Webseiten mit AsciiDoc und git

Oder: Warum gibt es inzwischen wieder immer mehr „Static WebSites“ und wie kann ich meine eigene erstellen?

Linux-Cafe 2019-01-07

Bernd Strößenreuther
mailto:linux-cafe@stroessenreuther.net

Lizenz

Agenda

(1) Kleiner Ausflug in die Geschichte

(2) Warum „Static Websites“?

(3) Übungen:
Die verwendeten Tools von Grund auf verstehen

- „pures“ AsciiDoc
- git

(4) Static Website Generatoren: Jekyll & Co.

(5) Hosting-Plattformen: GitHub Pages, GitLab Pages

Kleiner Ausflug in die Geschichte (1)

- Zu Beginn des WWW waren alle Webseiten „statisch“. Man schrieb HTML im Editor.
  - Wiederkehrende Elemente in jeder Seite vorhanden
    → Änderungen aufwändig
- Frames
  - Wiederkehrende Elemente in eigenem Frame
  - Hat nie wirklich gut funktioniert:
    Suchmaschine findet Einzel-Frame statt der kompletten Seite

Kleiner Ausflug in die Geschichte (2)

- Content-Management-Systeme
  (Typo3, WordPress, Drupal, Joomla, …)
  - Seite wird meist erst beim Abruf dynamisch zusammengebaut
  - Webseiten gestalten / Content pflegen meist per Webinterface

Warum „Static WebSites“? (1)

Performance:

- Pfeilschnelle Auslieferung durch den Webserver
- Gut zu cachen
- Ideal für Auslieferung per CDN (Content Delivery Network)
- Sehr viele Webseiten / Blogs / kleinere Firmen-präsentationen verwenden gar keinen dynamischen Content. Warum bei jedem Aufruf neu zusammenbauen?

Editor offlinefähig:

- Auch offline: Seiten anlegen / ändern / Vorschau

Warum „Static WebSites“? (2)

Security:

- Gegenüber CMS: Deutlich reduzierter Angriffsvektor auf den Server, siehe z. B.
  https://www.cvedetails.com/vendor/3887/Typo3.html
  https://www.cvedetails.com/product/2387/Drupal-Drupal.html?vendor_id=1367

Arbeitsablauf:

- Versionshaltung in einem vollwertigen Versions-verwaltungssystem, z. B. git
  → Ältere Version kann schnell wieder hergestellt werden
  → Änderungen selektiv aktivieren / zurückrollen
  → 4-Augen-Prinzip von git kann genutzt werden
  → Bei Bedarf: System zur Freigabe von Änderungen

Übung 1: lokale Arbeitsumgebung

Am Arbeitsplatz-PC:

- Lokales Arbeitsverzeichnis anlegen, `starterpack_asciidoc_website.tgz` herunterladen und dort entpacken

```
mkdir ~/myWebSite
cd ~/myWebSite
wget -qO- https://stroessenreuther.info/pub/adoc.tgz | tar xvz
```

- AsciiDoc installieren

```
sudo apt-get install asciidoc
```

AsciiDoc
Text based document generation

Quelle: http://asciidoc.org/

Übung 2: HTML generieren

Erste Webseite (lokal) erstellen mit AsciiDoc

Am Arbeitsplatz-PC:

- Arbeitsverzeichnis erkunden
- Webseite erstmalig lokal bauen (asciidoc → html)

```
cd ~/myWebSite
bin/build.sh -b
```

Quelle: https://openclipart.org/detail/69331/html-logo

Rückblick: AsciiDoc

- Vortrag von Sebastian im Linux-Cafe:
  https://wiki.gluga.de/talks/AsciiDoctor_Dokumentation_schreiben_kann_Spass_machen.pdf
- AsciiDoc CheatSheet:
  https://powerman.name/doc/asciidoc
- AsciiDoc User Guide:
  https://www.methods.co.nz/asciidoc/userguide.html

Quelle: http://asciidoc.org/

Rückblick: git

- Vortrag im Linux-Cafe:
  https://stroessenreuther.info/pub/Vortrag_Git.pdf
- Git Cheat Sheet:
  https://www.git-tower.com/blog/git-cheat-sheet/

Quelle: https://git-scm.com/downloads/logos

Übung 3: git Repo am Server

Am Server:

- Ein git Repository (bare) anlegen
- Wenn möglich: Direkt auf dem WebServer, der später die Seite ausliefern soll (dann werden die Scripts einfacher)

```
sudo apt-get install git
sudo mkdir /opt/git-repos
sudo chown booboo. /opt/git-repos
mkdir /opt/git-repos/myWebSite.git
cd /opt/git-repos/myWebSite.git && git init --bare
```

Übung 4: git User konfigurieren

Am Arbeitsplatz-PC:

- git installieren / git User konfigurieren:

```
sudo apt-get install git

git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
```

Übung 5: git Repo lokal anbinden

Am Arbeitsplatz-PC:

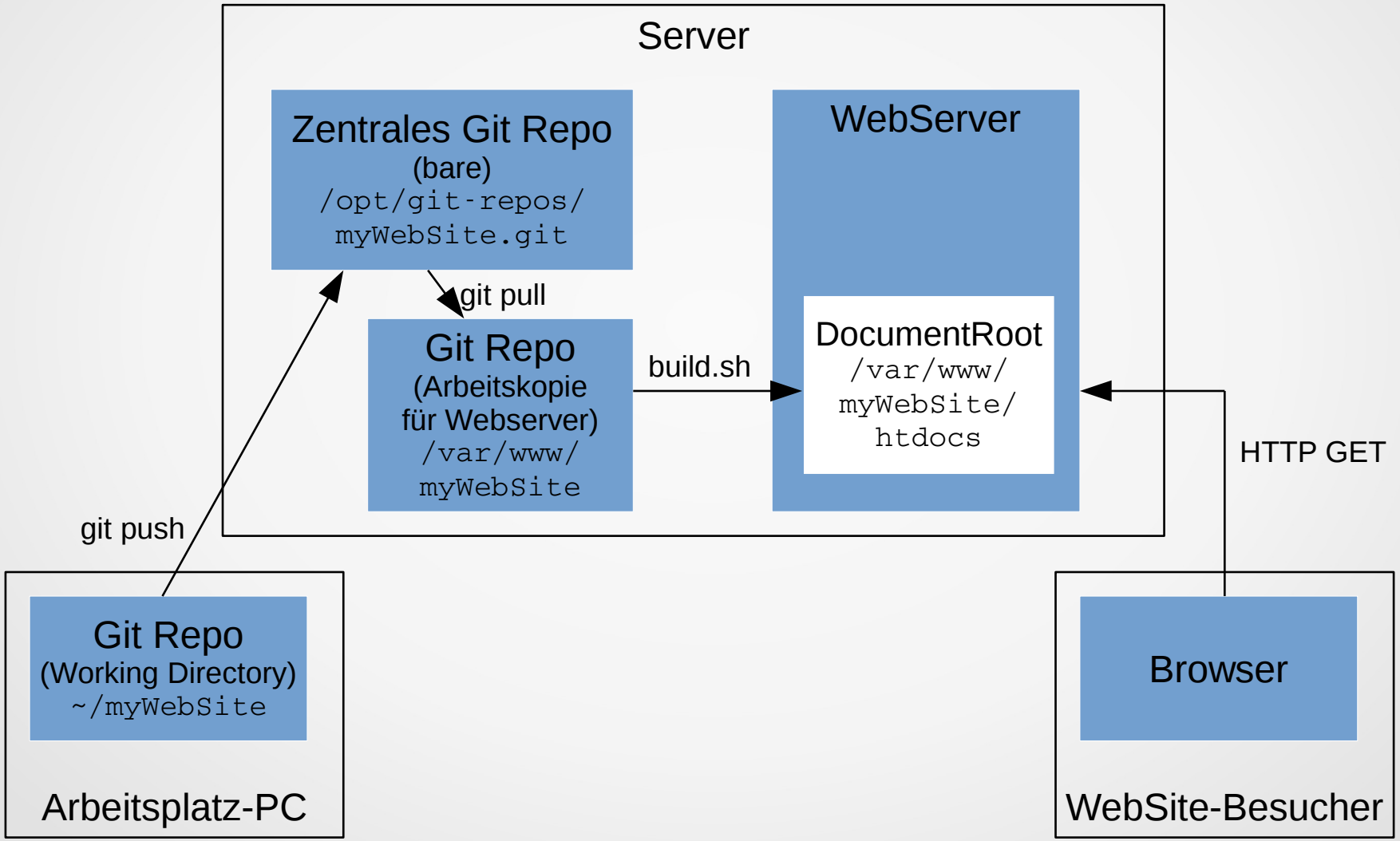
- Wir binden unser lokales Arbeitsverzeichnis ~/myWebSite an das git Repo am Server an

```
cd ~/myWebSite
git init

git remote add origin ssh://dunno.example.com/opt/git-
    repos/myWebSite.git

git remote -v              # Repo URL anzeigen lassen
git add .
git commit -am "initial release"
git push --set-upstream origin master
```

Infrastruktur



Übung 6: Webseite am Server

Am WebServer:

- Apache installieren
- git auschecken und HTML-Seiten erstellen
  (Wir tun das im ersten Schritt manuell)

```
sudo apt-get install apache2
sudo chown booboo. /var/www
cd /var/www

git clone /opt/git-repos/myWebSite.git

cd /var/www/myWebSite
bin/build.sh
```

Übung 7: Webserver Konfiguration

Am Server:

- Wir müssen die Webserver-Konfiguration so anpassen, dass er die Files in `/var/www/myWebSite/htdocs` ausliefert
- Am schnellsten, mit einer kleinen Beispiel-Konfiguration

```
sudo cp ~/myWebSite/conf/apache/myWebSite.conf
    /etc/apache2/sites-available/

sudo a2ensite myWebSite.conf
sudo apache2ctl configtest
sudo service apache2 reload
```

Am Arbeitsplatz-PC:

```
firefox http://dunno.example.com:8080/
```

Quelle: https://openclipart.org/detail/163741/web-server

Übung 8: Content Updates

Am Server:

- Immer wenn neuer Content ins zentrale git-Repo gepushed wird, sollen die Webseiten am Server neu gebaut werden
- Wir verwenden dazu das `post-update` Hook Script im zentralen git-Repository

```
cp ~/myWebSite/git-hooks/post-update /opt/git-
    repos/myWebSite.git/hooks
```

Übung 9: Hook testen

- Wir wollen testen, dass Änderungen automatisch am Server ankommen

Am Arbeitsplatz-PC:

```
cd ~/myWebSite
vim asciidoc/index.asciidoc
    # wir machen irgendeine kleine Änderung im Text
git commit -am "Dummy-Anpassung um Hooks zu testen"
git push
    # die Ausgabe des Hook-Scripts wird angezeigt
    # mit dem Präfix „remote:“

firefox http://localhost:8080
```

Übung 10: Ihr seid dran!

Arbeitsauftrag:

- Erstellt eine zweite Seite
- Stellt sicher, dass diese vom Server abgerufen werden kann
- Baut in beiden Seiten einen Link auf die jeweils andere Seite ein
- Stellt sicher, dass die Links funktionieren. Sowohl dann, wenn Ihr die Seiten lokal generiert (zur Vorschau) als auch am Server.

Hook Scripts für Euer Setup

- `pre-commit` Hook Script
  - für das lokale Arbeitsverzeichnis von git
  - Syntax-Prüfung der AsciiDoc File, … vor dem Commit
  - Hilft Fehler abzufangen, bevor Probleme entstehen

```
cd ~/myWebSite/.git/hooks
ln -s ../../git-hooks/pre-commit
```

- `update` Hook Script
  - für das zentrale git Repo am Server
  - die gleichen Syntax-Checks nochmal zentral, falls der User lokal den pre-commit Hook nicht nutzt

```
cd /opt/git-repos/myWebSite.git/hooks
cp ~/myWebSite/git-hooks/update .
```

Zusatzmaterial für Euer Setup

- `build.sh` kann auch eine `sitemap.xml` generieren. (Parameter `-s`)
  Im `post-update` Hook Script ist der Aufruf von `build.sh` entsprechend zu erweitern.
- Eine `sitemap.xml` hilft Suchmaschinen, Eure Webseite vollständig zu indizieren. Die URL zur `sitemap.xml` ist dazu in `robots.txt` einzutragen, z. B.

```
# Allow any robot full access
User-agent: *
Disallow:

Sitemap: https://dunno.example.com/sitemap.xml
```

Static Website Generatoren

- Jekyll
- Middleman
- Hexo
- Hugo
- Pelican
- ...

Jekyll Installation

- https://jekyllrb.com/
- Installation
  https://jekyllrb.com/docs/installation/

```
sudo apt-get install ruby-full build-essential
sudo gem install bundler jekyll
sudo gem install concurrent-ruby rouge jekyll-feed
sudo gem install jekyll-seo-tag minima
sudo gem install i18n -v '0.9.5'
```

Quelle: https://github.com/jekyll/brand

Jekyll ausprobieren

- Jekyll verwendet per Default Markdown: https://github.com/ithempel/markup-comparison/blob/master/markup-comparison.adoc
- Neue Site erstellen

```
jekyll new myFirstJekyllSite
```

- Seite bauen und im eingebauten „Test-Webserver“ hochfahren:

```
cd myFirstJekyllSite/ && jekyll serve
```

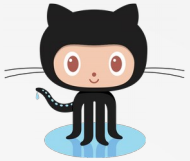
- Nur Seite bauen

```
jekyll build
```

- Theme anpassen: https://jekyllrb.com/docs/themes/#overriding-theme-defaults
- AsciiDoc Plugin: https://github.com/asciidoctor/jekyll-asciidoc

Hosting-Plattformen

- GitHub Pages:
  https://pages.github.com/
- GitLab Pages:
  https://about.gitlab.com/product/pages/
- Arbeiten nach dem Prinzip:
  git + Static Website Generator



Quelle: https://github.com/logos

Quelle: https://about.gitlab.com/press/press-kit/

Soll's Hosting sein?

Vorteile:

- Gehostete Plattform
- Kein Aufwand für Einrichtung / Maintenance

Nachteile:

- Keine 100% Flexibilität
  - z. B. keine Redirects
  - Ggf. nicht alle Konfigurationsmöglichkeiten / Plugins des Static Website Generators nutzbar
- Weniger „Spiel und Spaß“

Noch Fragen?

- Jetzt und hier
- Im Anschluß beim Bier
- Bei (fast) jedem Linux-Cafe Gluga-Stammtisch, ...
- Jederzeit auf der Gluga Users Mailingliste, siehe http://mailing.gluga.de/