# HTTPS no Github Pages com Custom Domain

## Aprenda a colocar a sua página do Github Pages com Custom Domain e HTTPS

## Introdução

Update: O Github agora já dá suporte a https em Custom Domain, você pode ler [nesse post do Github](https://blog.github.com/2018-05-01-github-pages-custom-domains-https/), ou seja, não é mais necessário utilizar os passos que digo abaixo.

Fala pessoal, o post de hoje será para falar sobre uma notícia boa, uma notícia triste e uma solução para isso. Alguns de vocês devem ter visto que o [Github liberou HTTPS para o Github Pages](https://github.com/blog/2186-https-for-github-pages), essa é a notícia boa! A notícia ruim é que não aceita para quem usa Custom Domain, ou seja, um domínio próprio, assim como o meu willianjusten.com.br.

Mas não fique triste, resolvi usar meu próprio blog de cobaia para tentar uma solução, funciona muito bem e é bem fácil de fazer. Só ter paciência (eu num tenho nenhuma) de esperar o DNS propagar direitinho e seguir os passos certinhos.

Vou ouvindo uma playlist que tem combinado um pouco com meus últimos dias, [Life Sucks](https://open.spotify.com/playlist/37i9dQZF1DX3YSRoSdA634?si=1g34PdUdTIa-NX6TCxi3cQ).

## Por que usar HTTPS? Isso num era só para ecommerce?

Primeiro de tudo, precisamos entender o que é HTTPS. O HTTPS é uma implementação do protocolo HTTP já conhecido, sobre uma camada adicional de segurança que utiliza o protocolo SSL/TLS. Essa camada adicional permite que os dados sejam transmitidos por meio de uma conexão criptografada e também é feita uma verificação de autenticidade do servidor e do cliente por meio de certificados digitais.

Esse protocolo ficou mais famoso inicialmente em ecommerces, pois esse tipo de serviço é o mais importante para se ter segurança e passou a ser obrigatório para quase todos gateways de pagamento.

Mas você pode ter esse protocolo em qualquer site, inclusive, a [Google fez um post](https://webmasters.googleblog.com/2014/08/https-as-ranking-signal.html) falando que irá utilizar o HTTPS como um elemento para pontuação nas buscas, então vale bastante a pena.

## Como fazer?

Bom, o primeiro de tudo é, você sabe como fazer uma página no Github Pages? Caso não saiba, dá uma olhada nesse [meu curso de Jekyll](https://www.udemy.com/criando-sites-estaticos-com-jekyll/), lá eu ensino a criar um blog do zero e subir no Github Pages.

A segunda pergunta é, você sabe como fazer um Custom Domain? Não? Então dá uma olhada [nesse post sobre como ter domínio próprio no Github Pages](https://willianjusten.com.br/dominio-proprio-no-github-pages/). Tendo isso feito, só partir para a real tarefa do dia =)

### Cadastro na Cloudflare

A Cloudflare é uma CDN que promete otimizar a entrega do seu site, garantindo segurança, proteção contra DDoS e outras várias coisas. Para se cadastrar é bem fácil, só [entrar no site deles](https://www.cloudflare.com/a/sign-up).

Assim que você se cadastra, deve ver uma tela parecida com a de baixo, pedindo para inserir seu domínio para ele começar a escanear.

Coloque a sua url do github pages e deixe ele escaneando, após o processo, ele ira mostrar uma tela listando o DNS do seu domínio e os apontamentos, similar a tela abaixo:

Ele também vai te passar 2 nameservers que você precisará mudar onde comprou seu domínio.

Em posse desses dois nameservers, vamos lá para o [Registro.br](https://registro.br/), que é onde comprei meu domínio. Lá na parte de DNS tem uma opção para Alterar Servidores DNS, que ao clicar, vai te pedir um Master e um Slave, conforme a imagem:

Feito isso, vem a parada chatinha, ter paciência para o DNS propagar...

### Adicionando SSL via Cloudflare

Bom, eu sei que certificados pagos são melhores que gratuitos, mas como tudo no Github Pages é estático, já sabemos o quanto ele é seguro não é mesmo? Então não tem problema utilizar o gratuito que a Cloudflare provê não.

Para isso, você precisa ir na seção Crypto no Cloudflare assim que já tiver tudo propagado e então em SSL defina a opção Flexible, conforme imagem abaixo:

Feito isso, já temos o nosso certificado ativado, só esperar mais um pouquinho (sim, chatão isso)... E então você já poderá acessar seu site com https e ver o cadeadinho verde na frente. Mas só isso não adianta, pois as pessoas vão continuar podendo acessar sem o https na frente.

### Redirecionando para HTTPS

Para garantir que todo mundo que acesse o site, acesse como https, devemos fazer um redirecionamento 301, onde queremos que todos os links acessados sem https sejam jogados para o link com o protocolo certo. Para isso, é só ir na parte de Crypto e definir essa regra, semelhante a imagem abaixo:

### Links Canônicos

Tá faltando pouquinho, mas não podemos deixar passar uma coisa muito importante, que é dizer para os buscadores que os conteúdos com e sem https são iguais, para eles não caracterizarem isso como cópia. Para isso, basta utilizarmos a seguinte tag:

<link rel="canonical" href="https://www.seudominio.com.br/url\_da\_pagina">

Se você estiverem utilizando um gerador estático como o Jekyll, vocês deixam a parte da url\_da\_pagina dinâmica.

### Disqus

Se for um blog que vocês estão mudando e utilizarem o disqus, vocês vão precisar só fazer uma migração do http para o https, para isso é só criar um arquivo csv, de um lado a url antiga e do outro a url nova, bem fácil, aqui [nesse link](https://help.disqus.com/customer/en/portal/articles/912757-url-mapper) tem o processo.

## Conclusão

Bom pessoal, foi um post bem simples, fiz mais até para salvar o procedimento, vai que no futuro eu precise fazer de novo né? Eu tinha ficado feliz quando vi que teria https no Github Pages, mas quando vi que não funcionaria em Custom Domain, fiquei tristinho e por isso fui ver como resolver o problema. Se você passou pelo mesmo que eu, só seguir aí, que vai funcionar beleza =)

https://willianjusten.com.br/https-no-github-pages-com-custom-domain/

# Criando apresentações e hospedando no github

## Aprenda a criar slides para suas apresentações e hospedá-las online no github pages.

## Introdução

Caramba, como fazia tempo que eu não postava nada aqui. O mês passado foi muito apertado, participei de vários eventos, tanto palestrando como assistindo e por isso não conseguia muito tempo para escrever. Um pouco também foi devido a preguiça...pois é, vergonha né? =/

Bom, para não perder o costume, hoje estou ouvindo uma playlist bem calminha e legal que se chama [Your favorite Coffeehouse](https://open.spotify.com/playlist/37i9dQZF1DX6ziVCJnEm59?si=90_msHLtR3S1YBlx-2vlLA), já tinha passado o olho nela, mas resolvi aproveitar o climinha de Petrópolis para ouvir.

O post de hoje será bem simples, mas que me salvou bastante nesse mês, que como eu disse, precisei fazer várias apresentações.

### Por que não usar o Keynote, PowerPoint ou outro similar?

Bom, eu não tenho as melhores aptidões em design e confesso que ver aquele monte de opções escondidas não me deixa muito confortável. Outra questão é que eu queria poder fazer meus slides em qualquer computador e de qualquer lugar, se eu usasse um desses programas, eu seria obrigado a escrever só em máquinas que tivessem esses programas...

Sempre que eu vou a eventos, me interesso em ter os slides depois para poder ler com mais calma e até mesmo fazer testes quando são códigos expostos ali. Então eu queria disponibilizar meus slides de forma fácil também, sendo assim, eu vi que realmente precisava de algo web.

Eu já havia visto um padrão de algumas palestras e todas elas eram web, fui caçar melhor e então encontrei o amado [RevealJS](https://lab.hakim.se/reveal-js/). Junto com ele achei alguns outros, como o [Slides](http://slides.com/), que é mantido pela galera do RevealJS, só que possui uma UI para poder montar os slides. Também encontrei o [Spectable](_blank), que foi baseado no RevealJS, mas feito todo em React <3 (preciso testá-lo depois!)

## Coisas legais que o RevealJS tem!

Muitos me perguntavam, mas poxa, o Keynote tem várias coisas legais e você vai acabar perdendo... Só que não, o RevealJS tem tudo que o Keynote tem e ainda melhor! Vou colocar uma listinha de coisas muito maneiras que ele tem:

* Slides agrupados, tendo uma navegação vertical no mesmo assunto
* Suporte a Markdown (quem me conhece sabe que amo escrever assim)
* Permissão de exportar em PDF
* Notas do Editor em tela separada + countdown para saber o tempo que já passou
* Sintaxe colorida para código <3
* Vários e vários plugins!

## Sobre o RevealJS

A documentação dele se encontra [aqui](https://github.com/hakimel/reveal.js) e é muito muito fácil de entender e usar. Eu vou passar por alguns detalhes importantes e básicos para construção dos slides.

### Markup

O Markup inicial precisa seguir um padrão, que é:

<div class="reveal">

<div class="slides">

<section>Slide Horizontal Simples</section>

<section>

<section>Slide Veritical 1</section>

<section>Slide Veritical 2</section>

</section>

</div>

</div>

Para iniciar uma apresentação, precisamos de uma div com a classe reveal, que deverá englobar todos os slides. A seção de slides deverá ficar dentro de uma div com a classe slides. Agora cada section dentro dessa organização já será um slide.

### Slide backgrounds

O legalzão do RevealJS é que eu posso colocar qualquer coisa como fundo do meu slide, vídeos, sites, cores e GIFS! E é fácil fácil de fazer funcionar:

<!-- fundos com cores -->

<section data-background="#ff0000"></section>

<!-- fundos com imagens -->

<section data-background="image.png"></section>

<!-- fundos com imagens em tamanho definido e se repetindo -->

<section data-background="image.png" data-background-size="100px" data-background-repeat="repeat"></section>

<!-- vídeos em loop! -->

<section data-background-video="video" data-background-video-loop></section>

<!-- um site dentro do próprio slide! -->

<section data-background-iframe="https://willianjusten.com.br"></section>

### Slide transitions

Podemos fazer diferentes transições entre os slides e para isso, usamos a propriedade data-transition dentro da nossa section, que é o nosso slide.

<!-- Essa transição irá ser passada como um zoom. -->

<section data-transition="zoom"></section>

<!-- Escolha a velocidade do slide entre: default, fast or slow! -->

<section data-transition-speed="fast"></section>

Podemos usar também as transições de entrada e saída de cada slide:

<!-- Transição padrão de deslizar para o lado. -->

<section data-transition="slide"></section>

<!-- Desliza quando entra (slide-in) e esmaece quando sai. (fade-out) -->

<section data-transition="slide-in fade-out"></section>

<!-- Esmaece quando entra (fade-in) e desliza quando sai. (slide-out) -->

<section data-transition="fade-in slide-out"></section>

### Transições dentro do slide

Digamos que queremos ir passando item por item de uma lista, sem mostrar ela toda, para isso, basta usarmos class='fragment' e esse só será mostrado ao pressionar a tecla de próximo.

<section>

<p class="fragment grow">grow</p>

<p class="fragment shrink">shrink</p>

<p class="fragment fade-out">fade-out</p>

<p class="fragment current-visible">visible only once</p>

<p class="fragment highlight-current-blue">blue only once</p>

<p class="fragment highlight-red">highlight-red</p>

<p class="fragment highlight-green">highlight-green</p>

<p class="fragment highlight-blue">highlight-blue</p>

</section>

### Configurações

O RevealJS te permite fazer várias configurações para iniciar a apresentação, como são muitas, prefiro passar [esse link](_blank) que diz todas e ainda explica um pouco sobre cada opção.

## Passos para criar uma apresentação do zero!

#### 1 - Baixe os arquivos no github

Só ir no [github deles](https://github.com/hakimel/reveal.js) e mandar baixar, lá já terão os arquivos necessários para criar a apresentação.

#### 2 - Monte um html básico

!doctype html>

<html lang="pt-br">

<head>

<meta charset="utf-8">

</head>

<body>

</body>

</html>

#### 3 - Adicione o css e o tema no head

<head>

<meta charset="utf-8">

<title>Reveal.js Slide Demo</title>

<link rel="stylesheet" href="css/reveal.css">

<link rel="stylesheet" href="css/theme/default.css" id="theme">

</head>

Existem diferentes temas, basta escolher dentro da pasta de "themes".

#### 4 - Crie o Markup básico

Como dito um pouco mais acima, precisamos ter um markup com as classes reveal e slides para tudo passar a funcionar direitinho.

<div class="reveal">

<div class="slides">

<section>Slide Horizontal Simples</section>

<section>

<section>Slide Veritical 1</section>

<section>Slide Veritical 2</section>

</section>

</div>

</div>

#### 5 - Carregue o Javascript no fim do html

<script src="lib/js/head.min.js"></script>

<script src="js/reveal.min.js"></script>

#### 6 - Inicialize o RevealJS e configure

Não adianta só colocar o js no final do arquivo, precisamos inicializar para que a mágica aconteça!

// Veja todas as configurações em:

// https://github.com/hakimel/reveal.js#configuration

Reveal.initialize({

controls: true, // mostra as setinhas na parte inferior

progress: true, // mostra uma barra de progresso

center: true, // centraliza os slides

transition: 'slide' // none/fade/slide/convex/concave/zoom

});

#### 8 - Hospedando no Github

Como sabemos, o Github tem um sisteminha legal chamado [Github Pages](https://pages.github.com/), que permite hospedar páginas estáticas nele. Para hospedarmos algo lá, basta criarmos um projetinho com o nome que desejarmos e ao invés de usar a branch master, usarmos a branch gh-pages. Após isso, o projeto vai ficar hospedado em nomedousuario.github.io/nome-do-projeto, caso você tenha um domínio configurado, ele ficará como dominio/nome-do-projeto. Segue um exemplo de uma das minhas últimas apresentações [Frontend Carioca 2015](https://willianjusten.com.br/frontend-carioca-2015/).

### Pow, legal, mas não tem nada mais prático não?



Eu sou a preguiça em pessoa e sempre que eu puder, vou tentar fazer algo para agilizar e organizar minhas coisas. Pensando nisso, eu criei o [Qualy Presenter](https://github.com/Qualy-org/qualy-presenter), que além de já ter tudo do RevealJS, inclui algumas coisas legais para o meu desenvolvimento.

Eu uso o [Jade](http://jade-lang.com/) para escrever meus slides separados de forma mais organizada. Tem também o [Stylus](http://learnboost.github.io/stylus/), que é o pre-processador mais lindinho de todos. E para rodar tudo, compilar e fazer todas as outras mágicas, eu tenho o [GulpJS](http://gulpjs.com/).

### Como funciona o Boilerplate?

#### 1 - Estrutura

Todos os arquivos importantes ficam dentro da pasta [src](https://github.com/Qualy-org/qualy-presenter/tree/master/src), que é onde ficam os arquivos ainda não compilados.

A pasta [src/templates](https://github.com/Qualy-org/qualy-presenter/tree/master/src/templates) é onde ficam os [includes](https://github.com/Qualy-org/qualy-presenter/tree/master/src/templates/inc), estes são responsáveis pela inclusão dos scripts, arquivos de css, descrições, meta-tag e etc.

E o arquivo [src/templates/index.pug](https://github.com/Qualy-org/qualy-presenter/blob/master/src/templates/index.pug) é onde eu monto e faço a chamada dos meus slides.

Cada slide fica separado na pasta [src/slides](https://github.com/Qualy-org/qualy-presenter/tree/master/src/slides) e pode ser chamado nessa index na ordem que eu desejar, assim fica tudo mais arrumadinho e me facilita a deletar/chamar ou não um slide =)

Os slides podem ser escritos usando a sintaxe do Pug ou tambem em html normal, então se você não souber escrever em Pug, não se preocupe, escreva em html e tudo irá funcionar igual, basta lembrar de quando for incluir um slide na index.pug, coloque a extensão .html no final do arquivo. Segue exemplo:

doctype html

html

include inc/head

body

.reveal

.slides

include ../slides/slide-1.html

include ../slides/slide-2.html

include inc/scripts

Quando o arquivo for compilado, ele irá ler normalmente e juntar tudo para você em um só arquivo.

#### 2 - Baixando o Boilerplate e instalando

Basta ir em [Qualy Presenter](https://github.com/Qualy-org/qualy-presenter/), clonar ou baixar os arquivos na sua máquina.

Depois basta criar um repositório no seu github com os arquivos iniciais. Segue os passos básicos:

Vá em Create new repository na parte superior do github e preencha os dados corretamente:

// inicializando um repositório git no seu local

git init

// adicionando todos os arquivos

git add .

// commitando tudo

git commit -m "initial commit"

// adicionando o repositório remoto do github

git remote add origin git@github.com:seu-usuario/seus-slides-lindos.git

// fazendo o primeiro deploy na mão para ligar os repositórios

git push -u origin master

#### 3 - Rodando, compilando e fazendo deploy

Como eu disse, uso o Gulp para fazer as mágicas, então a primeira coisa a se fazer é instalar as dependências do package.json, para isso é só rodar:

npm install

Depois de tudo instalado, é só rodar npm start. Ele possui alguns comandos separados, mas em geral o comando principal utilizado será só npm start, que é responsável por fazer todas as compilações necessárias e levantar um servidor com livereload em localhost:3000. Os arquivos compilados irão para uma pasta build, que irá conter só os arquivos finais que são as pastas: css, js, images e index.html.

Depois de tudo feito e compilado, só precisamos fazer o deploy para o github, para isso temos o seguinte comando:

* npm run deploy: faz deploy dos arquivos de build para o branch gh-pages.

## Conclusão

Bom, é só isso pessoal, espero que esse post ajude e que passem a criar mais apresentações direto no browser ao invés de usarem softwares proprietários. E é claro, quem quiser contribuir no meu [boilerplate](https://github.com/Qualy-org/qualy-presenter/), ficarei eternamente grato! Não sabe o que pode melhorar? Fala comigo que eu com certeza tenho uma lista de desejos, que inclusive deveria colocar lá nas issues, mas estou com preguiça, me pergunta que eu falo e você preenche para mim pelo menos, já vai estar ajudando ahuahuahua

# Como ter Domínio Proprio no Github Pages

## O passo a passo para deixar seu blog com seu domínio.

Como já disse no [Making of - Parte 1](https://willianjusten.com.br/making-of-parte-1/) e no [Making of - Parte 2](https://willianjusten.com.br/making-of-parte-2/), o meu blog é hospedado no github pages. E não é só o meu, mas vários, um que está fazendo muito sucesso agora é o blog do [Fernando Daciuk](http://blog.da2k.com.br/), que é feito no [Hexo](https://hexo.io/), mas também hospedado no github pages.

Ou seja, esse processo é para qualquer tipo de site feito no github pages, através de qualquer ferramenta!!

O processo engloba 3 etapas muito fáceis e rápidas:

## 1 - Subindo para o Github Pages

Para subir os arquivos para o Github é bastante simples, mas para isso você precisa ter o [Git](http://git-scm.com/) instalado na sua máquina. Teste o comando git na sua máquina, se ele disser que o comando não existe, faça o download e instale, se ele mostrar a possibilidade de comandos, quer dizer que está instalado =)

Outra dependência é que você cadastre uma conta no [Github](https://github.com/). Tendo isso pronto, é só seguir os seguintes passos:

#### Passo 1: Crie um repositório

Para que funcione no branch master, você precisa criar utilizando o seunomedeusuario.github.io, no meu caso ficou willianjusten.github.io.

#### Passo Final:

# Iniciar um repositório git na pasta do projeto

git init

# Adicionar todos os arquivos

git add -A

# Commitar os arquivos

git commit -m "Commit Inicial"

# Adicionar o repositório remoto

git remote add origin https://github.com/seunomedeusuario/seunomedeusuario.github.io.git

# Subir seu lindo blog

git push -u origin master

Com esses passos o seu Blog já está no Github Pages e se tudo estiver certinho em 20-30 minutos você já pode acessar através da url seunomedeusuario.github.io.

## 2 - Comprando um domínio e direcionando o DNS

Existem vários lugares para se comprar domínio, principalmente dependendo do domínio que você queira comprar. Aqui eu irei falar do mais comum, que é o domínio .com.br.

Para comprar um domínio .com.br você pode utilizar o [Registro.br](http://registro.br/). Abrindo o site, você tem uma ferramenta para verificar se o domínio desejado está livre ou não.

Tendo o domínio livre, basta fazer o registro do domínio e aí lhe serão dadas algumas opções de pagamento, quanto mais anos, maior o desconto no valor final. Após seguir todos os passos, você terá acesso ao painel daquele domínio e poderá ir nas opções de DNS.

Clicando para editar a zona de DNS, você terá que criar 2 A record e um CNAME, conforme a imagem abaixo. Caso tenha alguma outra dúvida, existe o [link oficial do github](https://help.github.com/articles/tips-for-configuring-an-a-record-with-your-dns-provider/) falando sobre.

Lembrando que onde está o meu domínio, você precisa mudar para o seu domínio desejado e o CNAME precisa ser o link do repositório do github.

Feito isso fica faltando somente mais uma etapa.

## 3 - Criação do arquivo CNAME

Esse é molezinha, basta criar um arquivo CNAME (precisar estar maiúsculo). E dentro deste arquivo, você irá colocar o seu domínio próprio. Lembre-se de não colocar protocolo e nem www.

Feito tudo isso, basta esperar um tempinho até o DNS propagar e tudo show de bola =)

# Making of - Parte 1

## Quais ferramentas foram utilizadas para montar esse blog e o porquê de ter escolhido cada uma delas.

Existe uma continuação desse post em [Making of - parte 2](https://willianjusten.com.br/making-of-parte-2/).

## Antes de qualquer coisa...

Bom, antes de começar explicando como criei esse blog, algumas ideias e o que aconteceu durante o processo, é melhor eu me apresentar né?

Meu nome é Willian Justen de Vasconcellos, tenho 24 anos (nada de piadinhas...) e sou Desenvolvedor Front End no [Queremos!](https://queremos.com.br/)/[WeDemand](https://wedemand.com/). Trabalho há cerca de 3 anos na área de web, tendo finalizado meu curso de Tecnologia da Informação na Faeterj-Petrópolis no ano de 2014. Mas curiosamente essa não foi minha primeira faculdade, eu também fiz Química Industrial na Uff, sim, você leu certo, eu realmente fiz Química... E por que eu trabalho com web agora? Ah...porque web é incrível e a facilidade em aprender cada dia mais, me deixa mais feliz com a escolha que fiz.

Eu pretendo escrever o máximo que der e sobre tudo, desde o avançado até dicas rápidas para iniciantes. O foco principal será em tecnologias Front End, mas isso não impede que um dia escreva sobre Python ou alguma técnica de estudo. Espero que gostem do blog e todo feedback é bem vindo =)

## Tecnologias utilizadas

Eu sou um cara muito curioso e gosto sempre de coisas novas, então eu utilizei:

* [Jekyll](http://jekyllrb.com/) - como a base de todo o blog, sendo meu gerador de páginas estáticas.
* [GulpJS](http://gulpjs.com/) - meu automatizador favorito, que utilizo para basicamente todas as tarefas tediosas do frontend.
* [SVG](http://pt.wikipedia.org/wiki/SVG) - ícones escaláveis, fáceis de utilizar e muito leves.
* [Stylus](http://learnboost.github.io/stylus/) - tem pre-processador css mais bonito que stylus? Totalmente flexível, completo e o mais importante, totalmente em Javascript =)
* [Jeet](http://jeet.gs/) - sistema de grid feito em stylus.
* [Kouto Swiss](http://kouto-swiss.io/) - um framework para stylus.
* [Rupture](http://jescalan.github.io/rupture/) - uma ferramenta para facilitar o trabalho com media queries no stylus.
* [Github Pages](https://pages.github.com/) - onde está hospedado esse lindo blog (se você não achou bonito, se retire! Brincadeira, pode continuar, só me manda um feedback do que não gostou).

### Por que de cada uma dessas tecnologias?

Eu acho bastante interessante quando o desenvolvedor explica o porquê dele ter usado tal ferramenta/tecnologia ao invés de outra mais famosa ou mais leve, etc... Por isso, vou explicar rapidamente cada escolha minha. Talvez eu crie posts mais detalhados de algumas dessas tecnologias.

#### Jekyll

Apesar de eu trabalhar atualmente com JS, eu acabei optando pelo Jekyll pela facilidade de iniciar um blog e fazer o deploy no github pages, que inclusive compila os arquivos lá mesmo se eu quiser. Outro fator de ter escolhido o Jekyll foi a sua comunidade, ele atualmente é o gerador estático com maior número de stars(dobro do segundo que é inspirado nele, o Octopress) e também o com maior número de forks, tudo isso você pode olhar nessa [listinha aqui](https://www.staticgen.com/).

E para iniciar o blog, basta ter o ruby instalado na máquina, se você é usuário Mac já vem instalado por padrão (mas aconselho fortemente a atualizar), se você é usuário Linux, algumas distribuições já vem por padrão e outras é só instalar via gerenciador de pacotes. Agora, se você é usuário Windows, muda essa bosta (brincadeirinha xD), mas vai ter algumas dores de cabeça para fazer tudo funcionar... Eu não deveria, mas vou te ajudar usuário Windows, segue o link desse cara que vai resolver o seu problema [Jekyll Windows - Juthilo](http://jekyll-windows.juthilo.com/)

Durante a criação desse blog, eu cheguei a falar sobre ele para alguns amigos, quando disse que estava utilizando Jekyll, alguns me pediram dicas de como usar e outras coisas mais. Então eu vou criar um post mais completo, explicando não só o passo a passo para instalar, mas como ele funciona, ferramentas para se trabalhar melhor com ele e outras coisinhas mais. Se você tem interesse em saber mais sobre o Jekyll, aguarda que logo logo tem post sobre ele.

#### GulpJS

Definitivamente meu automatizador favorito, seja pela velocidade, seja pela facilidade com que é de se escrever para ele. Junto com ele utilizei os seguintes plugins:

* [Browser Sync](https://browsersync.io/docs/gulp) - permite sincronizar todos os meus aparelhos e ainda fazer livereloading, depois que conheci essa ferramenta me apaixonei, se você não conhece, corre para olhar, é fantástica.
* [Gulp-stylus](https://www.npmjs.com/package/gulp-stylus) - para compilar meu lindo Stylus
* [Gulp-uglify](https://www.npmjs.com/package/gulp-uglify) - minificar o js
* [Gulp-concat](https://www.npmjs.com/package/gulp-concat) - concatenar os arquivos
* [Gulp-imagemin](https://www.npmjs.com/package/gulp-imagemin) - minificar todas as minhas imagens

Se quiser entender melhor como eu fiz toda essa bagunça funcionar, só abrir o meu [Gulpfile.js](https://github.com/willianjusten/will-jekyll-template/blob/gh-pages/gulpfile.js)

#### SVG

Para os que me conhecem, sabem que sou apaixonado por SVG e por isso criei a [awesome-svg](https://github.com/willianjusten/awesome-svg), que é um conjunto de informações relacionadas a SVG, desde seu uso básico até opções mais avançadas, se você quer aprender SVG, te aconselho fortemente a ler.

Para esse blog, eu não poderia deixar de usar SVG, sendo assim, todos os ícones utilizados no blog são feitos em svg. Para tal, eu criei uma partial do Jekyll contendo as defs de todos os ícones e então chamei cada ícone que eu desejava usando o use, conforme exemplo abaixo:

<svg class="icon icon-rss"><use xlink:href="#icon-rss"></use></svg>

Eu poderia ficar um post inteiro explicando os motivos de se utilizar SVG, mas os motivos para utilizar SVG no blog foram:

1- Mais leve, segue o arquivo SVG minificado utilizado, tendo somente 6.52kb, [svg-icons.html](https://github.com/willianjusten/will-jekyll-template/blob/gh-pages/_includes/svg-icons.html)

2- [Funciona bem na maioria dos navegadores](http://caniuse.com/" \l "search=svg) (ao menos os que desejo alcançar) e não tem erros bizarros quando demora para carregar, como os font-icons. Segue uma imagem com um bug bem comum dos font-icons.



#### Stylus

Praticamente o pre-processador mais desconhecido entre os grandes e na minha opinião o melhor deles também. Versátil, com centenas de opções e ainda fácil para quem já escreve seguindo indentação, como feito em Python, Ruby, Jade. Fui conhecer essa belezura ano passado através de um post da Ju Gonçalves, ela se não me engano foi uma primeiras pessoas que eu vi falando sobre Stylus no Brasil, aqui fica meu obrigado por ter me mostrado <3

Junto ao Stylus eu utilizo o Jeet, Rupture e Kouto Swiss, como eu disse lá em cima. O Jeet por ser um grid semântico (lê-se nada de col-md-xs-lg-motherfucker). O Rupture, para evitar de escrever @media direto e o Kouto Swiss, que possui várias e várias funções bem legais para trabalhar. Segue abaixo um exemplo de um trecho utilizando o poder desses 3 em conjunto:

.datetime

col(1/6)

text-align center

+below(cut)

stack()

.day

mainFont(800)

color main

font-size rem(80px)

line-height 1.6

+below(cut)

stack()

.month-year

color black

font-size rem(18px)

.content

col(5/6)

+below(cut)

stack()

A linha 2 mostra um dos poderes do Jeet, onde eu defino que o elemento .datetime irá ocupar cerca 1/6 de todo o container. Se estivéssemos trabalhando com o bootstrap seria similar a classe .col-md-2, que representa 2 colunas num total de 12. As linhas 5, 12 e 20 também são do Jeet e esse stack, quer dizer que os elementos devem ser empilhados e o elemento que recebe essa função passa a receber um width: 100%.

As linhas 4, 11 e 19 mostram a utilização do Rupture, onde o +below(cut) é compilado para @media only screen and (max-width:37.5rem), onde 37.5rem é o valor da variável cut. O Rupture permite utilizar palavras como below, above, between, at, o que é muito mais fácil de se escrever e entender do que ficar colocando várias medias queries.

As linhas 9 e 15 mostram uma das funções do Kouto Swiss, que é a conversão de px para rem, que é uma unidade de medida mais adaptável, se você nunca ouviu falar, dá uma lidinha nesse artigo do Tableless [Qual unidade utilizar – Pixel, EM ou REM](https://tableless.com.br/unidade-pixels-em-rem/) .

#### Github Pages

Eu não queria ter problemas com servidor, seja para configurá-lo, seja por problemas dos provedores. Então, a primeira coisa que me veio a cabeça foi o Github Pages, ele é simples, fácil, serve perfeitamente para os meus propósitos e o e melhor, ainda é de graça! O Pages hospeda sites estáticos, ou seja, só arquivos html, se você estava pensando em passar todos os seus clientes com site php para lá, vai tirando o cavalinho da chuva.

O Pages te dá duas opções, criar páginas pessoais, a partir de um repositório com o seu nome, exemplo: willianjusten.github.io ou através de um branch gh-pages num repositório de um projeto qualquer. Além disso, ele também permite configurar um domínio próprio a partir do CNAME.

## Concluindo essa primeira parte

Bom, essa foi uma explicação básica de cada uma das tecnologias empregadas e os motivos de ter escolhido, é importante lembrar que foram escolhas minhas e dentro de um projeto específico. Não vá pensando que essa é a combinação perfeita para tudo, porque não é. Se tiverem alguma dúvida quanto a algo que eu falei, só mandar um comentário aqui embaixo ou se estiver muito tímido, pode mandar uma DM no meu twitter [@willian\_justen](https://twitter.com/willian_justen) ou um email para [willianjustenqui@gmail.com](mailto:willianjustenqui@gmail.com). Na segunda parte eu irei falar um pouco das técnicas que utilizei e até algumas descobertas que fiz pelo caminho. Se não quiser perder nenhum post, só assinar o [RSS](https://willianjusten.com.br/sitemap.xml).

# Making of Parte 2

## Aqui vou mostrar o passo a passo de como criei o blog em Jekyll, usando Gulp e Stylus e coloquei para funcionar no Github Pages.

## Introdução

Na primeira parte eu passei as tecnologias que eu utilizei e o porquê delas, nessa segunda parte eu vou mostrar o passo a passo para montar o ambiente e criar o seu projeto a partir do zero.

Se você quiser pular todo o blá blá blá e ver só o [código fonte](https://github.com/willianjusten/will-jekyll-template/)

Aviso! Este post é bastante extenso, vai lá pegar seu café esperto e volta!

### Instalando o Jekyll

Primeiro de tudo foi necessário criar a base, como disse preferi usar o Jekyll, mesmo ele sendo em Ruby. Como utilizo Mac OS, ele já possuía o ruby instalado por padrão no meu Yosemite, mas como era uma versão mais velha e lenta, preferi atualizar. Vou dar aqui o passo considerando um Mac novo, se algumas dessas etapas você já tiver feito, só passar para a próxima. Caso utilize Linux segue esses passo [aqui](http://michaelchelen.net/81fa/install-jekyll-2-ubuntu-14-04/) e caso use Windows tem [aquele guia que passei](http://jekyll-windows.juthilo.com/)

Passo 1: Instalar o [Homebrew](http://brew.sh/)

ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"

Passo 2: Instalando o [Ruby 2.2](https://www.ruby-lang.org/)

Iremos utilizar o [rbenv](https://github.com/sstephenson/rbenv) que permite trabalhar com diferentes ambientes de Ruby.

brew install rbenv ruby-build

# Adicionando o rbenv ao bash para que seja carregado toda vez que abrir o terminal

echo 'if which rbenv > /dev/null; then eval "$(rbenv init -)"; fi' >> ~/.bash\_profile

source ~/.bash\_profile

# Instalando o Ruby 2.2.0 e definindo como versão padrão

rbenv install 2.2.0

rbenv global 2.2.0

ruby -v

# ruby 2.2.0

Passo 3: Após instalado o ruby, é chegada a vez do Jekyll de fato, o processo é fácil e rápido. Se demorar um pouco e nada acontecer na tela, espere mais um pouco, o ruby infelizmente não dá um feedback durante as instalações e aí acaba dando impressão de travado mesmo...

# Instalando o Jekyll

gem install jekyll

# Iniciando um projeto com ele

jekyll new meu-blog-lindo

Com esses comandos já temos toda a estrutura de pastas do jekyll, de acordo com a imagem abaixo.

### Organizando as pastas

Como podemos notar, o Jekyll já cria uma estrutura bem legal e organizada. Vou explicar resumidamente o que é cada uma e como funciona, mas se quiser algo ainda mais detalhado, só ver a [Documentação](http://jekyllrb.com/docs/home/).

* \_includes: são trechos que se repetem ao longo do site e podem ser incluídos sem ter que digitar tudo.
* \_layouts: como o nome já diz, é a estrutura básica das páginas, em geral são 3 tipos: default para a página de inicial ou listagem de posts. A page que serve para criação de páginas diferenciadas com qualquer conteúdo e post, que é a página de posts.
* \_posts: onde iremos escrever nossos posts, o padrão para escrever é em [Markdown](http://daringfireball.net/projects/markdown/syntax)
* \_sass: os arquivos que geram o css, se você preferir trabalhar em [sass](http://sass-lang.com/), o próprio jekyll compila para você.
* css: pasta para onde vão os arquivos gerados pelo sass.
* \_config.yml: o arquivo de configurações do seu blog, qualquer dado universal ao blog, deve ser colocado lá. Assim como outras definições, como qual tipo de markdown usar, excludes e etc.
* about.md: um arquivo do tipo page que irá gerenciar uma página /about
* feed.xml: arquivo para gerar o feed para o seu blog, essencial para que as pessoas possam seguir seu blog com mais facilidade.
* index.html: A página inicial do seu blog

Quando você compila estes arquivos com o Jekyll, ele gerá uma nova pasta \_site, que irá conter todos os arquivos do site já gerados e estáticos. Pastas iniciadas pelo underline não são compiladas e passadas para a pasta \_site. Outra forma de excluir arquivos é adicionando no \_config.xml. É muito importante adicionar a pasta node\_modules nesse exclude, senão o jekyll irá pensar que precisa copiar todo o conteúdo, o que estragaria muito com o desempenho.

exclude: ['package.json', 'src', 'node\_modules']

Como eu também precisei trabalhar com javascript, stylus e também adicionar imagens. Criei uma pasta src contendo as pastas js, styl e img. Durante a compilação esses arquivos e pastas são jogados para uma pasta assets.

### Usando o GulpJS

Para fazer a compilação do meu Stylus, concatenar meu javascript, minificar imagens, fazer livereloading, enfim, todas as tarefas básicas, resolvi utilizar o [GulpJS](http://gulpjs.com/). Ele funciona com o [NodeJS](http://nodejs.org/), se você não tem na sua máquina ~~se mata~~, só baixar e instalar.

Tendo o NodeJS já instalado, basta ir no terminal e iniciar o projeto:

npm init

Ele vai te fazer umas perguntas, vai seguindo os passos e no final é só confirmar e isso irá gerar um arquivo package.json com os dados preenchidos. Depois basta instalar os plugins necessários, que eu já falei no [post anterior](https://willianjusten.com.br/making-of-parte-1/).

npm install --save-dev gulp gulp-uglify gulp-concat gulp-stylus autoprefixer-stylus browser-sync gulp-imagemin gulp-plumber jeet kouto-swiss rupture

Depois de tudo instalado precisamos fazer nosso Gulpfile.js, que irá ter todas as tasks para automatizar o nosso sistema.

Para ser expert em Gulp, basta saber suas funções principais e que ele trabalha com pipes. Sabendo isso, você consegue criar qualquer tarefa.

* gulp.task(nome, fn): registra uma tarefa com um nome.
* gulp.run(tarefas): executa todas as tarefas sequenciamente
* gulp.watch(tipo de arquivo, fn): fica vigiando o arquivo e roda a função caso ele se modifique
* gulp.src(pasta ou arquivo): indica qual pasta ou arquivo será lido para a tarefa
* gulp.dest(pasta): diz para onde o arquivo final precisa ir

#### Parabéns! Você é um expert em Gulp!

Isso mesmo! É só isso que você precisa saber, sabendo disso, mão na massa!

#### Primeiro definir as variáveis e chamar as dependências

var gulp = require('gulp'),

plumber = require('gulp-plumber'),

browserSync = require('browser-sync'),

stylus = require('gulp-stylus'),

uglify = require('gulp-uglify'),

concat = require('gulp-concat'),

jeet = require('jeet'),

rupture = require('rupture'),

koutoSwiss = require('kouto-swiss'),

prefixer = require('autoprefixer-stylus'),

imagemin = require('gulp-imagemin'),

cp = require('child\_process');

var messages = {

jekyllBuild: '<span style="color: grey">Running:</span> $ jekyll build'

};

#### Definir as tasks do jekyll para dar build e rebuild a cada arquivo atualizado

/\*\*

\* Monta o site do Jekyll

\*/

gulp.task('jekyll-build', function (done) {

browserSync.notify(messages.jekyllBuild);

return cp.spawn('jekyll', ['build'], {stdio: 'inherit'})

.on('close', done);

});

/\*\*

\* Refaz o site e atualiza a página

\*/

gulp.task('jekyll-rebuild', ['jekyll-build'], function () {

browserSync.reload();

});

/\*\*

\* Espera até que o jekyll-build seja executado e então levanta o

\* servidor utilizando o \_site como pasta raiz

\*/

gulp.task('browser-sync', ['jekyll-build'], function() {

browserSync({

server: {

baseDir: '\_site'

}

});

});

#### Compilar o Stylus

Aqui ficam dois detalhes muito importantes, na linha 3, eu utilizo o .pipe(plumber()), essa função serve para receber os erros, mas não matar o processo do Gulp. O Gulp infelizmente fecha caso alguma tarefa dê erro e isso pode ser frustrante quando você ainda está digitando e ele acusa erro. Para evitar isso, utilizamos o gulp-plumber, que é uma mão na roda.

E na linha 5, eu faço as chamadas dos componentes do stylus, assim fica mais fácil de chamar no arquivo stylus, ao invés de colocar chamada para node\_modules eu simplesmente coloco @import "jeet".

gulp.task('stylus', function(){

gulp.src('src/styl/main.styl')

.pipe(plumber())

.pipe(stylus({

use:[koutoSwiss(), prefixer(), jeet(),rupture()],

compress: true

}))

.pipe(gulp.dest('\_site/assets/css/'))

.pipe(browserSync.reload({stream:true}))

.pipe(gulp.dest('assets/css'))

});

#### Minificar e concatenar o JS

Não podemos confundir a ordem aqui, primeiro devemos concatenar todos os arquivos e só depois minificar. O processo de uglify além de deixar tudo em uma linha só, muda também o nome de varíaveis e funções para diminuir ainda mais o arquivo. Se você minificar tudo primeiro e depois concatenar, as vezes pode correr o risco de haver conflito. Outro detalhe importante, a concatenação é feita em ordem alfabética, se você tiver um arquivo a que depende de z, melhor modificar seus arquivos para nomes numerados para que o Gulp concatene na ordem que você deseja. Caso contrário, pode dar erro de dependência. Um erro comum é um plugin que precisa de Jquery estar acima dele e acusar que a variável $ não foi definida.

gulp.task('js', function(){

return gulp.src('src/js/\*\*/\*.js')

.pipe(plumber())

.pipe(concat('main.js'))

.pipe(uglify())

.pipe(gulp.dest('assets/js/'))

});

#### Otimizar imagens

gulp.task('imagemin', function() {

return gulp.src('src/img/\*\*/\*')

.pipe(plumber())

.pipe(imagemin({ optimizationLevel: 3, progressive: true, interlaced: true }))

.pipe(gulp.dest('assets/img/'));

});

#### Vigiar todos os arquivos e Tarefa default

A tarefa default é determinada para que quando se digite somente gulp no terminal, rode a sequência de funções desejadas, neste caso fazemos a compilação de nossos assets primeiro, depois disparamos o server e por último ficamos assintindo para possíveis mudanças.

gulp.task('watch', function () {

gulp.watch('src/styl/\*\*/\*.styl', ['stylus']);

gulp.watch('src/js/\*\*/\*.js', ['js']);

gulp.watch('src/img/\*\*/\*.{jpg,png,gif}', ['imagemin']);

gulp.watch(['index.html', '\_includes/\*.html', '\_layouts/\*.html', '\_posts/\*'], ['jekyll-rebuild']);

});

gulp.task('default', ['js', 'stylus', 'imagemin', 'browser-sync', 'watch']);

Prontinho, agora basta digitar gulp no seu terminal e ele irá rodar todas as tarefas e levantar o servidor em localhost:3000, conforme a imagem abaixo.

Depois disso, basta ter criatividade e montar seu blog com o layout mais legal possível e vai ficar faltando somente a última etapa.

### Subindo para o Github Pages

Para subir os arquivos para o Github é bastante simples, mas para isso você precisa ter o [Git](http://git-scm.com/) instalado na sua máquina. Teste o comando git na sua máquina, se ele disser que o comando não existe, faça o download e instale, se ele mostrar a possibilidade de comandos, quer dizer que está instalado =)

Outra dependência é que você cadastre uma conta no [Github](https://github.com/). Tendo isso pronto, é só seguir os seguintes passos:

#### Passo 1: Crie um repositório

Para que funcione no branch master, você precisa criar utilizando o seunomedeusuario.github.io, no meu caso ficou willianjusten.github.io.

#### Passo Final:

# Iniciar um repositório git na pasta do projeto

git init

# Adicionar todos os arquivos

git add -A

# Commitar os arquivos

git commit -m "Commit Inicial"

# Adicionar o repositório remoto

git remote add origin https://github.com/seunomedeusuario/seunomedeusuario.github.io.git

# Subir seu lindo blog

git push -u origin master

Com esses passos o seu Blog já está no Github Pages e se tudo estiver certinho em 20-30 minutos você já pode acessar através da url seunomedeusuario.github.io.

## Conclusão

Uffa, esse foi um post gigante, mas até o mais leigo se seguir esses passos direitinhos irá conseguir construir seu próprio blog em Jekyll. Se estiver faltando algum detalhe ou algum erro no texto ou código, por favor digam nos comentários, para que eu possa corrigir o mais rápido possível. Se quiserem saber sobre algum determinado assunto, só mandar sugestões =)