/`	`posts/single.html`
	`posts/list.html`
`unternehmen/`	`unternehmen/single.html`
	`unternehmen/list.html`
`seminare/`	`seminare/single.html`
`referenten/`	`referente/single.html`
`termine/`	`termine/single.html`
	`termine/list.html`

Die Zuordnung von Vorlagen zu Inhalten ist überaus flexibel in Hugo: `https://gohugo.io/templates/lookup-order/`

Post mit Bild erstellen

Was wäre ein Urlaubs-Blog ohne Fotos. Wir kopieren dafür die Fotodatei `strand.jpg` in den Ordner `content/posts/tag1/`.
In der Datei `content/posts/tag1/index.md` fügen wir das Bild ein.

```
![Angler am Strand](strand.jpg "Im Urlaub entspannen.")
```

Das Ergebnis ist in mehrfacher Hinsicht enttäuschend. Erstens wird das Bild in der Originalauflösung eingefügt und zweitens wird unsere Bildunterschrift nicht eingefügt.
Wir werden deshalb eine interne Vorlage überschreiben.

Bilder richtig skalieren

Die interne Vorlage, die wir überschreiben wollen, heißt `render-image.html`. Sie übernimmt es, die in Markdown eingefügten Dateien zu formatieren. Um das Verhalten dieser Funktion zu ändern, legen wir folgende Datei an: `clt25/layouts/_default/_markup/render-image.html`.

```
{{ $src := .Page.Resources.GetMatch (printf "%s" (.Destination | safeURL))  }}
{{ $alt := .PlainText | safeHTML }}
{{ $title := .Title | safeHTML }}

{{ with $src }}
{{ $scaled := .Resize "720x CatmullRom" }}
<figure>
  <img
    alt="{{ $alt }}"
    title="{{ $title }}"
    src="{{ $scaled.Permalink }}">
  <figcaption>{{ $title }}</figcaption>
</figure>
{{end}}
```

Der Code in den ersten Zeilen interessiert uns hier im Moment nicht. Er dient dazu, das Bild in der Variable `$src` zu speichern. Es kommt uns auf diese beiden Zeilen an:

```
{{ with $src }}
{{ $scaled := .Resize "720x CatmullRom" }}
```

An dieser Stelle wird das in der Variable `$src` gespeicherte Bild bearbeitet. Es wird mit Hilfe des Algorithmus "CatmullRom" auf 720 Pixel Breite gebracht. Hugo hat zahlreiche Bildbearbeitungsfunktionen. Eine Stelle an der dies besonders praktisch ist, sind Vorschaubilder, die im Allgemeinen viel kleiner sind als die Bilder in den Artikeln selbst.

Bilder in Artikeln und Übersichten richtig skalieren

Ein allgemeines Problem ist es, Bilder in jeweils passenden Größen bereitzustellen, damit Webseiten nicht größer werden als unbedingt notwendig.
Dafür müssen wir dem Artikel ein Bild als Ressource zuweisen.
`content/posts/tag1/index.md`:

```
---
title: 'Tag 1'
date: 2024-08-12T12:48:31+02:00
description: "Am ersten Tag sind wir gleich zum Strand gegangen."
draft: true
resources:
- src: "strand.jpg"
  name: "feature"
  title: "Im Urlaub entspannen"
---

Aliquam erat volutpat.
Nunc eleifend leo vitae magna.
```

```
In id erat non orci commodo lobortis.
Proin neque massa, cursus ut, gravida ut, lobortis eget, lacus.
Sed diam.
```

Wir fügen das Bild nicht mehr unten ein, sondern im Vorspann. Nun können wir das Bild sowohl in der Einzelseitenvorlage als auch in der Listenvorlage anzeigen.
Wir modifizieren dafür zunächst die Listenvorlage:
`layouts/_default/list.html`:

```
{{ define "main" }}
<h1>{{ .Title }}</h1>
<strong>{{ .Description }}</strong>
{{ .Content }}

<h2>Übersicht</h2>
{{ range .Pages }}
{{ with .Resources.GetMatch "feature" }}
{{ $scaled := .Resize "200x CatmullRom" }}
<img title="{{.Title }}" src="{{ $scaled.Permalink }}" />
{{ end }}
<h3>{{ .Title }}</h3>
```

```
<p>{{ .Summary }}</p>
<p><a href="{{ .Permalink }}">Weiterlesen</a></p>
{{ end }}
{{ end }}
```

Wir iterieren über die Seiten, nehmen die jeweilige Ressource "feature" und bearbeiten sie mit der Bildbearbeitungsfunktion ".Resize".
In dieser Vorlage kommt dreimal die Variable `.Title` vor. Sie steht jedoch jedesmal in einem anderen Kontext.

1. Ganz oben ist dies der Kontext der jeweiligen Seite, also der Titel aus dem Vorspann in der Markdown-Datei für die Übersicht über alle Blogbeiträge.

2. Mit der Funktion `range` iterieren wir über alle Seiten im Blog, sodass der Kontext wechselt und die Variable `.Title` sich auf den Titel im Vorspann der jeweiligen Markdown-Datei im Ordner `posts` bezieht. In `<h3>{{ .Titel }}</h3>` wird deshalb der jeweilige Titel des Blogartikels eingefügt.

3. Durch die Funktion `with` wechseln wir in den Kontext der Resource "feature", sodass der Titel im `img`-Tag sich auf den Titel der Ressource bezieht.

An vielen Stellen kann es zu einem Kontextwechsel führen. Wenn etwas nicht so funktioniert, wie wir es uns vorstellen, dann liegt es häufig daran, dass wir einen Kontextwechsel übersehen haben und eine Variable auf Informationen aus einem falschen Kontext zugreift.
Wir möchten das Bild aber auch in dem Blogartikel selbst einfügen. Dazu ändern wir die Vorlage `single.html`.

```
{{ define "main" }}
{{ with .Resources.GetMatch "feature" }}
{{ $scaled := .Resize "800x CatmullRom" }}
<img title="{{.Title }}" src="{{ $scaled.Permalink }}" />
{{ end }}
<h1>{{ .Title }}</h1>
<strong>{{ .Description }}</strong>
{{ .Content }}
{{ end }}
```

Wir fügen nahezu den gleichen Code ein und skalieren das Bild auf 800 Pixel Breite.
Hugo hat umfangreiche Bildbearbeitungsfunktionen:

- `https://gohugo.io/functions/images/`
- `https://gohugo.io/methods/resource/`

HTML-Code an anderen Stellen wiederverwerten

Wir haben bereits gesehen, wie man in die Vorlage `baseof.html` den Block `main` einbinden kann, um HTML-Seiten mit unterschiedlichen Inhalten zu erzeugen.
Wenn man bestimmte HTML-Abschnitte in verschiedenen Vorlagen einbinden möchte, nutzt man die sogenannten "Partials".
Partials werden folgendermaßen eingebunden:

```
<html>
  <head>
<title>{{ .Title }}</title>
</head>
  <body>
    {{ block "main" . }}{{ end }}
    <footer>
      {{ partial "footer" . }}
    </footer>
  </body>
</html>
```

Die Partial-Vorlage befindet sich im Ordner `layouts/partials` und heißt `footer.html`.

layouts/partials/footer.html:

```
<hr>
<p><em>Hugo-Workshop auf den Chemnitzer Linux-Tagen</em></p>
```

Theme installieren

Erst einmal ein Git-Repository anlegen.

```
$ cd clt25
$ git init
```

Alternative 1: Manuelle Installation

Bei der manuellen Installation wird das ZIP-File von Github heruntergeladen, entpackt und in den Ordner `themes` kopiert.

Alternative 2: Git Submodul

Dann das Theme als Submodule installieren.

```
$ git submodule add https://github.com/adityatelange/hugo-PaperMod.git themes/hugo-
-PaperMod
```

Theme in hugo.toml eintragen.

```
baseURL = 'https://example.org/'
languageCode = 'en-us'
```

```
title = 'My New Hugo Site'
theme = hugo-PaperMod
```

Alternative 3: Hugo Modul

Die Modularisierung von Hugo, die im Einzelnen hier nicht erklärt werden soll, ermöglicht die Wiederverwendung und getrennte Versionierung von Komponenten.
Um die Modularisierung nutzen zu können, muss die Programmiersprache Go installiert sein. Dann muss man im Projekt die Modularisierung initialisieren. Wir wechseln in den Projektordner und führen den Initialisierungsbefehl aus:

```
$ cd clt25
$ hugo mod init clt25
go: creating new go.mod: module clt25
go: to add module requirements and sums:
        go mod tidy
```

Nach der Ausführung des Befehls befindet sich eine neue Datei im Projektordner mit dem Namen go.mod und folgendem Inhalt:

```
module clt25
```

```
go 1.21.3
```

Anschließend fügen wir das Theme hinzu:

```
$ hugo mod get github.com/adityatelange/hugo-PaperMod
```

Das neue Modul erscheint in der Datei go.mod:
module clt25

```
go 1.21.3

require (
      github.com/adityatelange/hugo-PaperMod v0.0.0-20231104103144-72ab73ffe5ba //
indirect
)
```

Zu guter Letzt muss das Theme in der Konfigurationsdatei eingetragen werden, wobei der gesamte Pfad zum Repository benutzt wird:

```
baseURL = 'http://example.org/'
languageCode = 'de'
title = 'My New Hugo Project'
theme = 'github.com/adityatelange/hugo-PaperMod'
```

Eigenes Theme erstellen

Ein eigenes Theme lässt sich mit einem Befehl erstellen.

```
$ hugo new theme clt25
```

Was ist entstanden?

```
themes
 clt25
     archetypes
        default.md
     assets
        css
           main.css
        js
            main.js
     content
        _index.md
        posts
            _index.md
```

```
        post-1.md
        post-2.md
        post-3
            bryce-canyon.jpg
            index.md
hugo.toml
layouts
   _default
      baseof.html
      home.html
      list.html
      single.html
   partials
       footer.html
       head
          css.html
          js.html
       header.html
       head.html
       menu.html
```

```
        terms.html
    LICENSE
    README.md
    static
       favicon.ico
    theme.toml

14 directories, 25 files
```

Hugo erstellt nicht nur die Ordner, die für ein Theme gebraucht werden. Das Programm legt auch beispielhafte Inhaltsdateien an. Sobald wir `hugo server` aufrufen, sehen wir eine rudimentär gestaltete Website mit Artikeln.
Wir können zu den verschiedenen Post navigieren.
Wenn man die Beispieldateien studiert, erkennt man einige Funktionsweisen von Hugo.

My New Hugo Site

- Home
- Posts
- Tags

Laborum voluptate pariatur ex culpa magna nostrud est incididunt fugiat pariatur do dolor ipsum enim. Consequat tempor do dolor eu. Non id id anim anim excepteur excepteur pariatur nostrud qui irure ullamco.

Post 3

Occaecat aliqua consequat laborum ut ex aute aliqua culpa quis irure esse magna dolore quis. Proident fugiat labore eu laboris officia Lorem enim. Ipsum occaecat cillum ut tempor id sint aliqua incididunt nisi incididunt reprehenderit. Voluptate ad minim sint est aute aliquip esse occaecat tempor officia qui sunt. Aute ex ipsum id ut in est velit est laborum incididunt. Aliqua qui id do esse sunt eiusmod id deserunt eu nostrud aute sit ipsum.

Post 2

Anim eiusmod irure incididunt sint cupidatat. Incididunt irure irure irure nisi ipsum do ut quis fugiat consectetur proident cupidatat incididunt cillum. Dolore voluptate occaecat qui mollit laborum ullamco et. Ipsum laboris officia anim laboris culpa eiusmod ex magna ex cupidatat anim ipsum aute. Mollit aliquip occaecat qui sunt velit ut cupidatat reprehenderit enim sunt laborum. Velit veniam in officia nulla adipisicing ut duis officia. Exercitation voluptate irure in irure tempor mollit Lorem nostrud ad officia.

Post 1

Tempor proident minim aliquip reprehenderit dolor et ad anim Lorem duis sint eiusmod. Labore ut ea duis dolor. Incididunt consectetur proident qui occaecat incididunt do nisi Lorem. Tempor do laborum elit laboris excepteur eiusmod do. Eiusmod nisi excepteur ut amet pariatur adipisicing Lorem. Occaecat nulla excepteur dolore excepteur duis eiusmod ullamco officia anim in voluptate ea occaecat officia. Cillum sint esse velit ea officia minim fugiat. Elit ea esse id aliquip pariatur cupidatat id duis minim incididunt ea ea.

Website mit Musterinhalten

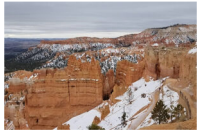
My New Hugo Site

- Home
- Posts
- Tags

Post 3

March 15, 2023

Occaecat aliqua consequat laborum ut ex aute aliqua culpa quis irure esse magna dolore quis. Proident fugiat labore eu laboris officia Lorem enim. Ipsum occaecat cillum ut tempor id sint aliqua incididunt nisi incididunt reprehenderit. Voluptate ad minim sint est aute aliquip esse occaecat tempor officia qui sunt. Aute ex ipsum id ut in est velit est laborum incididunt. Aliqua qui id do esse sunt eiusmod id deserunt eu nostrud aute sit ipsum. Deserunt esse cillum Lorem non magna adipisicing mollit amet consequat.



Sit excepteur do velit veniam mollit in nostrud laboris incididunt ea. Amet eu cillum ut reprehenderit culpa aliquip labore laborum amet sit sit duis. Laborum id proident nostrud dolore laborum reprehenderit quis mollit nulla amet veniam officia id id. Aliquip in deserunt qui magna duis qui pariatur officia sunt deserunt.

Tags:

- Red
- Green
- Blue

Copyright 2025. All rights reserved.

Post 3

Nützliche Links

- Hugo Homepage
- Hilfeforum
- Dokumentation
- Themes
- Github Release Seite