WordPress mit Composer und Git verwalten

Walter Ebert @wltrd

WordCamp Nürnberg 16.-17. April 2016

https://www.flickr.com/photos/20792787@N00/4429177815/
<table>
<thead>
<tr>
<th>Plugin</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable Google Fonts</td>
<td>Disable enqueueing of Open Sans and other fonts used by WordPress from Google.</td>
</tr>
<tr>
<td>HTML Purified</td>
<td>Replaces default HTML filters with HTML Purifier</td>
</tr>
<tr>
<td>Limit Login Attempts</td>
<td>Limit rate of login attempts, including by way of cookies, for each IP.</td>
</tr>
<tr>
<td>Limit Login Attempts on Login only</td>
<td>Trigger the Limit Login Attempts plugin only on login</td>
</tr>
<tr>
<td>Math Captcha</td>
<td>Math Captcha is a 100% effective CAPTCHA for WordPress that integrates into login, registration, comments, Contact Form 7 and bbPress.</td>
</tr>
</tbody>
</table>
WordPress-Aktualisierungen


Du benutzt die aktuelle Version von WordPress.

Wenn du Version 4.4.2-de_DE neu installieren musst, kannst du das hier erledigen oder das Installationspaket herunterladen und manuell installieren:

- Erneut installieren
- 4.4.2-de_DE herunterladen
- Dieses Update ausblenden


Plugins

Für die folgenden Plugins sind neue Versionen verfügbar. Markiere diejenigen, die du aktualisieren möchtest und klicke auf „Plugins aktualisieren“.

- Plugins aktualisieren

- Alle auswählen

- Yoast SEO
  Kompatibilität mit WordPress 4.4.2: 100% (laut dem Autor)

- Alle auswählen

Themes

Alle Themes sind auf dem neuesten Stand.

Übersetzungen
composer installieren

```bash
php -r "readfile('https://getcomposer.org/installer');" > composer-setup.php

sudo php composer-setup.php --install-dir=/usr/local/bin --filename=composer
```
Composer aktualisieren

// stable
composer self-update

// alpha/beta/...
composer self-update --preview

// dev builds
composer self-update --snapshot

https://seld.be/notes/composer-goes-gold
Pakete installieren

$ composer require wp-cli/wp-cli

$ composer require wp-cli/wp-cli:dev-master

./composer.json has been created
Loading composer repositories with package information
Updating dependencies (including require-dev)
...

https://getcomposer.org/doc/03-cli.md#require
Getting Started

Define Your Dependencies
Put a file named `composer.json` at the root of your project, containing your project dependencies:

```json
{
    "require": {
        "vendor/package": "1.3.2",
        "vendor/package2": "1.*",
        "vendor/package3": "^2.0.3"
    }
}
```

For more information about packages versions usage, see the composer documentation.

Install Composer In Your Project
Run this in your command line:

```
curl -sS https://getcomposer.org/installer | php
```

Or download composer.phar into your project root.

See the Composer documentation for complete Installation Instructions on various platforms.

Install Dependencies
Execute this in your project root.

```
php composer.phar install
```
$ ls -lh
-rw-r--r--  1 walter walter   65 composer.json
-rw-r--r--  1 walter walter  45K composer.lock
drwxr-xr-x 13 walter walter  4,0K vendor
composer.json

{
    "require": {
        "wp-cli/wp-cli": "dev-master"
    }
}
composer.json

composer install

https://getcomposer.org/doc/03-cli.md#install
composer.json

composer install

1. composer.lock
2. composer.json

https://getcomposer.org/doc/03-cli.md#install
composer install --no-dev

{
    ...
    "require": {
        "wp-cli/wp-cli": "*",
        ...
    },
    "require-dev": {
        "phpunit/phpunit": "*",
        "squizlabs/php_codesniffer": "*",
        "wp-coding-standards/wpcs": "*"
    },
    ...
}
composer install --no-dev -o

{
  ...
  "require": {
    "wp-cli/wp-cli": "*",
    ...
  },
  "require-dev": {
    "phpunit/phpunit": "*",
    "squizlabs/php_codesniffer": "*",
    "wp-coding-standards/wpcs": "*"
  },
  ...
}

composer.json

composer update

composer update --dry-run

composer update wp-cli/wp-cli
$ vendor/bin/wp --version
WP-CLI 0.24.0-alpha
Composer-Skripte

```
{
  ...
  "scripts": {
    "post-update-cmd": [
      "vendor/bin/wp core update-db",
      "vendor/bin/wp core language update"
    ]
  },
  ...
}
```
Versionen festlegen

"wp-cli/wp-cli": "*"
"wp-cli/wp-cli": "0.22.0"
"wp-cli/wp-cli": "^0.22.0"  
>= 0.22.0 < 1.0.0
"wp-cli/wp-cli": "~0.22.0"  
>= 0.22.0 < 0.23.0
"wp-cli/wp-cli": "<1.0"
"wp-cli/wp-cli": "dev-master"
Versionen festlegen

"wp-cli/wp-cli": "*"
"wp-cli/wp-cli": "0.22.0"
"wp-cli/wp-cli": "^0.22.0"  >= 0.22.0 < 1.0.0
"wp-cli/wp-cli": "~0.22.0"  >= 0.22.0 < 0.23.0
"wp-cli/wp-cli": "<1.0"
"wp-cli/wp-cli": "dev-master"
Semantische Versionierung

1. **MAJOR** wird erhöht, wenn API-inkompatible Änderungen veröffentlicht werden,

2. **MINOR** wird erhöht, wenn neue Funktionalitäten, welche kompatibel zur bisherigen API sind, veröffentlicht werden, und

3. **PATCH** wird erhöht, wenn die Änderungen ausschließlich API-kompatible Bugfixes umfassen.
Nicht nur Pakete

"php": "^5.3.2 || ^7.0"

"ext-gd": "*"
Entwicklerversionen nutzen

{
    "minimum-stability": "dev",
    "prefer-stable": true,
    ...
}

WordPress installieren

```json
{
    "require": {
        "php": "^5.3.2 || ^7.0",
        "ext-gd": "*",
        "composer/installers": "~1.0",
        "johnpbloch/wordpress": "*"
    },
    "extra": {
        "wordpress-install-dir": "web/wp"
    }
}
```
WordPress Packagist

```json
{
    "repositories": [
        {
            "type": "composer",
            "url": "https://wpackagist.org"
        }
    ],
    "require": {
        ...
        "wpackagist-plugin/wordpress-importer": "*",
        "wpackagist-theme/twentyfifteen": "*"
    }
}
```

```
"extra": {
    "wordpress-install-dir": "web/wp",
    "installer-paths": {
        "web/wp-content/mu-plugins/{$name}": ["type:wordpress-muplugin"],
        "web/wp-content/plugins/{$name}": ["type:wordpress-plugin"],
        "web/wp-content/themes/{$name}": ["type:wordpress-theme"]
    }
}
```

http://wpackagist.org/
web/wp-config.php

require __DIR__ . '/../vendor/autoload.php';

... 

define( 'WP_HOME', 'http://example.com' );
define( 'WP_SITEURL', 'http://example.com/wp' );
define( 'WP_CONTENT_URL', 'http://example.com/wp-content' );
define( 'WP_CONTENT_DIR', __DIR__ . '/wp-content/' );
...

define( 'DISALLOW_FILE_MODS', true );
<?php
define( 'WP_USE_THEMES', true );
require __DIR__ . '/wp/wp-blog-header.php';
Verzeichnisstruktur

vendor/
web/wp/
web/wp-content/
web/index.php
web/wp-config.php
web/.htaccess
Strikte Dateirechte

$ ls -l web/wp-content
-rw-r--r-- 1 walter walter index.php
drwxr-xr-x 2 walter walter mu-plugins
drwxr-xr-x 3 walter walter plugins
drwxr-xr-x 5 walter walter themes
drwxr-xr-x 2 www-data www-data uploads

https://bjornjohansen.no/strict-file-ownership-for-wordpress
Eigenes Paket nutzen

```json
{
    "repositories": [ {
        "type":"vcs",
        "url":"https://gitlab/user/plugin"
    } ],
    "require": { ....
        "vendor/package": "*"
    },
    ....
}
```
Kommerzielle Pakete

"config": {"secure-http": false},
"repositories": [
  {
    "type": "package",
    "package": {
      "name": "advanced-custom-fields/advanced-custom-fields-pro",
      "version": "dev-master",
      "type": "wordpress-plugin",
      "dist": {
        "type": "zip",
        "url": "http://connect.advancedcustomfields.com/index.php?p=pro&a=download&k=<license key>"
      }
    }
  }
]
it is a good way to support open source financially. You can find more information about how to set it up and use it on the [Horan Proxy](https://getcomposer.org/doc/articles/handling-private-packages-with-satis.md) website.

## Satis #

Satis on the other hand is open source but only a static repository generator. It is a bit like an ultra-lightweight, static file-based version of packagist and can be used to host the metadata of your company's private packages, or your own. You can get it from [GitHub](https://github.com) or install via CLI:

```
php composer.phar create-project composer/satis --stable=dev --keep-vcs
```

## Setup #

For example let's assume you have a few packages you want to reuse across your company but don't really want to open-source. You would first define a Satis configuration: a json file with an arbitrary name that lists your curated repositories.

Here is an example configuration, you see that it holds a few VCS repositories, but those could be any types of repositories. Then it uses `"require-all": true` which selects all versions of all packages in the repositories you defined.

The default file Satis looks for is `satis.json` in the root of the repository.

```json
{
    "name": "My Repository",
    "homepage": "http://packages.example.org",
    "repositories": [
        { "type": "vcs", "url": "https://github.com/mycompany/privaterepo" },
        { "type": "vcs", "url": "http://svn.example.org/private/repo" },
        { "type": "vcs", "url": "https://github.com/mycompany/privaterepo2" }
    ],
    "require-all": true
}
```
Settings

Public Repositories Proxy

- Proxy packagist.org packages - enables the Packagist proxy repository

Additional Composer repos to proxy (e.g. https://wpackagist.org, https://packages.firegento.com, ...) - one per line

   https://wpackagist.org

Archive Settings

Which zip archives should be pre-fetched by the cron job?

- Lazy: every archive is built on demand when you first install a given package's version
- New tags: tags newer than the oldest version you have used will be pre-cached as soon as they are available
- All: all releases will be pre-cached as soon as they become available

Git Settings (optional, enables git mirroring capability)

- git path (where to store git clones on this machine, must be writable by the web user)
  /home/git/path/to/mirrors/

- git clone url (so composer can clone your repositories, e.g. git@your.toran.proxy:path/to/mirrors/)
  git@::mirrors/
Composer Proxy

{
   "repositories": [
      {
         "type": "composer",
         "url": "https://toranproxy/repo/packagist"
      },
      {
         "packagist": false
      }
   ],
   ...
}
Parallele Downloads

composer global require hirak/prestissimo

{
  ...
  "config": {
    "prestissimo": {
      "maxConnections": 6,
      "minConnections": 3,
      ...
    }
  }
}

...

...
Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.

Learn Git in your browser for free with Try Git.
.gitignore

vendor/
web/wp/
web/wp-content/
ignore ignorieren

vendor/
web/wp/
web/wp-content/*
!web/wp-content/mu-plugins/
Git-Repo-Strategien

• dev-master
• Tags
• Branches
• Remote Branches, z.B.:
  • site, theme, plugin 1, plugin 2, ...
  • dev, stage, prod
  • Feature-Klone
Tags

```
git tag 1.2.1

git tag -a 1.2.1 -m "Version 1.2.1"

git push origin 1.2.1

git push origin --tags
```
.gitattributes

* text=auto
/.gitattributes export-ignore
/.gitignore export-ignore
/composer.json export-ignore
One-click App Deployment with Server-side Git Hooks

Matthew Setter
December 02, 2013

Want cutting-edge content?

Get the best of PHP plus exclusive deals and freebies in your inbox!

you@email.com

Synopsis
INTRODUCTION

In the modern era, software is commonly delivered as a service: called web apps, or software-as-a-service. The twelve-factor app is a methodology for building software-as-a-service apps that:

- Use **declarative** formats for setup automation, to minimize time and cost for new developers joining the project;
- Have a **clean contract** with the underlying operating system, offering **maximum portability** between execution environments;
- Are suitable for **deployment** on modern **cloud platforms**, obviating the need for servers and systems administration;
- Minimize **divergence** between development and production, enabling **continuous deployment** for maximum agility;
- And can **scale up** without significant changes to tooling, architecture, or development practices.

The twelve-factor methodology can be applied to apps written in any programming language, and which use any combination of backing services (database, queue, memory cache, etc).

BACKGROUND

The contributors to this document have been directly involved in the development and deployment of hundreds of apps, and indirectly witnessed the development, operation, and scaling of hundreds of thousands of apps via our work on the **Heroku** platform.
WordPress-Projekte starten

roots/bedrock
https://roots.io/bedrock/

org_heigl/wordpress_bootstrap
https://github.com/heiglandreas/wp_bootstrap

HoloTree
https://github.com/HoloTree/ht-build

wordpress-12factor
https://github.com/dzuelke/wordpress-12factor

wee/wordpress-project
https://gitlab.com/walterebert/wordpress-project
WordPress-Projekte starten

composer create-project roots/bedrock pfad

composer create-project \--repository-url=https://meinrepo/ \meins/wordpress-project \pfad
composer.json
{
   "name": "meins/wordpress-project",
   "description": "WordPress Starter Project",
   "type": "project",
   "repositories": [
      {
         "type": "composer",
         "url": "https://wpackagist.org"
      }
   ],
   ...
}
walter.ebert.engineering
@wltrd
walterebert.de
slideshare.net/walterebert