

*Yihui Xie, Amber Thomas, Alison Presmanes Hill*

---

# ***blogdown: Creación de sitios web con R Markdown***









To ...





---

# *Contents*

---

List of Tables	v
List of Figures	vii
Prefacio	ix
Sobre los autores	xvii
<b>1 Comienzo</b>	<b>1</b>
1.1 Instalación . . . . .	1
1.1.1 Actualización . . . . .	2
1.2 Un ejemplo rápido . . . . .	2
1.3 RStudio IDE . . . . .	6
1.4 Opciones globales . . . . .	11
1.5 R Markdown vs. Markdown . . . . .	14
1.6 Otros temas . . . . .	20
1.7 Un flujo de trabajo recomendado . . . . .	23
<b>2 Hugo</b>	<b>25</b>
2.1 Sitios estáticos y Hugo . . . . .	25
2.2 Configuración . . . . .	27
2.2.1 Sintaxis TOML . . . . .	29
2.2.2 Opciones . . . . .	31

2.3	Contenido . . . . .	34
2.3.1	Metadatos YAML . . . . .	34
2.3.2	Cuerpo . . . . .	35
2.3.3	Código corto . . . . .	35
2.4	Temas . . . . .	37
2.4.1	El tema por defecto . . . . .	38
2.5	Plantillas . . . . .	42
2.5.1	Un pequeño ejemplo . . . . .	43
2.5.2	Implementando más funciones . . . . .	58
2.6	Layouts personalizados . . . . .	65
2.7	Archivos estáticos . . . . .	68
<b>3</b>	<b>Implementación</b>	<b>71</b>
3.1	Netlify . . . . .	72
3.2	Updog . . . . .	75
3.3	GitHub Pages . . . . .	76
3.4	Travis + GitHub . . . . .	80
3.5	GitLab Pages . . . . .	84
<b>4</b>	<b>Migration</b>	<b>87</b>
4.1	From Jekyll . . . . .	88
4.2	From WordPress . . . . .	92
4.3	From other systems . . . . .	94
<b>5</b>	<b>Other Generators</b>	<b>95</b>
5.1	Jekyll . . . . .	95
5.2	Hexo . . . . .	100
5.3	Default site generator in rmarkdown . . . . .	102
5.4	pkgdown . . . . .	103



<i>Contents</i>	iii
<b>Appendix</b>	<b>105</b>
<b>A R Markdown</b>	<b>105</b>
<b>B Website Basics</b>	<b>109</b>
B.1 HTML . . . . .	111
B.2 CSS . . . . .	116
B.3 JavaScript . . . . .	121
B.4 Useful resources . . . . .	126
B.4.1 File optimization . . . . .	126
B.4.2 Helping people find your site . . . . .	126
<b>C Domain Name</b>	<b>129</b>
C.1 Registration . . . . .	130
C.2 Nameservers . . . . .	130
C.3 DNS records . . . . .	131
<b>D Advanced Topics</b>	<b>135</b>
D.1 More global options . . . . .	135
D.2 LiveReload . . . . .	136
D.3 Building a website for local preview . . . . .	138
D.4 Functions in the blogdown package . . . . .	140
D.4.1 Exported functions . . . . .	140
D.4.2 Non-exported functions . . . . .	141
D.5 Paths of figures and other dependencies . . . . .	142
D.6 HTML widgets . . . . .	144
D.7 Version control . . . . .	145
D.8 The default HTML template . . . . .	147
D.9 Different building methods . . . . .	149

<b>E Personal Experience</b>	<b>153</b>
<b>Bibliography</b>	<b>157</b>
<b>Index</b>	<b>159</b>

---

## *List of Tables*

---

1.1	Opciones globales que afectan el comportamiento de blogdown. . . . .	12
D.1	A few more advanced global options. . . . .	135



---

## *List of Figures*

---

1.1	La página de inicio del nuevo sitio por defecto. . .	4
1.2	Crear una nueva publicación usando el complemento de RStudio. . . . .	8
1.3	Actualizar los metadatos de una publicación existente usando el complemento de RStudio. . . . .	9
1.4	Crear un nuevo proyecto de página web en RStudio.	10
1.5	Crear un proyecto de página web basado en blog-down. . . . .	10
1.6	Opciones de proyecto de RStudio. . . . .	12
2.1	Posibles archivos y carpetas creados cuando crea un nuevo sitio usando <b>blogdown</b> . . . . .	28
2.2	A tweet by Jeff Leek. . . . .	36
2.3	Editar un archivo de texto en línea en GitHub. . .	66
3.1	Configuraciones de ejemplo de un sitio web presentado en Netlify. . . . .	74
B.1	Developer Tools in Google Chrome. . . . .	110
C.1	Some DNS records of the domain yihui.name on Cloudflare. . . . .	132



---

## *Prefacio*

---

En el verano de 2012, Yihui Xie hizo su internado en los laboratorios de investigación AT&T, donde asistió a una charla de Carlos Scheidegger (<https://cscheid.net>), y Carlos dijo algo asó como que “si no tienes un sitio web, hoy en día, no existes”. Luego lo parafraseé como:

---

“Hago web, por ende soy ~~spiderman~~.”

---

Las palabras de Carlos sonaron muy bien, aunque fueron un poco exageradas. Un sitio web bien diseñado y mantenido puede ser extremadamente útil para que otras personas lo conozcan, y usted no necesita esperar oportunidades adecuadas en conferencias u otras ocasiones para presentarse en persona a los demás. Por otro lado, un sitio web también es muy útil para que usted realice un seguimiento de lo que ha hecho y ha pensado. A veces puede regresar a una determinada publicación anterior suya para volver a aprender los trucos o métodos que una vez dominó en el pasado pero que olvidó.

We introduce an R package, **blogdown**, in this short book, to teach you how to create websites using R Markdown and Hugo. If you have experience with creating websites, you may naturally ask what the benefits of using R Markdown are, and how **blogdown** is different from existing popular website platforms, such as WordPress. There are two major highlights of **blogdown**:

En este libro, se presenta un paquete de R, **blogdown**, para enseñarle cómo crear sitios web usando R Markdown y Hugo. Si

tiene experiencia en la creación de sitios web, naturalmente puede preguntarse sobre cuáles son los beneficios de usar R Markdown y cómo **blogdown** es diferente de las plataformas de sitios web populares existentes, como WordPress. Hay dos aspectos principales de **blogdown**:

1. Produce un sitio web estático, lo que significa que el sitio web solo consta de archivos estáticos como HTML, CSS, JavaScript e imágenes, etc. Puede alojar el sitio web en cualquier servidor web (consulte el Capítulo ?? para obtener más información). El sitio web no requiere scripts del lado del servidor como PHP o bases de datos como WordPress. Es solo una carpeta de archivos estáticos. Se explicarán más beneficios de los sitios web estáticos en el Capítulo 2, cuando se presente el generador de sitios web estáticos Hugo.
2. El sitio web se genera a partir de documentos R Markdown (R es opcional, es decir, puede usar documentos de Markdown sin fragmentos de código R). Esto brinda una gran cantidad de beneficios, especialmente si su sitio web está relacionado con el análisis de datos o la programación (en R). Poder utilizar Markdown implica simplicidad y, lo que es más importante, *portabilidad* (por ejemplo, se está dando la oportunidad de convertir sus publicaciones de blog a formato PDF y publicarlas en revistas o incluso libros en el futuro). R Markdown le brinda los beneficios de los documentos dinámicos — todos sus resultados, tales como tablas, gráficos y valores en línea, se pueden calcular y representar dinámicamente desde el código en R, por lo tanto, es más probable que los resultados que presente en su sitio web sean reproducibles. Un beneficio adicional pero importante de usar R Markdown es que podrá escribir documentos técnicos fácilmente, debido a que **blogdown** hereda el formato de salida HTML de **bookdown** (Xie, 2016). Por ejemplo, es posible escribir ecuaciones matemáticas LaTeX, citas en BibTeX e incluso teoremas y pruebas si lo desea.



No se deje engañar por la palabra “blog” en el nombre del paquete: **blogdown** es para sitios web de propósito general, y no solo para blogs. Por ejemplo, todos los autores de este libro tienen sus sitios web personales, donde puede encontrar información sobre sus proyectos, blogs, documentación de paquetes, etc.<sup>1</sup> Todas sus páginas están compiladas a partir de **blogdown** y Hugo.

Si no prefiere usar Hugo, también existen otras opciones. El capítulo ?? presenta posibilidades para usar otros generadores de sitios web, tales como Jekyll y el generador por defecto de **rmarkdown**.

Este libro ha sido publicado por Chapman & Hall/CRC<sup>2</sup>. La versión en línea de este libro está licenciada bajo Licencia Internacional Creative Commons Attribution-NonCommercial-ShareAlike 4.0<sup>3</sup>.

---

## Estructura del libro

El capítulo ?? tiene como objetivo comenzar con un nuevo sitio web basado en **blogdown**: contiene una guía de instalación, un ejemplo rápido, una introducción a los complementos de RStudio relacionados con **blogdown**, y comparaciones de diferentes formatos de documentos de origen. Todos los lectores de este libro deben terminar al menos este capítulo (para saber cómo crear un sitio web localmente) y la sección 3.1 (para saber cómo publicar

---

<sup>1</sup>La página principal de Yihui está en <https://yihui.name>. Escribe entradas de blog en chino (<https://yihui.name/cn/>) e inglés (<https://yihui.name/en/>), y documenta sus paquetes de software como **knitr** (<https://yihui.name/knitr/>) y **animation** (<https://yihui.name/animation/>). De vez en cuando también escribe artículos como <https://yihui.name/rlp/> cuando encuentra temas interesantes, pero no se molesta con un envío formal de un diario. La página principal de Amber está en <https://amber.rbind.io>, donde puede encontrar su blog y páginas de proyectos. El sitio web de Alison está en <https://alison.rbind.io>, que utiliza un tema académico en este momento.

<sup>2</sup><https://www.crcpress.com/p/book/9780815363729>

<sup>3</sup><http://creativecommons.org/licenses/by-nc-sa/4.0/>

un sitio web). El resto del libro es principalmente para aquellos que desean personalizar aún más sus sitios web.

El capítulo 2 presenta brevemente el generador de sitios web estáticos Hugo, en el que se basa **blogdown**. Se intentó resumir la documentación oficial de Hugo en un breve capítulo. Debe consultar la documentación oficial en caso de duda. Puede omitir la sección 2.5 si no tiene conocimientos básicos de tecnologías web. Sin embargo, esta sección es crítica para que entienda completamente Hugo. Se ha invertido la mayor parte del tiempo en esta sección de este capítulo. Es muy técnico, pero debería ser útil, no obstante. Una vez que haya aprendido cómo crear plantillas Hugo, tendrá la plena libertad de personalizar su sitio web.

El capítulo ?? le dice cómo publicar un sitio web, para que otras personas puedan visitarlo a través de un enlace. El capítulo ?? muestra cómo migrar sitios web existentes desde otras plataformas a Hugo y **blogdown**. El capítulo ?? ofrece algunas otras opciones si no desea usar Hugo como su generador de sitios.

El Apéndice A es un tutorial rápido sobre R Markdown, el requisito previo de **blogdown** si va a escribir código R en sus publicaciones. El Apéndice ?? contiene conocimientos básicos sobre sitios web, como HTML, CSS y JavaScript. Si realmente le importa su sitio web, tendrá que aprenderlo algún día. Si desea tener su propio nombre de dominio, el Apéndice @ref (nombre-dominio) proporciona una introducción. También hemos cubierto algunos temas opcionales en el Apéndice ?? para usuarios avanzados.

---

## Información de Software y convenciones

La información de la sesión de R al compilar este libro se muestra a continuación:

**sessionInfo()**

```
## R version 3.4.3 (2017-11-30)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS High Sierra 10.13.3
##
## Matrix products: default
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-
## 8
##
## attached base packages:
## [1] stats      graphics  grDevices  utils      datasets
## [6] base
##
## loaded via a namespace (and not attached):
## [1] bookdown_0.7    blogdown_0.5    rmarkdown_1.9
## [4] htmltools_0.3.6 knitr_1.20
```

No agregamos avisos (> y +) al código fuente en R de este libro, y comentamos el resultado de texto con dos numerales ## por defecto, como puede ver en la información de la sesión en R anterior. Esto es para su conveniencia cuando quiera copiar y ejecutar el código (la salida de texto será ignorada ya que está comentada). Los nombres de los paquetes están en negrita (por ejemplo, **rmarkdown**), y el código en línea y los nombres de los archivos están formateados en una fuente de máquina de escribir (por ejemplo, `knitr::knit('foo.Rmd')`). Los nombres de las funciones están seguidos por paréntesis (por ejemplo, `blogdown::serve_site()`). El operador doble punto `::` significa acceder a un objeto desde un paquete.

Una barra inclinada a menudo indica un nombre de directorio, por ejemplo, `content/` significa un directorio llamado `content` en lugar de un archivo llamado `content`. Una barra diagonal en una ruta indica

el directorio raíz del sitio web, por ejemplo, `/static/css/style.css` significa el archivo `static/css/style.css` bajo el directorio raíz de su proyecto de sitio web en lugar de que esté bajo su sistema operativo. Tenga en cuenta que algunos nombres de directorios son configurables, como `public/`, pero usaremos sus valores predeterminados en todo el libro. Por ejemplo, su sitio web se presentará en el directorio `public/` de forma predeterminada, y cuando vea `public/` en este libro, debería considerarlo como el directorio de publicación real que estableció si cambió el valor predeterminado. `Rmd` significa R Markdown en este libro, y es la extensión del nombre de archivo de R Markdown.

Una “publicación” a menudo no significa literalmente una publicación de blog, sino que se refiere a cualquier documento fuente (Markdown o R Markdown) en el proyecto del sitio web, incluidas publicaciones de blog y páginas normales. Normalmente, las publicaciones de blog se almacenan en el directorio `content/post/`, y las páginas están en otros directorios (incluido el directorio raíz `content/` y sus subdirectorios), pero Hugo no requiere esta estructura.

La URL `http://www.example.com` se usa solo con fines ilustrativos. No queremos decir que realmente deba visitar este sitio web. En la mayoría de los casos, debe reemplazar `www.example.com` con su nombre de dominio real.

Un asterisco `*` en una cadena de caracteres a menudo significa una cadena arbitraria. Por ejemplo, `*.example.com` denota un subdominio arbitrario de `example.com`. Podría ser `foo.example.com` o `123.example.com`. En realidad, `foo` y `bar` también indican caracteres u objetos arbitrarios.

---

## Agradecimientos

Originalmente, Yihui planeó escribir solo una oración en esta sección: “Agradezco a Tareef”. Este libro y el paquete **blogdown** no se habrían terminado sin Tareef, el presidente de RStudio. Él ha

estado “empujándolo suavemente” todas las semanas desde el Día 1 de **blogdown**. Como una persona sin una gran autodisciplina y trabajando de forma remota, Yihui se benefició mucho de las reuniones semanales con Tareef. También le dio muchas buenas sugerencias técnicas para mejorar el paquete. En realidad, fue uno de los primeros usuarios de **blogdown**.

Por supuesto que a Yihui le gustaría agradecer a RStudio por la maravillosa oportunidad de trabajar en este nuevo proyecto. Él estaba aún más entusiasmado con **blogdown** que **bookdown** (su proyecto anterior). Él empezó a bloguear hace 12 años y ha usado y dejado varias herramientas para crear sitios web. Finalmente Yihui se siente satisfecho con su propia “comida para perros”.

Muchos usuarios han suministrado valiosa retroalimentación y han reportado problemas a través de GitHub issues (<https://github.com/rstudio/blogdown/issues>). Dos de los favoritos de Yihui son <https://github.com/rstudio/blogdown/issues/40> y <https://github.com/rstudio/blogdown/issues/97>. Algunos usuarios también han contribuido con código y han mejorado este libro a través de pull requests (<https://github.com/rstudio/blogdown/pulls>). Puede encontrar la lista de contribuyentes en <https://github.com/rstudio/blogdown/graphs/contributors>. Muchos usuarios siguieron la sugerencia de formular preguntas en StackOverflow (<https://stackoverflow.com/tags/blogdown>) en lugar de usar GitHub issues o correos electrónicos. Yihui aprecia su ayuda, paciencia y comprensión. Él también quisiera hacer una mención especial a su pequeño amigo Jerry Han, quien fue, probablemente, el usuario de **blogdown** más joven.

Para este libro, Yihui tuvo la suerte de trabajar con sus coautores, Amber y Alison, que son excepcionalmente buenas para explicar las cosas a los principiantes. Esa es la habilidad que más deseo. Huelga decir que han hecho este libro más amigable para principiantes. Además, Sharon Machlis contribuyó con algunos consejos sobre optimización de motores de búsqueda en este libro (<https://github.com/rstudio/blogdown/issues/193>). Raniere Silva contribuyó con la sección 3.5 (<https://github.com/rstudio/blogdown/pull/225>).

A Yihui le gustaría agradecer a todos los autores y colaboradores

de Hugo (Bjørn Erik Pedersen y Steve Francia *et al.*) por su potente generador de sitios estáticos. Al menos le hizo disfrutar construyendo sitios web estáticos y blogs, nuevamente.

Por alguna razón, una parte de la comunidad de R comenzó a adoptar el modelo de “desarrollo impulsado por stickers” al desarrollar paquetes. Esperaba que **blogdown** también tuviera un sticker, así que Yihui pidió ayuda en Twitter (<https://twitter.com/xieyihui/status/907269861574930432>) y obtuvo toneladas de borradores de logotipos. En particular, quisiera agradecer a Thomas Lin Pedersen por su arduo trabajo en un diseño muy inteligente. La versión final del logotipo fue proporcionada por Taras Kaduk y Angelina Kaduk, y realmente lo aprecia.

Este es el tercer libro que Yihui ha publicado con su editor en Chapman & Hall/CRC, John Kimmel. Siempre le ha gustado trabajar con él. Rebecca Condit y Suzanne Lassandro revisaron el manuscrito y aprendió mucho de sus comentarios y sugerencias profesionales.

Yihui Xie  
Elkhorn, Nebraska

---

## *Sobre los autores*

---

Yihui es el desarrollador principal del paquete **blogdown**. No comenzó a trabajar en la documentación sistemática (es decir, este libro) hasta cuatro meses después de comenzar el proyecto **blogdown**. Un día, encontró un tutorial de **blogdown** muy agradable en Twitter escrito por Amber Thomas. Sorprendido de que pudiera crear un gran sitio web personal usando **blogdown** y escribir un tutorial *cuando no había documentación oficial*, Yihui inmediatamente la invitó a unirse a él para escribir este libro, aunque nunca antes se habían conocido. Esto definitivamente no habría sucedido si Amber no tuviera un sitio web. Por cierto, Amber formuló la primera pregunta<sup>4</sup> con la etiqueta `blogdown` en StackOverflow.

Alrededor de medio año después, Yihui vio otro tutorial de **blogdown** muy bien escrito por Alison en su sitio web personal, cuando este libro todavía no estaba completo. Sucedió lo mismo, y Alison se convirtió en el tercer autor de este libro. Los tres autores no se conocían.

Con suerte, puede ver mejor por qué debería tener un sitio web ahora.

---

### Yihui Xie

Yihui Xie (<https://yihui.name>) es ingeniero de software en RStudio (<https://www.rstudio.com>). Obtuvo su doctorado en el Departamento de Estadística de la Universidad Estatal de Iowa. Está interesado en gráficos estadísticos interactivos y computación es-

---

<sup>4</sup><https://stackoverflow.com/q/41176194/559676>

tadística. Como usuario de R activo, es autor de varios paquetes en R, como **knitr**, **bookdown**, **blogdown**, **xaringan**, **animation**, **DT**, **tuftes**, **formatR**, **fun**, **mime**, **highr**, **servr**, y **Rd2roxygen**, entre los cuales **animation** ganó el premio John M. Chambers Statistical Software Award (ASA) 2009 . También fue coautor de algunos otros paquetes R, entre los que se incluyen **shiny**, **rmarkdown** y **leaflet**.

En 2006, fundó Capital of Statistics (<https://cosx.org>), que se ha convertido en una gran comunidad en línea de estadísticas en China. Inició la conferencia de R de China en 2008 y desde entonces ha participado en la organización de conferencias de R en China. Durante su formación de doctorado en la Iowa State University, ganó el Vince Sposito Statistical Computing Award (2011) y el Snedecor Award (2012) en el Departamento de Estadística.

De vez en cuando despotrica en Twitter (<https://twitter.com/xieyihui>), y la mayoría de las veces lo pueden encontrar en GitHub (<https://github.com/yihui>).

Le gusta la comida picante tanto como la literatura china clásica.

---

## Amber Thomas

Amber Thomas (<https://amber.rbind.io>) es periodista de datos y “creadora” de la publicación en línea de ensayos visuales: The Pudding (<https://pudding.cool>). Sin embargo, su formación académica se encontraba en un campo completamente diferente: la biología marina. Tiene una licenciatura en biología marina y química de la Universidad Roger Williams y una maestría en ciencias marinas de la Universidad de Nueva Inglaterra. A lo largo de su carrera académica y profesional como bióloga marina, se dio cuenta de que amaba el análisis de datos, la visualización y la narración de historias a través de los datos y, por lo tanto, cambió las trayectorias profesionales por algo un poco más centrado en los datos.

Mientras buscaba trabajo, comenzó a realizar proyectos person-



ales para ampliar su conocimiento del funcionamiento interno de R. Decidió poner todos sus proyectos en un solo lugar en línea (para que pudiera ser descubierta, naturalmente) y después de muchas búsquedas, tropezó con un lanzamiento anticipado del paquete **blogdown**. Se enganchó de inmediato y pasó unos días configurando su sitio web personal y escribiendo un tutorial sobre cómo lo hizo. Puede encontrar ese tutorial y algunos de sus otros proyectos y reflexiones en su sitio de blogdown.

Cuando no está machacando números y tratando de mantenerse al tanto de la bandeja de entrada de su correo electrónico, por lo general, Amber recibe aire fresco de Seattle o se acurruca con su perro, Sherlock. Si la está buscando en el mundo digital, pruebe <https://twitter.com/ProQuesAsker>.

---

### Alison Presmanes Hill

Alison (<https://alison.rbind.io>) es profesora de pediatría en el Centro para el Entendimiento del Lenguaje Hablado de la Universidad de Salud y Ciencia de Oregon (OHSU, por sus siglas en inglés) en Portland, Oregon. Alison obtuvo su doctorado en psicología del desarrollo con una concentración en métodos cuantitativos de la Universidad de Vanderbilt en 2008. Su investigación actual se centra en desarrollar mejores medidas de resultado para evaluar el impacto de nuevos tratamientos para niños con autismo y otros trastornos del neurodesarrollo, utilizando procesamiento del lenguaje natural y otros métodos computacionales. Alison es autora de numerosos artículos de revistas y capítulos de libros, y su trabajo ha sido financiado por los Institutos Nacionales de Salud, el Instituto de Investigación Translacional y Clínica de Oregón y Autism Speaks.

Además de la investigación, Alison imparte cursos de postgrado en el programa de Ciencias de la Computación de OHSU (<https://www.ohsu.edu/csee>) sobre estadística, ciencia de datos y visualización de datos usando R. También ha desarrollado y dirigido varios talleres

de R y sesiones de entrenamiento en equipos más pequeñas, y le encanta entrenar nuevos “usuarios”. Puede encontrar algunos de sus talleres y materiales de enseñanza en GitHub (<https://github.com/apreshill>) y, por supuesto, en su sitio **blogdown**.

Siendo una madre nueva, los libros favoritos actuales de Alison son *The Circus Ship* y *Bats at the Ballgame*. También realiza interpretaciones entusiastas de la mayoría de las canciones de Emily Arrow (solo para audiencias privadas).

# 1

---

## Comienzo

---

En este capítulo, mostramos cómo crear un sitio web simple desde cero. El sitio web contendrá una página de inicio, una página “Acerca de”, una publicación de R Markdown y una publicación de markdown normal. Aprenderá los conceptos básicos para crear sitios web con **blogdown**. Para principiantes, le recomendamos que comience con RStudio IDE, pero realmente no es necesario. RStudio IDE puede facilitar algunas cosas, pero puede usar cualquier editor si no le importan los beneficios adicionales en RStudio.

---

### 1.1 Instalación

Asumimos que ya ha instalado R (<https://www.r-project.org>) (R Core Team, 2017) y RStudio IDE (<https://www.rstudio.com>). Si no tiene instalado RStudio IDE, instale Pandoc (<http://pandoc.org>). A continuación, tenemos que instalar el paquete **blogdown** en R. Está disponible en CRAN y GitHub, y puede instalarlo con:

```
## Intalación desde el CRAN
install.packages('blogdown')
## O, instalación desde GitHub
if (!requireNamespace("devtools")) install.packages('devtools')
devtools::install_github('rstudio/blogdown')
```

Como **blogdown** se basa en el generador de sitios estáticos Hugo

(<https://gohugo.io>), también debe instalar Hugo. Hay una función auxiliar en **blogdown** para descargar e instalar automáticamente en los principales sistemas operativos (Windows, MacOS y Linux):

```
blogdown::install_hugo()
```

Por defecto, instala la última versión de Hugo, pero puede elegir una versión específica a través del argumento `versión`, si lo prefiere.

Para los usuarios de macOS, `install_hugo()` usa el administrador de paquetes Homebrew (<https://brew.sh>) si ya se ha instalado, de lo contrario solo descarga el binario de Hugo directamente.

### 1.1.1 Actualización

Para actualizar o reinstalar Hugo, use `blogdown::update_hugo()`, que es equivalente a `install_hugo(force = TRUE)`. Puede verificar la versión de Hugo instalada mediante `blogdown::hugo_version()`, y encontrar la última versión de Hugo en <https://github.com/gohugoio/hugo/releases>.

---

## 1.2 Un ejemplo rápido

Según nuestra experiencia, la documentación de Hugo puede ser un poco desalentadora para leer y digerir para principiantes.<sup>1</sup> Por ejemplo, su guía de “Inicio rápido” solía tener 12 pasos, y usted puede perderse fácilmente si no ha utilizado un generador de sitio web estático antes. Para **blogdown**, esperamos que los usuarios de todos los niveles al menos puedan comenzar lo más rápido posible. Hay muchas cosas que puede desear modificar para el sitio web

---

<sup>1</sup>Un día Yihui estaba casi listo para suicidarme cuando intentaba averiguar cómo funciona `_index.md` leyendo la documentación una y otra vez, y buscando desesperadamente en el foro de Hugo.

más adelante, pero el primer paso es bastante simple: crear un nuevo proyecto en un directorio nuevo en RStudio IDE (`File -> New Project`) y llamar a la función en la consola de R del nuevo proyecto:

```
blogdown::new_site()
```

Luego espere a que esta función cree un sitio nuevo, descargue el tema predeterminado, agregue algunas publicaciones de muestra, ábralas, cree el sitio y ejecútelo en RStudio Viewer, para que pueda obtener una vista previa de inmediato. Si no usa RStudio IDE, necesita asegurarse de que se encuentra actualmente en un directorio vacío,<sup>2</sup> en cuyo caso `new_site()` hará lo mismo, pero el sitio web se lanzará en su navegador web en lugar de RStudio Viewer.

Ahora debería ver un grupo de directorios y archivos en el proyecto RStudio o en su directorio de trabajo actual. Antes de explicar estos nuevos directorios y archivos, primero introduzcamos una tecnología importante y útil: *LiveReload*. Esto significa que su sitio web<sup>3</sup> Se reconstruirá y volverá a cargar automáticamente en su navegador web<sup>4</sup> Cuando modifique cualquier archivo fuente de su sitio web y lo guarde. Básicamente, una vez que inicie el sitio web en un navegador web, ya no necesita volver a generarlo explícitamente. Todo lo que necesita hacer es editar los archivos fuente, como los documentos R Markdown, y guardarlos. No es necesario hacer clic en ningún botón ni ejecutar ningún comando. LiveReload se implementa a través de `blogdown::serve_site()`, que

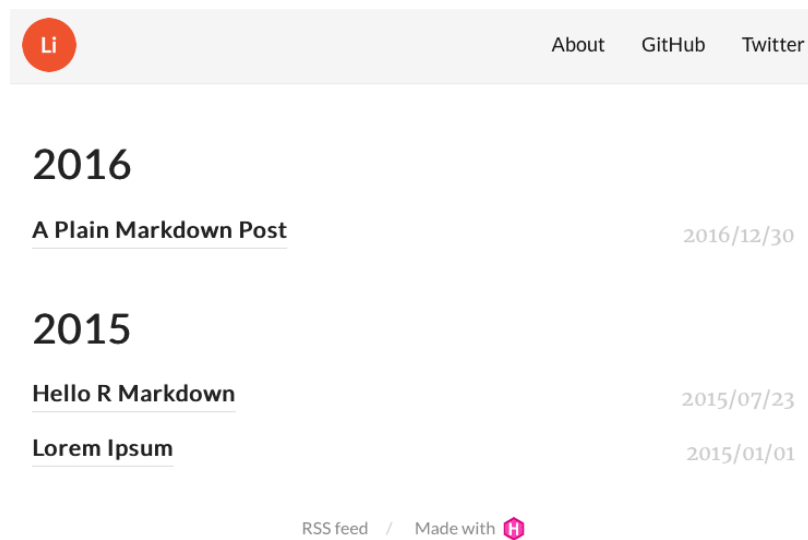
<sup>2</sup>En R, compruebe la salida de `list.files('.')` y asegúrese de que no incluya archivos distintos a `LICENSE`, el archivo de proyecto de RStudio (`*.Rproj`), `README` o `README.md`.

<sup>3</sup>Hasta que configure su sitio web para ser implementado, LiveReload solo actualiza la versión *local* de su sitio web. Esta versión solo es visible para usted. Para que su sitio web pueda buscarse, descubrirse y vivir en Internet, tendrá que cargar los archivos de su sitio web en un creador de sitios. Consulte el capítulo ?? para obtener más detalles.

<sup>4</sup>También puede pensar en RStudio Viewer como un navegador web.

está basado en el paquete de R **servr** (Xie, 2018c) de manera predeterminada.<sup>5</sup>

La función `new_site()` tiene varios argumentos, y puede revisar su página de ayuda de R (`?blogdown::new_site`) para más detalles. Un tema predeterminado mínimo llamado “hugo-lithium-theme” se proporciona como el tema predeterminado del nuevo sitio,<sup>6</sup> Y se puede ver cómo se ve en Figure 1.1.



**FIGURE 1.1:** La página de inicio del nuevo sitio por defecto.

Tiene que saber tres conceptos más básicos para un sitio web basado en Hugo:

1. El archivo de configuración `config.toml`, en el que puede especificar algunas configuraciones globales para su sitio. Incluso si no sabe qué es TOML en este momento (se presentará en el capítulo 2), aún podrá cambiar algunas

<sup>5</sup>Hugo tiene su propia implementación LiveReload. Si desea aprovecharlo, puede establecer la opción global `options(blogdown.generator.server = TRUE)`. Ver la sección D.2 para más información.

<sup>6</sup>Puede encontrar su código fuente en GitHub: <https://github.com/yihui/hugo-lithium-theme>. Este tema se bifurcó de <https://github.com/jrutherford/hugo-lithium-theme> y se modificó para que funcione mejor con **blogdown**.

configuraciones obvias. Por ejemplo, puede ver configuraciones como estas en `config.toml`:

```
baseUrl = "/"
languageCode = "en-us"
title = "A Hugo website"
theme = "hugo-lithium-theme"

[[menu.main]]
    name = "About"
    url = "/about/"
[[menu.main]]
    name = "GitHub"
    url = "https://github.com/rstudio/blogdown"
[[menu.main]]
    name = "Twitter"
    url = "https://twitter.com/rstudio"
```

Puede cambiar el título de la página web, e.g., `title = "Mi propia página web chévere"`, y actualizar las URL de GitHub y Twitter.

2. El directorio de contenido (por defecto, `content/`). Aquí es donde usted escribe los archivos de origen R Markdown o Markdown para sus publicaciones y páginas. Bajo `content/` del sitio predeterminado, puede ver `about.md` y un directorio `post/` que contiene algunas publicaciones. La organización del directorio de contenido depende de usted. Puede tener archivos y directorios arbitrarios allí, según la estructura del sitio web que desee.
3. El directorio de publicación (por defecto, `public/`). Su sitio web se generará en este directorio, lo que significa que no necesita agregar manualmente ningún archivo a este directorio.<sup>7</sup> Por lo general, contiene una gran cantidad de

---

<sup>7</sup>Ejecutando `serve_site()` o `build_site()`, los archivos se generarán y publicarán en su directorio de publicación automáticamente.

archivos `*.html` y dependencias como `*.css`, `*.js` e imágenes. Puede cargar todo en `public/` a cualquier servidor web que pueda publicar sitios web estáticos, y su sitio web estará en funcionamiento. Hay muchas opciones para publicar sitios web estáticos, y hablaremos más sobre ellos en el capítulo ?? si no está familiarizado con la implementación de sitios web.

Si está satisfecho con este tema predeterminado, ¡está básicamente listo para comenzar a escribir y publicar su nuevo sitio web! Mostraremos cómo usar otros temas en la sección 1.6. Sin embargo, tenga en cuenta que un tema más complicado y elegante puede requerir que aprenda más sobre todas las tecnologías subyacentes, como el lenguaje de plantillas de Hugo, HTML, CSS y JavaScript.

---

### 1.3 RStudio IDE

Hay algunos complementos básicos de RStudio para facilitar la edición y la vista previa de su sitio web, y puede encontrarlos en el menú “Addins” en la barra de herramientas de RStudio:

- “Serve Site”: este complemento llama a `blogdown::serve_site()` para presentar continuamente su sitio web localmente utilizando la tecnología LiveReload, para que pueda ver en vivo el sitio web. Puede seguir editando material para su sitio mientras lo está viendo, pero esta función bloqueará su consola de R de manera predeterminada, lo que significa que no podrá usar su consola de R una vez que inicie este servidor web local. Para desbloquear la consola, haga clic en el signo de stop rojo en la esquina superior derecha de la ventana de la consola. Si prefiere evitar este comportamiento por completo, establezca la opción `options(servr.daemon = TRUE)`, antes de hacer clic en este complemento o llame a la función `serve_site()`, para que el servidor sea



demonizado y no bloquee su consola de R.<sup>^</sup> [Hemos oído de casos en los que el servidor demonizado bloquea R en Windows. Si tiene problemas con el servidor daemonizado, existen tres soluciones alternativas, y puede probar una de ellas: (1) instalar el paquete **later** a través de `install.packages("later")` y volver a iniciar el servidor; (2) use el servidor de Hugo (vea la sección D.2); (3) llame `blogdown::serve_site()` en una sesión de R separada, y puede obtener una vista previa de su sitio web en su navegador web, pero aún puede editar el sitio web en RStudio.]

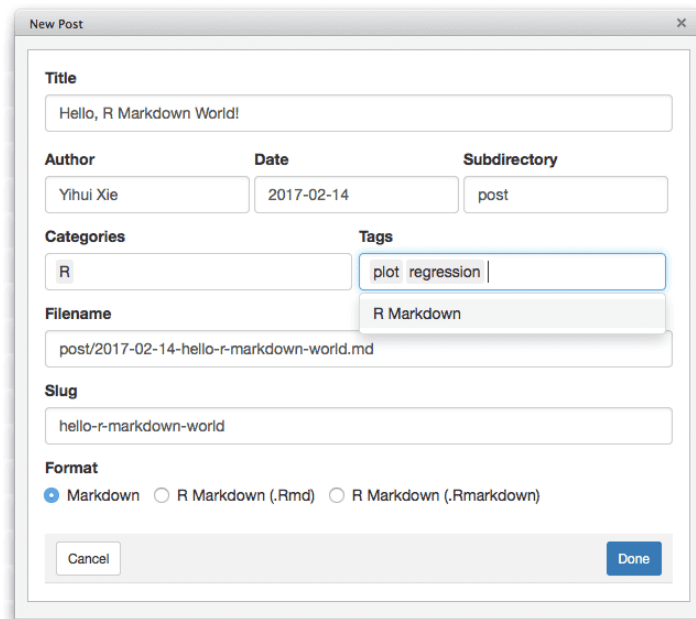
- “New Post”: este complemento proporciona un cuadro de diálogo para que ingrese los metadatos de la publicación de su blog, incluidos el título, el autor, la fecha, etc. Ver la Figura 1.2 para un ejemplo. Este complemento realmente llama a la función `blogdown::new_post()`, pero hace algunas cosas automáticamente:
  - A medida que escribe el título de la publicación, generará un nombre de archivo para usted, y puede editarlo si no le gusta el generado automáticamente. De hecho, también puede usar este complemento para crear páginas normales en cualquier directorio bajo `content/`. Por ejemplo, si desea agregar una página de currículum, puede cambiar el nombre del archivo a `resume.md` del `'post/YYYY-mm-dd-resume.md'` predeterminado.
  - Puede seleccionar la fecha desde un widget de calendario proporcionado por Shiny.<sup>8</sup>
  - Esto escaneará las categorías y etiquetas de las publicaciones existentes, por lo que cuando quiera ingresar categorías o etiquetas, puede seleccionarlas de los menús desplegables o crear otras nuevas.
  - Después de crear una nueva publicación, se abrirá automáti-

---

<sup>8</sup>Shiny es un paquete R para crear aplicaciones web interactivas usando R. Usando este complemento, el widget de calendario le permite ver un calendario interactivo por mes para seleccionar fechas. Este es un uso simple de Shiny, pero puede leer más acerca de las aplicaciones Shiny aquí: <https://shiny.rstudio.com>.

camente, por lo que puede comenzar a escribir el contenido de inmediato.

- “Update Metadata”: Este complemento le permite actualizar los metadatos YAML de la publicación abierta actualmente. Ver la Figura @ref(fig: update-meta) para un ejemplo. La principal ventaja de este complemento es que puede seleccionar categorías y etiquetas de los menús desplegables en lugar de tener que recordarlas.



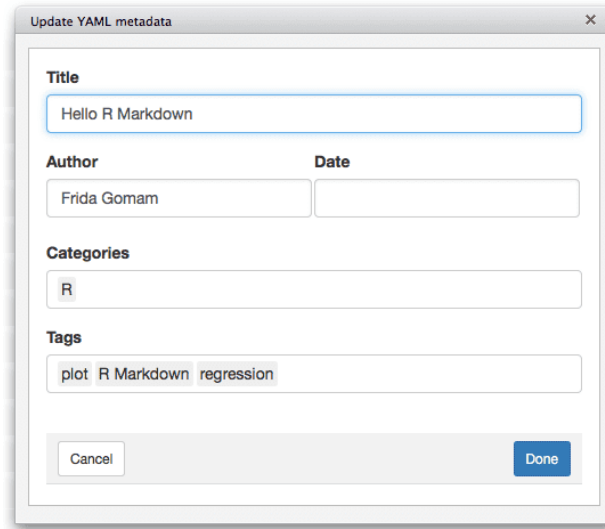
The image shows a "New Post" dialog box from RStudio. It contains the following fields and options:

- Title:** Hello, R Markdown World!
- Author:** Yihui Xie
- Date:** 2017-02-14
- Subdirectory:** post
- Categories:** R
- Tags:** plot regression
- Filename:** post/2017-02-14-hello-r-markdown-world.md
- Slug:** hello-r-markdown-world
- Format:** ☒ Markdown ☐ R Markdown (.Rmd) ☐ R Markdown (.Rmarkdown)

At the bottom, there are "Cancel" and "Done" buttons.

**FIGURE 1.2:** Crear una nueva publicación usando el complemento de RStudio.

Con estos complementos, rara vez deberá ejecutar los comandos en R manualmente después de haber configurado su sitio web, ya que todas sus publicaciones se compilarán automáticamente cada vez que cree una nueva publicación o modifique una existente debido a la función LiveReload.



**FIGURE 1.3:** Actualizar los metadatos de una publicación existente usando el complemento de RStudio.

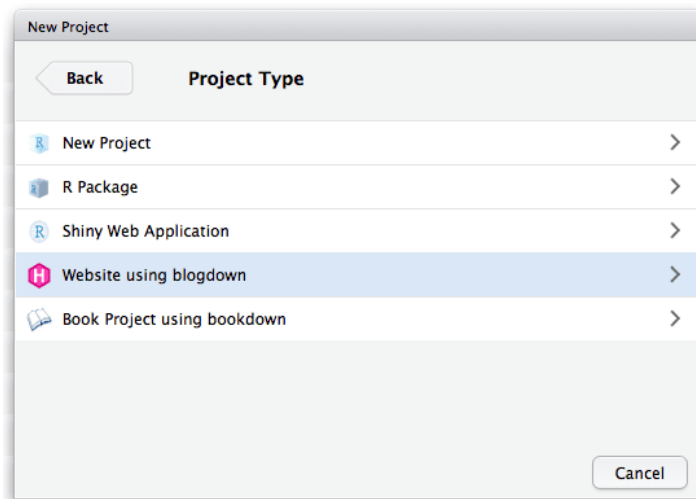
Si su versión de RStudio es por lo menos la v1.1.383,<sup>9</sup> puede actualmente crear un proyecto de página web directamente desde el menú `File -> New Project -> New Directory` (vea la Figura 1.4 y 1.5).

Si su sitio web se creó utilizando la función `blogdown::new_site()` en lugar del menú de RStudio por primera vez, puede salir de RStudio y volver a abrir el proyecto. Si accede al menú `Tools -> Project Options`, su tipo de proyecto debería ser “Website” como lo puede ver en la Figura 1.6.

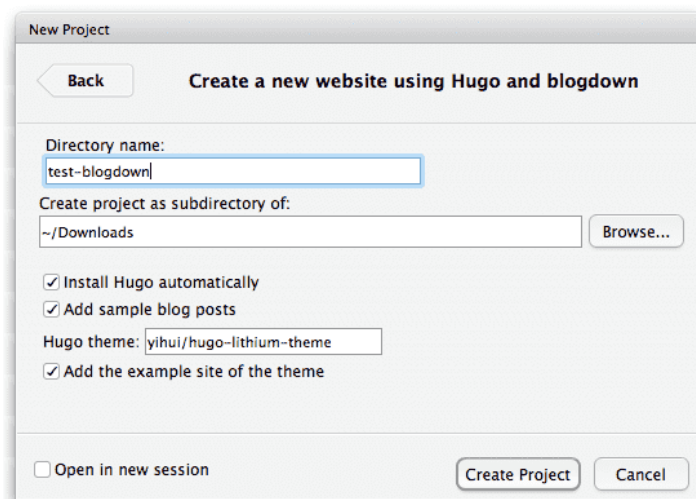
Luego verá un panel en RStudio llamado “Build” y hay un botón “Build Website”. Al hacer clic en este botón, RStudio llamará a `blogdown::build_site()` para construir el sitio web. Esto generará automáticamente archivos en el directorio `public/`.<sup>10</sup> Si desea compilar el sitio web y publicar los archivos de salida en `public/` man-

<sup>9</sup>Puede descargar todas las versiones del sitio oficial de RStudio incluyendo la v1.1.383 desde <https://www.rstudio.com/products/rstudio/download/>.

<sup>10</sup>O donde sea que esté ubicado su directorio de publicación. Es `public/` de forma predeterminada, pero se puede cambiar especificando `publishDir = "myNewDirectory"` en el archivo `config.toml`.



**FIGURE 1.4:** Crear un nuevo proyecto de página web en RStudio.



**FIGURE 1.5:** Crear un proyecto de página web basado en blogdown.

ualmente, se recomienda reiniciar su sesión de R y hacer clic en este botón “Build Website” antes de publicar el sitio web, en lugar de publicar la carpeta `public/` generada de forma continua y automática por `blogdown::serve_site()`, porque este último llama a `blogdown::build_site(local = TRUE)`, que tiene algunas diferencias sutiles con `blogdown::build_site(local = FALSE)` (ver la sección [D.3](#) para más detalles).

Recomendamos mucho que desmarque la opción “Preview site after building” en las opciones de proyecto de RStudio (Figura [1.6](#)).<sup>11</sup> También puede desmarcar la opción “Re-knit current preview when supporting files change”, ya que esta opción no es realmente útil después de llamar a `serve_site()`.

---

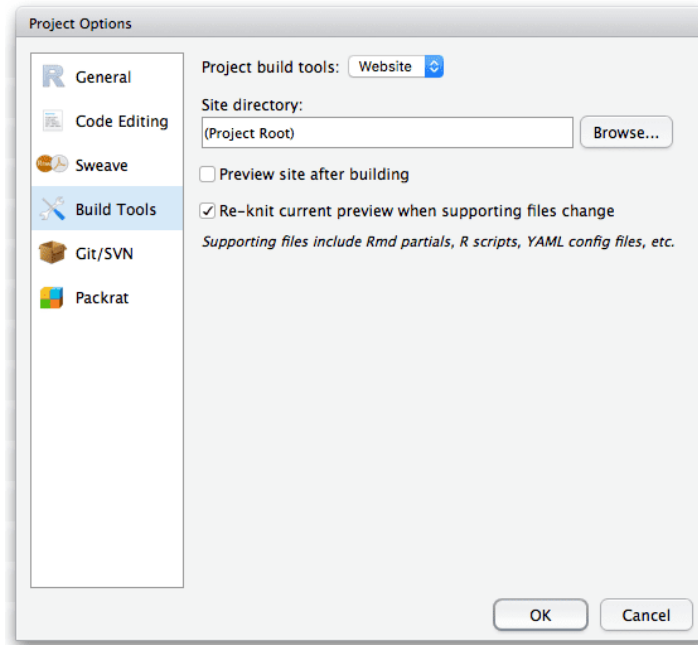
## 1.4 Opciones globales

Dependiendo de sus preferencias personales, puede establecer algunas opciones globales antes de trabajar en su sitio web. Estas opciones se deben configurar usando `options(name = value)`, y las opciones disponibles actualmente se presentan en [Table 1.1](#).

Le recomendamos que configure estas opciones en su archivo de perfil de inicio de R. Puede consultar la página de ayuda `?Rprofile` para más detalles, y aquí hay una introducción simplificada. Un archivo de perfil de inicio es básicamente un script en R que se ejecuta cuando se inicia la sesión de R. Este es un lugar perfecto para establecer opciones globales, por lo que no necesita escribir estas opciones nuevamente cada vez que inicie una nueva sesión

---

<sup>11</sup>En caso de que se pregunte por qué: a menos que haya establecido la opción `relativeurls` a `true` en `config.toml`, requiere un servidor web para obtener una vista previa del sitio local, de lo contrario, incluso si puede ver la página de inicio de su sitio web en RStudio Viewer, la mayoría de los enlaces como los enlaces a archivos CSS y JavaScript son poco probables que funcionen. Cuando RStudio Viewer le muestra la vista previa, en realidad no ejecuta un servidor web.



**FIGURE 1.6:** Opciones de proyecto de RStudio.

**TABLE 1.1:** Opciones globales que afectan el comportamiento de blogdown.

Option name	Default	Meaning
servr.daemon	FALSE	Si debe usar un servidor demonizado
blogdown.author		El autor por defecto de nuevas publicaciones
blogdown.ext	.md	Extensión por defecto de nuevas publicaciones
blogdown.subdir	post	Un subdirectorio bajo content/
blogdown.yaml.empty	TRUE	Preservar campos vacíos en YAML?

en R. Puede usar un archivo de perfil global `~/.Rprofile`,<sup>12</sup> O un archivo por proyecto `.Rprofile` en el directorio raíz de su proyecto de RStudio. El primero se aplicará a todas las sesiones de R que inicie, a menos que haya proporcionado el último para anularlo. La forma más fácil de crear un archivo de este tipo es usar `file.edit()` en RStudio, por ejemplo,

```
file.edit('~/.Rprofile')  
# o file.edit('.Rprofile')
```

Supongamos que siempre prefiere el servidor demonizado y quiere que el autor de las nuevas publicaciones sea “John Doe” de manera predeterminada. Puede establecer estas opciones en el archivo de perfil:

```
options(servr.daemon = TRUE, blogdown.author = 'John Doe')
```

Una buena consecuencia de establecer estas opciones es que cuando usa el complemento de RStudio “New post”, los campos “Author”, “Subdirectory” y “Format” se completarán automáticamente, por lo que no tendrá que manipularlos todas las veces a menos que desea cambiar los valores predeterminados (ocasionalmente).

R solo lee un archivo de perfil de inicio. Por ejemplo, si tiene un `.Rprofile` en el directorio actual y un `~/.Rprofile` global, solo el anterior se ejecutará cuando R se inicie desde el directorio actual. Esto puede hacer que sea inconveniente para varios autores que colaboran en el mismo proyecto de un sitio web, ya que no puede establecer opciones específicas del autor. En particular, no es posible establecer la opción `blogdown.author` en un solo `.Rprofile`, porque esta opción debería ser diferente para diferentes autores. Una solución consiste en establecer opciones comunes en `.Rprofile` bajo del directorio raíz del proyecto del sitio web, y también ejecutar el

---

<sup>12</sup>La tilde `~` indica el directorio principal en su sistema.

~/Rprofile global si existe. Las opciones específicas del autor se pueden establecer en el ~/Rprofile global en la computadora de cada autor.

```
# en el .Rprofile del proyecto de la página web
if (file.exists('~/Rprofile')) {
  base::sys.source('~/Rprofile', envir = environment())
}
# luego configure options(blogdown.author = 'Your Name') en ~/Rprofile
```

---

## 1.5 R Markdown vs. Markdown

Si no está familiarizado con R Markdown , consulte el Apéndice [A](#) para obtener un tutorial rápido. Cuando crea una nueva publicación, debe decidir si desea usar R Markdown o Markdown simple , como puede ver en [Figure 1.2](#). Las principales diferencias son:

1. No puede ejecutar ningún código en R en un documento de Markdown simple, mientras que en un documento de Markdown R, puede incrustar fragmentos de código R (`{r}`). Sin embargo, aún puede incrustar código de R en Markdown simple usando la sintaxis para bloques de código delimitados ````r` (tenga en cuenta que no hay llaves {}). Tales bloques de código no se ejecutarán y pueden ser adecuados para propósitos de demostración pura. A continuación se muestra un ejemplo de un fragmento de código de R en R Markdown:

```
```{r cool-plot, fig.width='80%', fig.cap='A cool plot.'}
plot(cars, pch = 20) # no es muy chévere
```
```



Y aquí hay un ejemplo de un bloque de código de R en Markdown simple:

```
```r
1 + 1 # no ejecutada
```
```

2. Una publicación en Markdown simple es ejecutada en HTML a través de Blackfriday<sup>13</sup> (un paquete escrito en lenguaje Go y adoptado por Hugo). Un documento R Markdown se compila a través de los paquetes **rmarkdown**, **bookdown**, y Pandoc, lo que significa que puede usar la mayoría de las características de Markdown de Pandoc<sup>14</sup> y extensiones de Markdown para **bookdown**<sup>15</sup> en **blogdown**. Si usa R Markdown (Allaire et al., 2018) con **blogdown**, le recomendamos que lea la documentación de Pandoc y **bookdown** al menos una vez para conocer todas las características posibles. No repetiremos los detalles en este libro, pero enumeraremos las características brevemente a continuación, que también se muestran en el sitio web de ejemplo: <https://blogdown-demo.rbind.io>.

- Formateo en línea: texto en *italica* / **negrita** y `código en línea`.
- Elementos en línea: subíndices (e.g., H<sub>2</sub>O) y superíndices (e.g., R<sup>2</sup>); links ([texto](url)) e imágenes ![título](url); notas al pie texto^[nota al pie].
- Elementos de nivel bloque: párrafos; encabezados de sección numerados y no numerados; listas ordenadas y no ordenadas; citas en bloque; bloques de código; tablas; reglas horizontales.

---

<sup>13</sup><https://gohugo.io/overview/configuration/>

<sup>14</sup><http://pandoc.org/MANUAL.html#pandocs-markdown>

<sup>15</sup><https://bookdown.org/yihui/bookdown/components.html>

- Expresiones matemáticas y ecuaciones.
- Teoremas y demostraciones.
- Bloques de código en R que se pueden usar para producir salida de texto (incluidas tablas) y gráficos. Tenga en cuenta que las ecuaciones, teoremas, tablas y figuras se pueden numerar y referenciadas cruzadamente.
- Citas y bibliografía.
- HTML widgets, y aplicaciones en Shiny incrustadas mediante `<iframe>`.

Hay muchas diferencias en la sintaxis entre el Markdown de Blackfriday y el Markdown de Pandoc. Por ejemplo, puede escribir una lista de tareas con Blackfriday, pero no con Pandoc:

```
- [x] Escribir un paquete en R.
- [ ] Escribir un libro.
- [ ] ...
- [ ] Beneficio!
```

Del mismo modo, Blackfriday no admite matemática en LaTeX y Pandoc sí. Hemos agregado el soporte MathJax<sup>16</sup> MathJax al tema predeterminado (hugo-lithium-theme<sup>17</sup> en **blogdown** para compilar matemática en LaTeX en páginas HTML, pero hay una advertencia para las publicaciones simples de Markdown: debe incluir expresiones matemáticas en línea con un par de comillas ``$math$``, por ejemplo, ``$s_n = \sum_{i=1}^n x_i$``. Del mismo modo, las expresiones matemáticas del estilo de visualización deben escribirse en ``$$math$$``. Para las publicaciones de R Markdown, puede usar `$math$` para expresiones matemáticas en línea, y `$$math$$` para expresiones de estilo de visualización.<sup>18</sup>

<sup>16</sup><https://www.mathjax.org/#docs>

<sup>17</sup><https://github.com/yihui/hugo-lithium-theme>

<sup>18</sup>El motivo por el que necesitamos los respaldos para documentos de Mark-

Si considera que es un dolor tener que recordar las diferencias entre R Markdown y Markdown, una opción conservadora es usar siempre R Markdown, incluso si su documento no contiene ningún fragmento de código en R. Markdown de Pandoc es mucho más rico que Blackfriday, y solo hay un pequeño número de características no disponibles en Pandoc pero presentes en Blackfriday. Las principales desventajas de usar R Markdown son:

1. Puede sacrificar algo de velocidad en la renderización del sitio web, pero esto puede no ser notorio debido a un mecanismo de almacenamiento en caché en **blogdown** (lea más sobre esto en la sección D.3). Hugo es muy rápido cuando procesa archivos de Markdown simples, y típicamente debería tomar menos de un segundo para renderizar unos cientos de archivos de Markdown.
2. Tendrá algunos archivos HTML intermedios en el directorio fuente de su sitio web, porque **blogdown** tiene que llamar a **rmarkdown** para renderizar previamente los archivos `*.Rmd` `*.html`. También tendrá carpetas intermedias para las figuras (`*_files/`) y la memoria caché (`*_cache/`) si tiene una salida de trazado en fragmentos de código en R o ha habilitado el almacenamiento en cache de **knitr**. A menos que le importe mucho la “limpieza” del repositorio fuente de su sitio web (especialmente cuando usa una herramienta de control de versiones como GIT), estos archivos intermedios no deberían importar.

En este libro, generalmente nos referimos a los archivos `.Rmd` cuando decimos “Documentos de R Markdown”, que se compilan a `.html` de forma predeterminada. Sin embargo, hay otro tipo de documento de R Markdown con la extensión de nombre de archivo

---

down simples es que tenemos que evitar que Blackfriday interprete el código LaTeX como Markdown. Las comillas asegurarán que el contenido interno no se traduzca como Markdown a HTML, por ejemplo, ``$$x *y* z$$`` se convertirá en `<code> $$x *y* z$$</code>`. Sin las comillas, se convertirá en `$$x <em>y</em> z$$`, que no es una expresión matemática en LaTeX válida para MathJax. Problemas similares pueden surgir cuando tenga otros caracteres especiales como guiones bajos en sus expresiones matemáticas.

`.Rmarkdown`. Dichos documentos de R Markdown se compilan para los documentos Markdown con la extensión `.markdown`, que serán procesados por Hugo en lugar de por Pandoc. Hay dos limitaciones principales de usar `.Rmarkdown` en comparación con `.Rmd`:

- No puede usar las funciones de reducción solo compatibles con Pandoc, como las citas. Las expresiones matemáticas solo funcionan si ha instalado el paquete **xaringan** (Xie, 2018d) y ha aplicado la solución de JavaScript mencionada en la sección B.3.
- Los widgets HTML no son compatibles.

La principal ventaja de usar `.Rmarkdown` es que los archivos de salida son más limpios porque son archivos Markdown. Puede ser más fácil para usted leer la salida de sus publicaciones sin mirar las páginas web reales renderizadas. Esto puede ser particularmente útil al revisar los pull requests de GitHub. Tenga en cuenta que las tablas, figuras, ecuaciones y teoremas numerados también son compatibles. No puede usar directamente la sintaxis de Markdown en las leyendas de tabla o figura, pero puede usar referencias de texto como una solución alternativa (consulte la documentación de **bookdown**).

Para cualquier documento de R Markdown (no específico de **blogdown**), debe especificar un formato de salida. Hay muchos posibles formatos de salida<sup>19</sup> en el paquete **rmarkdown** (como `html_document` y `pdf_document`) y otros paquetes de extensión (tales como `tufte::tufte_html` y `bookdown::gitbook`). Por supuesto, el formato de salida para los sitios web debe ser HTML. Hemos proporcionado una función de formato de salida `blogdown::html_page` en **blogdown**, y todos los archivos R Markdown se renderizan con este formato. Se basa en el formato de salida `bookdown::html_document2`, lo que significa que ha heredado muchas características de **bookdown** además de las características en Pandoc. Por ejemplo, puede numerar y hacer referencias cruzadas de ecuaciones matemáticas, figuras, tablas y teoremas, etc. Consulte

<sup>19</sup><http://rmarkdown.rstudio.com/lesson-9.html>

el Capítulo 2 del libro **bookdown** (Xie, 2016) para obtener más detalles sobre la sintaxis.

Note que el formato de salida `bookdown::html_document2` a su vez hereda de `rmarkdown::html_document`, entonces necesita ver la página de ayuda `?rmarkdown::html_document` para todas las opciones posibles para el formato `blogdown::html_page`. Si desea cambiar los valores predeterminados de las opciones de este formato de salida, puede agregar un campo `output` a sus metadatos YAML. Por ejemplo, podemos agregar una tabla de contenido a una página, establecer el ancho de la figura en 6 pulgadas y usar el dispositivo `svg` para los gráficos estableciendo estas opciones en YAML:

```
---
title: "Mi grandiosa publicación"
author: "John Doe"
date: "2017-02-14"
output:
  blogdown::html_page:
    toc: true
    fig_width: 6
    dev: "svg"
---
```

Para establecer opciones para `blogdown::html_page()` globalmente (es decir, aplicar ciertas opciones a todos los archivos Rmd), puede crear un archivo `_output.yml` en el directorio raíz de su sitio web. Este archivo YAML debe contener el formato de salida directamente (no coloque el formato de salida bajo la opción `output`), por ejemplo,

```
blogdown::html_page:
  toc: true
  fig_width: 6
  dev: "svg"
```

Por el momento, no todas las funciones de `rmarkdown::html_document` son compatibles con **blogdown**, como `df_print`, `code_folding`, `code_download`, etc.

Si su trozo de código tiene salida de gráficos, le recomendamos que evite caracteres especiales como espacios en la etiqueta de fragmentos. Lo ideal es que solo use caracteres alfanuméricos y guiones, por ejemplo, ```{r, my-label}` en lugar de ```{r, my label}`.

No se recomienda cambiar las opciones **knitr** chunk `fig.path` o `cache.path` en R Markdown. Los valores predeterminados de estas opciones funcionan mejor con **blogdown**. Lea la sección D.5 para conocer los motivos técnicos, si lo prefiere.

Si está trabajando en una publicación de R Markdown, pero no quiere que **blogdown** la compile, puede cambiar temporalmente su extensión de nombre de archivo de `.Rmd` a otra extensión desconocida como `.Rmkd`.

---

## 1.6 Otros temas

En Hugo, los temas controlan toda la apariencia y funcionalidad de su sitio. Entonces, si le importa mucho el aspecto de su sitio web, probablemente pasará bastante tiempo al principio buscando un tema de Hugo que le guste de la colección que figura en <http://themes.gohugo.io>. Tenga en cuenta que no todos los temas se han probado en **blogdown**. Si encuentra que un determinado tema no funciona bien con **blogdown**, puede informar a <https://github.com/rstudio/blogdown/issues>, e intentaremos investigar el motivo, pero puede ser una cuestión de tiempo aprender y comprender cómo funciona un nuevo tema, por lo que le recomendamos que aprenda más acerca de Hugo por su cuenta antes de preguntar, y también alentamos a los usuarios a ayudarse mutuamente allí.

Después de haber encontrado un tema satisfactorio, debe averiguar

su nombre de usuario y el nombre del repositorio de GitHub,<sup>20</sup> luego instale el tema a través de `blogdown::install_theme()`, o simplemente cree un nuevo sitio bajo otro directorio nuevo y pase el nombre del repositorio de GitHub al argumento `theme` de `new_site()`. Recomendamos que use el segundo enfoque, porque los temas de Hugo podrían ser muy complicados y el uso de cada tema puede ser muy diferente y muy dependiente del `config.toml`. Si instala un tema con `install_theme()` en lugar de `new_site()`, deberá crear manualmente el archivo `config.toml` en el directorio raíz de su sitio web para que coincida con el tema recién instalado.<sup>21</sup>

```
# por ejemplo, cree un sitio nuevo con el tema academic
blogdown::new_site(theme = 'gcushen/hugo-academic')
```

Para ahorrarle tiempo, enumeramos algunos temas a continuación que coinciden con nuestro gusto:

- Temas Simples/mínimos: XMin,<sup>22</sup> Tanka,<sup>23</sup> simple-a,<sup>24</sup> and ghostwriter.<sup>25</sup>
- Temas sofisticados: hugo-academic<sup>26</sup> (fuertemente recomendado para usuarios de la academia), hugo-tranquilpeak-theme,<sup>27</sup> hugo-creative-portfolio-theme,<sup>28</sup> and hugo-universal-theme.<sup>29</sup>

<sup>20</sup>Para la mayoría de los temas, puede encontrar esto navegando al tema de su elección desde <http://themes.gohugo.io> y luego haciendo clic en Homepage.

<sup>21</sup>Una solución alternativa, si usó `install_theme()` y establece el argumento `theme_example` en `TRUE`, entonces puede acceder a un archivo `config.toml` de ejemplo. En el directorio `themes/`, vaya al archivo del tema que acaba de descargar y busque `exampleSite/config.toml`. Este archivo puede copiarse en su directorio raíz (para reemplazar el archivo `config.toml` de su tema original) o usarse como una plantilla para escribir correctamente un nuevo archivo `config.toml` para su nuevo tema.

<sup>22</sup><https://github.com/yihui/hugo-xmin>

<sup>23</sup><https://github.com/road2stat/hugo-tanka>

<sup>24</sup><https://github.com/AlexFinn/simple-a>

<sup>25</sup><https://github.com/jbub/ghostwriter>

<sup>26</sup><https://github.com/gcushen/hugo-academic>

<sup>27</sup><https://github.com/kakawait/hugo-tranquilpeak-theme>

<sup>28</sup><https://github.com/kishaningithub/hugo-creative-portfolio-theme>

<sup>29</sup><https://github.com/devcows/hugo-universal-theme>

- Temas que contienen multimedia: Si está interesado en agregar contenido multimedia a su sitio (como archivos de audio de un podcast), el tema *castanet*<sup>30</sup> proporciona un excelente marco adaptado para esta aplicación. Un ejemplo de un sitio que usa **blogdown** con el tema *castanet* es *R-Podcast*<sup>31</sup>

Si no entiende HTML, CSS o JavaScript, y no tiene experiencia con los temas o plantillas de Hugo, puede tardar unos 10 minutos en comenzar a usar su nuevo sitio web, ya que debe aceptar todo lo que le ofrecen (como el tema predeterminado); Si tiene el conocimiento y la experiencia (y desea personalizar su sitio al máximo), puede tardar varios días en comenzar. Hugo es realmente poderoso. Tenga cuidado con el poder.

Otra cosa a tener en cuenta es que cuanto más esfuerzo hagas en un tema complicado, más difícil será cambiar a otros temas en el futuro, porque es posible que haya personalizado muchas cosas que no son fáciles de transferir a otro tema. Por lo tanto, pregúntese seriamente: “¿Me gusta tanto este tema tan elegante que definitivamente no lo cambiaré en los próximos años?”.

---

Si elige cavar un hoyo bastante profundo, algún día no tendrá más remedio que seguir cavando, incluso con lágrimas.

— Liyun Chen<sup>32</sup>

---

<sup>30</sup><https://github.com/mattstratton/castanet>

<sup>31</sup><https://www.r-podcast.org>

<sup>32</sup>Traducido de su weibo Chino: <http://weibo.com/1406511850/Dhrb4toHc> (no puede ver esta página a menos que haya iniciado sesión).



---

## 1.7 Un flujo de trabajo recomendado

Hay muchas maneras de comenzar a construir un sitio web y presentarlo. Debido a la gran cantidad de tecnologías que necesita aprender para comprender completamente cómo funciona un sitio web, nos gustaría recomendar un flujo de trabajo a los principiantes, por lo que es de esperar que no necesiten digerir el resto de este libro. Definitivamente este no es el flujo de trabajo más óptimo, pero requiere que conozca la menor cantidad de detalles técnicos.

Para comenzar un nuevo sitio web:

1. Elija cuidadosamente un tema en <http://themes.gohugo.io>, y encuentre el enlace a su repositorio GitHub, que tiene la forma `https://github.com/user/repo`.
2. Cree un nuevo proyecto en RStudio y escriba el código `blogdown::new_site (theme = 'user/repo')` en la consola R, donde `user/repo` proviene del enlace en el paso 1.
3. Juegue con el nuevo sitio por un tiempo y si no le gusta, puede repetir los pasos anteriores, de lo contrario edite las opciones en `config.toml`. Si no comprende ciertas opciones, vaya a la documentación del tema, que a menudo es la página README del repositorio de GitHub. No todas las opciones tienen que ser cambiadas.

Para editar una página web:

1. Establezca `options(servr.daemon = TRUE)` a menos que ya lo haya configurado en `.Rprofile`. Si esta opción no funciona para usted (por ejemplo, bloquea su sesión en R), consulte la sección 1.4 para obtener una solución alternativa.
2. Haga clic en el complemento de RStudio “Serve Site” para obtener una vista previa del sitio en RStudio Viewer. Esto solo debe hacerse una vez cada vez que abra el proyecto

RStudio o reinicie su sesión en R. No haga clic en el botón knit en la barra de herramientas de RStudio.

3. Use el complemento “New Post” para crear una nueva publicación o página, luego empiece a escribir el contenido.
4. Use el complemento “Update Metadata” para modificar los metadatos del YAML, si es necesario.

Para publicar un sitio web, si no está familiarizado con GIT o GitHub:

1. Reinicie la sesión de R, y ejecute `blogdown::hugo_build()`. Debería obtener un directorio `public/` bajo el directorio raíz de su proyecto.
2. Inicie sesión en <https://www.netlify.com> (puede usar una cuenta de GitHub, si la tiene). Si esta es la primera vez que publica este sitio web, puede crear un sitio nuevo; de lo contrario, puede actualizar el sitio existente que creó la última vez. Puede arrastrar y soltar la carpeta `public/` desde su visor de archivos al área indicada en la página web de Netlify, donde dice “Drag a folder with a static site here”.
3. Espere unos segundos para que Netlify despliegue los archivos y le asignará un subdominio aleatorio de la forma `random-word-12345.netlify.com`. Puede (y debería) cambiar este subdominio aleatorio a uno más significativo si todavía está disponible.

Puede ser mucho más fácil publicar un sitio web si está familiarizado con GIT y GitHub. Recomendamos que cree un nuevo sitio en Netlify desde su repositorio de GitHub que contenga los archivos fuente de su sitio web, para que pueda disfrutar los beneficios de la implementación continua en lugar de cargar manualmente la carpeta `public/` cada vez. Con este enfoque, no es necesario ejecutar `blogdown::hugo_build()` localmente, ya que el sitio web se puede construir en Netlify a través de Hugo. Consulte el capítulo ?? para obtener más información.

## 2

---

### *Hugo*

---

En este capítulo, presentaremos brevemente Hugo (<https://gohugo.io>), el generador de sitios estáticos en el que se basa **blogdown**. Este capítulo no pretende reemplazar la documentación oficial de Hugo, sino proporcionar una guía para aquellos que recién están comenzando con Hugo. En caso de duda, consulte la documentación oficial de Hugo.

---

#### 2.1 Sitios estáticos y Hugo

Un sitio estático a menudo consiste en archivos HTML (con dependencias externas opcionales como imágenes y bibliotecas de JavaScript), y el servidor web envía exactamente el mismo contenido al navegador web sin importar quién visita las páginas web. No hay computación dinámica en el servidor cuando se solicita una página. En contraste, un sitio dinámico se basa en un lenguaje del lado del servidor para hacer cierta informática y envía contenido potencialmente diferente dependiendo de las diferentes condiciones. Un lenguaje común es PHP, y un ejemplo típico de un sitio dinámico es un foro web. Por ejemplo, cada usuario tiene una página de perfil, pero generalmente esto no significa que el servidor haya almacenado una página de perfil HTML diferente para cada usuario. En cambio, el servidor obtendrá los datos del usuario de una base de datos y renderizará la página de perfil de forma dinámica.

Para un sitio estático, cada URL que visita a menudo tiene un archivo HTML correspondiente almacenado en el servidor, por lo

que no es necesario calcular nada antes de presentar el archivo a los visitantes. Esto significa que los sitios estáticos tienden a ser más rápidos en tiempo de respuesta que los sitios dinámicos, y también son mucho más fáciles de implementar, ya que la implementación simplemente significa copiar archivos estáticos a un servidor. Un sitio dinámico a menudo se basa en bases de datos, y tendrá que instalar más paquetes de software para presentar un sitio dinámico. Para obtener más ventajas de los sitios estáticos, lea la introducción<sup>1</sup> en el sitio web de Hugo.

Existen muchos generadores de sitios estáticos existentes, incluyendo Hugo, Jekyll,<sup>2</sup> y Hexo,<sup>3</sup> etc. La mayoría de ellos puede construir sitios web de propósito general, pero a menudo se utilizan para construir blogs.

Amamos a Hugo por muchas razones, pero hay algunas que se destacan. A diferencia de otros generadores de sitios estáticos, la instalación de Hugo es muy simple porque proporciona un único archivo ejecutable sin dependencias para la mayoría de los sistemas operativos (consulte la sección ??). También se diseñó para procesar cientos de páginas de contenido más rápido que los generadores de sitios estáticos comparables y, según los informes, puede presentar una página en aproximadamente 1 milisegundo. Por último, la comunidad de usuarios de Hugo es muy activa tanto en el foro de discusión de Hugo<sup>4</sup> y en los issues de GitHub.<sup>5</sup>

Aunque creemos que Hugo es un fantástico generador de sitios estáticos, en realidad hay una única característica importante que falta: el soporte para R Markdown. Ese es básicamente el objetivo del paquete **blogdown**.<sup>6</sup> Esta función faltante significa que

---

<sup>1</sup><https://gohugo.io/overview/introduction/>

<sup>2</sup><http://jekyllrb.com>

<sup>3</sup><https://hexo.io>

<sup>4</sup><https://discuss.gohugo.io>

<sup>5</sup><https://github.com/gohugoio/hugo/issues>

<sup>6</sup>Otra motivación fue una manera más fácil de crear nuevas páginas o publicaciones. Los generadores de sitios estáticos a menudo proporcionan comandos para crear nuevas publicaciones, pero a menudo tiene que abrir y modificar el nuevo archivo creado a mano después de usar estos comandos. Estaba muy frustrado por esto, porque estaba buscando una interfaz gráfica de usuario

no puede generar resultados fácilmente usando el código de R en sus páginas web, ya que solo puede usar documentos estáticos de Markdown. Además, el motor de Markdown predeterminado de Hugo es “Blackfriday”, que es menos poderoso que Pandoc.<sup>7</sup>

Hugo usa una estructura especial de archivos y carpetas para crear su sitio web (Figura 2.1). El resto de este capítulo brindará más detalles sobre los siguientes archivos y carpetas:

- `config.toml`
- `content/`
- `static/`
- `themes/`
- `layouts/`

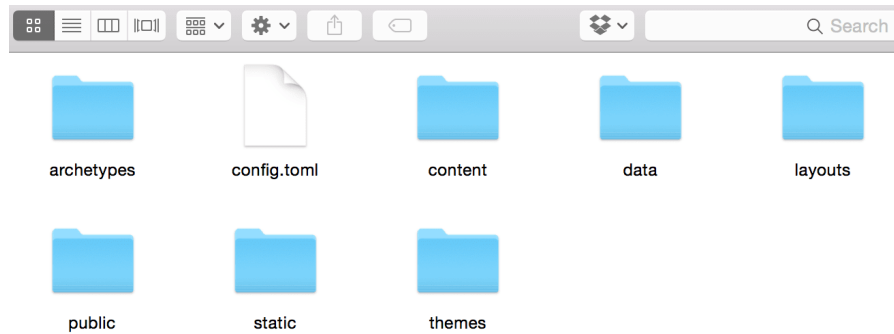
---

## 2.2 Configuración

El primer archivo que puede ver es el archivo configuration o `config` en su directorio raíz, en el que puede establecer configuraciones globales de su sitio. Puede contener opciones como el título y la descripción de su sitio, así como otras opciones globales como enlaces a sus redes sociales, el menú de navegación y la URL base de su sitio web.

donde simplemente pudiera completar el título, el autor, la fecha y otra información sobre una página, luego poder comenzar a escribir el contenido de inmediato. Es por eso que proporcioné el complemento de RStudio “New Post” y la función `blogdown::new_post()`. En los últimos años, lo odié cada vez que estaba a punto de crear una nueva publicación, ya sea a mano o a través de la línea de comandos de Jekyll. Finalmente, me volví adicto a los blogs una vez que terminé el complemento de RStudio.

<sup>7</sup>El soporte de Pandoc se ha agregado en un pull request de Hugo: <https://github.com/gohugoio/hugo/pull/4060>. Sin embargo, creo que el soporte es bastante limitado, y le recomiendo que use el formato R Markdown, porque con el soporte oficial de Pandoc en Hugo, no puede personalizar las opciones de la línea de comandos de Pandoc, la renderización no está en caché (podría ser lento), y no podrá usar ninguna extensión de Markdown del paquete **bookdown** (como la numeración de los títulos de las figuras).



**FIGURE 2.1:** Posibles archivos y carpetas creados cuando crea un nuevo sitio usando **blogdown**.

Al generar su sitio, Hugo buscará primero un archivo llamado `config.toml`. Si no puede encontrar uno, continuará buscando `config.yaml`.<sup>8</sup> Como la mayoría de los temas de Hugo contienen sitios de ejemplo que envían archivos `config.toml`, y el formato TOML (Tom’s Obvious, Minimal Language) parece ser más popular en la comunidad de Hugo, hablaremos principalmente de `config.toml` aquí.

Recomendamos que utilice la sintaxis TOML solo para el archivo de configuración (también puede usar YAML si lo prefiere), y use YAML como el formato de datos para los metadatos de las páginas y publicaciones de R Markdown, porque R Markdown y **blogdown** son totalmente compatibles solo con YAML.<sup>9</sup> Si tiene un sitio web que ya ha utilizado TOML, puede usar `blogdown::hugo_convert(unsafe = TRUE)` para convertir los datos de TOML a YAML, pero primero asegúrese de hacer una copia de seguridad del sitio web porque sobrescribirá los archivos de Markdown.

La documentación de Hugo no utiliza TOML o YAML consisten-

---

<sup>8</sup>Hugo también admite `config.json`, pero **blogdown** no lo admite, por lo que no recomendamos que lo use.

<sup>9</sup>TOML tiene sus ventajas, pero creo que no son significativas en el contexto de los sitios web de Hugo. Es un dolor tener que conocer otro idioma, TOML, cuando YAML significa “Yet Another Markup Language”. No estoy seguro de si el cómic XKCD se aplica en este caso: <https://xkcd.com/927/>.

temente en sus ejemplos, lo que puede ser confuso. Preste mucha atención al formato de configuración al copiar ejemplos en su propio sitio web.

### 2.2.1 Sintaxis TOML

Si no está familiarizado con la sintaxis de TOML, le daremos una breve descripción general y podrá leer la documentación completa<sup>10</sup> para conocer los detalles.

TOML se compone de pares clave-valor separados por signos iguales:

```
key = value
```

Cuando desee editar una configuración en el archivo TOML, simplemente cambie el valor. Los valores que son cadenas de caracteres deben estar entre comillas, mientras que los valores booleanos deben estar minúsculos y descubiertos.

Por ejemplo, si desea darle a su sitio el título “Mi Sitio Impresionante” y usar URL relativas en lugar de las URL absolutas pre-determinadas, puede tener las siguientes entradas en su archivo `config.toml`.

```
title = "Mi sitio impresionante"
```

```
relativeURLs = true
```

La mayoría de las variables globales de su sitio web se ingresan en el archivo `config.toml` exactamente de esta manera.

Más adelante en su archivo `config`, puede observar algunos valores entre paréntesis como este:

---

<sup>10</sup><https://github.com/toml-lang/toml>

```
[social]
  github = "https://github.com/rstudio/blogdown"
  twitter = "https://twitter.com/rstudio"
```

Esta es una tabla en el lenguaje TOML y Hugo los usa para completar información en otras páginas dentro de su sitio. Por ejemplo, la tabla anterior rellenará la variable `.site.social` en las plantillas de su sitio (más información sobre esto en la sección ??).

Por último, puede encontrar algunos valores en corchetes dobles como este:

```
[[menu.main]]
  name = "Blog"
  url = "/blog/"

[[menu.main]]
  name = "Categories"
  url = "/categories/"

[[menu.main]]
  name = "About"
  url = "/about/"
```

En TOML, los corchetes dobles se usan para indicar una matriz de tablas. Hugo interpreta esta información como un menú. Si el código anterior se encuentra en un archivo `config.toml`, el sitio web resultante tendrá enlaces a las páginas Blog, Categorías y Acerca de en el menú principal del sitio. La ubicación y el estilo de ese menú se especifican en otra parte, pero aquí se definen los nombres de las opciones de cada menú y los enlaces a cada sección.

El archivo `config.toml` es diferente para cada tema. Asegúrese de que cuando elija un tema, lea su documentación a fondo para comprender lo que hace cada una de las opciones de configuración (más sobre los temas en la sección 2.4).



### 2.2.2 Opciones

Todas las opciones incorporadas que puede establecer para Hugo se enumeran en <https://gohugo.io/overview/configuration/>. Puede cambiar cualquiera de estas opciones, excepto `contentDir`, que está codificado en `content` en **blogdown**. Nuestra recomendación general es que será mejor que no modifique los valores predeterminados a menos que comprenda las consecuencias. Enumeramos algunas opciones que pueden ser de su interés:

- `baseUrl`: Normalmente tiene que cambiar el valor de esta opción a la URL base de su sitio web. Algunos temas de Hugo pueden tenerlo configurado para `http://replace-this-with-your-hugo-site.com/` o `http://www.example.com/` en sus sitios de ejemplo, pero asegúrese de reemplazarlos con su propia URL (consulte el capítulo ?? y el apéndice ?? para obtener más información sobre la publicación de sitios web y la obtención de nombres de dominio). Tenga en cuenta que esta opción puede ser una URL con un subtrayecto, si su sitio web se publicará en una subruta de un nombre de dominio, e.g., `http://www.example.com/docs/`.
- `enableEmoji`: Puede configurarlo en `true` para que pueda usar Emoticones Emoji<sup>11</sup> como `:smile:` en Markdown.
- `permalinks`: Reglas para generar enlaces permanentes de sus páginas. Por defecto, Hugo usa nombres de archivos completos bajo `content/` para generar links, e.g., `content/about.md` será renderizado a `public/about/index.html`, y `content/post/2015-07-23-foo.md` será renderizado a `public/post/2015-07-23-foo/index.html`, entonces los enlaces reales son `/about/` y `/post/2015-07-23-foo/` en el sitio web. Aunque no es necesario establecer reglas personalizadas para enlaces permanentes, es común ver enlaces de la forma `/YYYY/mm/dd/post-title/`. Hugo le permite usar varias piezas de información sobre un archivo fuente para generar un enlace, como la fecha (año, mes y día), título y nombre de archivo, etc. El enlace puede ser independiente del nombre del archivo. Por

---

<sup>11</sup><http://www.emoji-cheat-sheet.com>

ejemplo, puede pedirle a Hugo que presente páginas bajo `content/post/` usando la fecha y el título de sus enlaces:

```
[permalinks]
  post = "[:year/:month/:day/:title/"]
```

Personalmente, le recomiendo que use la variable `:slug`<sup>12</sup> En lugar de `:title`:

```
[permalinks]
  post = "[:year/:month/:day/:slug/"]
```

Esto se debe a que el título de su publicación puede cambiar, y es probable que no desee que el enlace a la publicación cambie; de lo contrario, debe redirigir el enlace anterior al nuevo enlace, y habrá otros tipos de problemas, como los comentarios de Disqus. La variable `:slug` vuelve a `:title` si un campo llamado `slug` no está establecido en los metadatos YAML de la publicación. Puede establecer un slug fijo para que el enlace a la publicación siempre sea fijo y tendrá la libertad de actualizar el título de su publicación.

Puede encontrar una lista de todas las posibles variables que usted puede usar en la opción `permalinks` en <https://gohugo.io/extras/permalinks/>.

- `publishDir`: El directorio bajo el cual quiere generar el sitio web.
- `theme`: El nombre del directorio de Hugo bajo `themes/`.

---

<sup>12</sup>Una slug es simplemente una cadena de caracteres que puede usar para identificar una publicación específica. Una slug no cambiará, incluso si el título cambia. Por ejemplo, si decide cambiar el título de su publicación de “Me encanta el blogdown” a “Por qué blogdown es el mejor paquete de la historia” y usó el título de la publicación en la URL, sus enlaces anteriores ahora se romperán. Si, en cambio, especifica la URL a través de un slug (algo así como “blogdown-love”), puede cambiar el título tantas veces como quiera y no terminará con enlaces rotos.

- `ignoreFiles`: Una lista de patrones de archivo (expresiones regulares) para Hugo con el fin de que ignore ciertos archivos cuando se construye el sitio. Recomiendo que especifique al menos estos patrones `["\\.Rmd$", "\\.Rmarkdown$", "_files$", "_cache$"]`. Debería ignorar los archivos `.Rmd` porque **blogdown** los compilará a `.html`, y le basta a Hugo usar los archivos `.html`. No hay necesidad de que Hugo construya archivos `.Rmd`, y actualmente Hugo no sabe cómo. Los directorios con sufijos `_files` y `_cache` deberían ser ignorados porque contienen archivos auxiliares una vez que un archivo `Rmd` se compila, y **blogdown** los almacenará. Hugo no los debería copiar de nuevo al directorio `public/`.
- `uglyURLs`: Por defecto, Hugo genera URLs “limpias” `uglyURLs`. Esto puede ser un poco sorprendente y requiere que comprenda cómo funcionan las URL cuando su buscador obtiene una página de un servidor. Básicamente, Hugo genera `foo/index.html` para `foo.md` de forma predeterminada en lugar de `foo.html`, porque el primero le permite visitar la página a través de la URL limpia `foo/` sin `index.html`. La mayoría de los servidores web entienden solicitudes como `http://www.example.com/foo/` y presentan `index.html` bajo `foo/`. Si prefiere el mapeo estricto de `*.md` a `*.html`, puede habilitar las URL “feas” configurando `uglyURLs` en `true`.
- `hasCJKLanguage`: Si su sitio web se encuentra principalmente en CJK (chino, coreano y japonés), le recomiendo que configure esta opción en `true`, para que el resumen automático y el recuento de palabras de Hugo funcionen mejor.

Además de las opciones incorporadas de Hugo, puede establecer otras opciones arbitrarias en `config.toml`. Por ejemplo, es muy común ver una opción llamada `params`, que se usa ampliamente en muchos temas de Hugo. Cuando vea una variable `.Site.Params.FOO` en un tema de Hugo, significa una opción `foo` que se establece bajo `[params]` en `config.toml`, por ejemplo, `.Site.Params.author` es Frida Gomam con el siguiente archivo de configuración:

```
[params]
  author = "Frida Gomam"
  dateFormat = "2006/01/02"
```

El objetivo de todas estas opciones es evitar cualquier problema de codificación en los temas de Hugo, de modo que los usuarios puedan editar fácilmente un único archivo de configuración para aplicar el tema a sus sitios web, en lugar de pasar por muchos archivos HTML y realizar cambios uno por uno.

---

## 2.3 Contenido

La estructura del directorio `content/` puede ser arbitraria. Una estructura común es que hay algunas páginas estáticas bajo la raíz de `content/`, y un subdirectorio `post/` que contiene publicaciones de blog:

```
|— _index.md
|— about.md
|— vitae.md
|— post/
|   |— 2017-01-01-foo.md
|   |— 2017-01-02-bar.md
|   └ ...
└ ...
```

### 2.3.1 Metadatos YAML

Cada página debe comenzar con los metadatos YAMLYAML que especifican información como el título, la fecha, el autor, las categorías, las etiquetas, etc. Según el tema específico de Hugo y las plantillas que use, algunos de estos campos pueden ser opcionales.

Entre todos los campos de YAML, queremos llamar su atención sobre estos:

- `draft`: Puede marcar un documento como borrador Borrador configurando `draft: true` en sus metadatos YAML. Los borradores de mensajes no se mostrarán si el sitio se compila mediante `blogdown::build_site()` o `blogdown::hugo_build()`, pero se presentarán en el modo de vista previa local (consulte la sección [D.3](#))
- `publishdate`: Puede especificar una fecha futura para publicar un post. Al igual que en las publicaciones preliminares, las publicaciones futuras solo se presentan en el modo de vista previa local.
- `weight`: Este campo puede tomar un valor numérico para indicarle a Hugo el orden de las páginas al ordenarlas ; por ejemplo, cuando genera una lista de todas las páginas debajo de un directorio y dos publicaciones tienen la misma fecha, puede asignar diferentes ponderaciones para obtener el orden deseado en la lista.
- `slug`: Una cadena de caracteres como la cola de la URL. Es particularmente útil cuando define reglas personalizadas para URL permanentes (vea la sección [2.2.2](#)).

### 2.3.2 Cuerpo

Como mencionamos en la sección `@ref(formato de salida)`, su publicación puede escribirse en R o Markdown. Tenga cuidado con las diferencias de sintaxis entre los dos formatos cuando escribe el cuerpo de una publicación.

### 2.3.3 Código corto

Además de todas las características de Markdown, Hugo proporciona una característica útil llamada “códigos abreviados”. Puede usar un shortcode en el cuerpo de su publicación. Cuando Hugo

presenta la publicación, puede generar automáticamente un fragmento de HTML basado en los parámetros que pasa al código corto. Esto es conveniente porque no tiene que escribir o insertar una gran cantidad de código HTML en su publicación. Por ejemplo, Hugo tiene un código abreviado incorporado para incrustar tarjetas de Twitter. Normalmente, así es como inserta una tarjeta de Twitter (Figura @ref(fig: jtleek-tweet)) en una página:

```
<blockquote class="twitter-tweet">
  <p lang="en" dir="ltr">Anyone know of an R package for
    interfacing with Alexa Skills?
    <a href="https://twitter.com/thosjleeper">@thosjleeper</a>
    <a href="https://twitter.com/xieyihui">@xieyihui</a>
    <a href="https://twitter.com/drob">@drob</a>
    <a href="https://twitter.com/JennyBryan">@JennyBryan</a>
    <a href="https://twitter.com/HoloMarkeD">@HoloMarkeD</a> ?
  </p>
  &mdash; Jeff Leek (@jtleek)
  <a href="https://twitter.com/jtleek/status/852205086956818432">
    April 12, 2017
  </a>
</blockquote>
<script async src="//platform.twitter.com/widgets.js" charset="utf-8">
</script>
```



**FIGURE 2.2:** A tweet by Jeff Leek.

Si usa el código abreviado, todo lo que necesita en el documento fuente de reducción es:

```
{{< tweet 852205086956818432 >}}
```

Básicamente, solo necesita pasar el ID del tweet a un código corto llamado `tweet`. Hugo buscará el tweet automáticamente y renderizará el fragmento de HTML por usted. Para obtener más información sobre los códigos abreviados, consulte <https://gohugo.io/extras/shortcodes/>.

Se supone que los códigos cortos funcionan solo en documentos de Markdown. Para usar códigos abreviados en R Markdown en lugar de Markdown simple, debe llamar a la función `blogdown::shortcode()`, e.g.,

```
```{r echo=FALSE}  
blogdown::shortcode('tweet', '852205086956818432')  
```
```

---

## 2.4 Temas

Un tema de Hugo es una colección de plantillas y archivos opcionales del sitio web, como archivos CSS y JavaScript. En pocas palabras, un tema define el aspecto de su sitio web después de que su contenido fuente se presente a través de las plantillas.

Hugo ha proporcionado una gran cantidad de temas aportados por los usuarios en <https://themes.gohugo.io>. A menos que sea un diseñador web experimentado, es mejor que comience desde un tema existente aquí. La calidad y la complejidad de estos temas varían mucho, y debe elegir uno con precaución. Por ejemplo, puede ver el número de estrellas de un repositorio de temas en GitHub, así como si el repositorio todavía está relativamente activo. No recomendamos utilizar un tema que no se haya actualizado durante más de un año.

En esta sección, explicaremos cómo funciona el tema predeterminado en **blogdown**, que también puede brindarle algunas ideas sobre cómo comenzar con otros temas.

### 2.4.1 El tema por defecto

El tema predeterminado en **blogdown**, `hugo-lithium-theme`, está alojado en GitHub en <https://github.com/yihui/hugo-lithium-theme>. Fue escrito originalmente por Jonathan Rutheiser, y he realizado varios cambios en él. Este tema es adecuado para quienes prefieren estilos mínimos y desean crear un sitio web con algunas páginas y algunas publicaciones en el blog.

Normalmente, un repositorio de temas en GitHub tiene un archivo `README`, que también sirve como la documentación del tema. Después de leerlo, el siguiente archivo para buscar es `config.toml` en el directorio `exampleSite`, que contiene configuraciones de muestra para un sitio web basado en este tema. Si un tema no tiene un archivo `README` o `exampleSite`, probablemente no debería usarlo.

El `config.toml` del tema `hugo-lithium-theme` contiene las siguientes opciones:

```
baseurl = "/"
relativeurls = false
languageCode = "en-us"
title = "A Hugo website"
theme = "hugo-lithium-theme"
googleAnalytics = ""
disqusShortname = ""
ignoreFiles = ["\\.Rmd$", "\\..Rmarkdown", "_files$", "_cache$"]

[permalinks]
  post = "/:year/:month/:day/:slug/"

[[menu.main]]
  name = "About"
```



```
url = "/about/"
[[menu.main]]
  name = "GitHub"
  url = "https://github.com/rstudio/blogdown"
[[menu.main]]
  name = "Twitter"
  url = "https://twitter.com/rstudio"

[params]
  description = "A website built through Hugo and blogdown."

  highlightjsVersion = "9.11.0"
  highlightjsCDN = "//cdn.bootcss.com"
  highlightjsLang = ["r", "yaml"]
  highlightjsTheme = "github"

  MathJaxCDN = "//cdn.bootcss.com"
  MathJaxVersion = "2.7.1"

[params.logo]
  url = "logo.png"
  width = 50
  height = 50
  alt = "Logo"
```

Algunas de estas opciones pueden ser obvias para comprender, y algunas pueden necesitar explicaciones:

- `baseUrl`: Puede configurar esta opción después, después de tener un nombre de dominio para su sitio web. No olvide la barra inclinada.
- `relativeurls`: Esto es opcional. Puede configurarlo como `true` solo si tiene la intención de ver su sitio web localmente a través de su visor de archivos, por ejemplo, hacer doble clic en un archivo HTML y verlo en su navegador. Esta opción tiene como

valor predeterminado `false` en Hugo, y significa que su sitio web debe ser visto a través de un servidor web, por ejemplo, `blogdown::serve_site()` ha proporcionado un servidor web local, por lo que puede obtener una vista previa localmente cuando `relativeurls = false`.

- `title`: El título de su sitio web. Típicamente esto se muestra en la barra de título del buscador web o sobre una pestaña de página.
- `theme`: El nombre del directorio del tema. Debe tener mucho cuidado al cambiar los temas, porque un tema puede ser drásticamente diferente de otro tema en términos de configuraciones. Es muy posible que un tema diferente no funcione con su `config.toml` actual. De nuevo, debe leer la documentación de un tema para saber qué opciones son compatibles o requeridas.
- `googleAnalytics`: El ID de seguimiento de Google Analytics (por ejemplo, `UA-000000-2`). Puede inscribirse en <https://analytics.google.com> para obtener un ID de seguimiento.
- `disqusShortname`: El ID de Disqus que creó durante el proceso de configuración de la cuenta en <https://disqus.com>. Esto es necesario para habilitar los comentarios en su sitio.<sup>13</sup> Tenga en cuenta que debe configurar un `baseurl` funcional y publicar su sitio web antes de que los comentarios de Disqus pueda funcionar.
- `ignoreFiles` y `permalinks`: Estas opciones han sido explicadas en la sección 2.2.2.
- `menu`: Esta lista de opciones especifica el texto y la URL de los elementos del menú en la parte superior. Ver la figura 1.1 para una página de muestra. Puede cambiar o agregar más elementos de menú. Si desea ordenar los artículos, puede asignar un peso a cada artículo, e.g.,

---

<sup>13</sup>Como mencionamos en la sección @ref(sitios estáticos), **blogdown** genera contenido estático e inmutable. Para agregar algo dinámico y siempre cambiante (como la posibilidad de que sus seguidores dejen comentarios), debe incorporar un sistema de comentarios externo como Disqus.

```
[[menu.main]]
  name = "Home"
  url = "/"
  weight = 1
[[menu.main]]
  name = "About"
  url = "/about/"
  weight = 2
[[menu.main]]
  name = "GitHub"
  url = "https://github.com/rstudio/blogdown"
  weight = 3
[[menu.main]]
  name = "CV"
  url = "/vitae/"
  weight = 4
[[menu.main]]
  name = "Twitter"
  url = "https://twitter.com/rstudio"
  weight = 5
```

En el ejemplo anterior, agregué un elemento de menú cv con la URL `/vitae/`, y se supone que hay un archivo fuente correspondiente `vitae.md` debajo del directorio `content/` para generar la página `/vitae/index.html`, por lo que el enlace realmente funcionará.

- `params`: Diversos parámetros del tema.
  - `description`: Una breve descripción de su sitio web. No es visible en las páginas web (solo puede verlo desde la fuente HTML), pero debe dar a los motores de búsqueda una pista sobre su sitio web.
  - `highlightjs*`: Estas opciones se usan para configurar las librerías de JavaScript `highlight.js`<sup>14</sup> para resaltar la sintaxis de los bloques de código sobre las páginas web.

---

<sup>14</sup><https://highlightjs.org>

Puede cambiar la versión (e.g., 9.12.0), el hosto CDN (e.g., usando cdnjs<sup>15</sup>: `//cdnjs.cloudflare.com/ajax/libs`), agregar más lenguajes (e.g., `["r", "yaml", "tex"]`), y cambiar el tema (e.g., `atom-one-light`). Vea <https://highlightjs.org/static/demo/> para todos los lenguajes y temas que highlight.js soporta.

- `MathJax*`: La librería de JavaScript MathJax puede renderizar expresiones matemáticas en LaTeX sobre páginas web. De la misma forma que `highlightjsCDN`, puede especificar el host CDN de MathJax, e.g., `//cdnjs.cloudflare.com/ajax/libs`, y puede especificar la versión de MathJax.
- `logo`: Una lista de opciones para definir el logo del sitio web. Por defecto, la imagen `logo.png` bajo el directorio `static/` se usa.

Si quiere ser un desarrollador de temas y comprender completamente todos los detalles técnicos sobre estas opciones, debe comprender las plantillas de Hugo, que presentaremos en la sección 2.5.

---

## 2.5 Plantillas

Un tema de Hugo consta de dos componentes principales: plantillas y archivos web. El primero es esencial y le dice a Hugo cómo presentar una página.<sup>16</sup> El último es opcional pero también importante. Por lo general, consta de archivos CSS y JavaScript, así como otros recursos, como imágenes y videos. Estos activos determinan la apariencia y la funcionalidad de su sitio web, y algunos pueden estar integrados en el contenido de sus páginas web.

---

<sup>15</sup><https://cdnjs.com>

<sup>16</sup>La funcionalidad más común de las plantillas es hacer páginas HTML, pero también puede haber plantillas especiales, por ejemplo, para fuentes RSS y sitemaps, que son archivos XML.

Puede obtener más información sobre las plantillas de Hugo en la documentación oficial (<https://gohugo.io/templates/overview/>). Hay muchos tipos diferentes de plantillas. Para que le resulte más fácil dominar las ideas clave, creé un tema de Hugo muy mínimo, que cubre la mayoría de las funcionalidades que un usuario promedio puede necesitar, pero el número total de líneas es de solo 150, por lo que podemos hablar de todas las fuentes código de este tema en la siguiente subsección.

### 2.5.1 Un pequeño ejemplo

XMin<sup>17</sup> es un tema de Hugo. Lo escribí desde cero en aproximadamente 12 horas. Aproximadamente media hora se gastó en plantillas, se dedicaron 3,5 horas a modificar los estilos CSS y se gastaron 8 horas en la documentación (<https://xmin.yihui.name>). Creo que este puede ser un caso representativo de cuánto tiempo pasaría en cada parte cuando diseñe un tema. Está, quizás, en nuestra naturaleza pasar mucho más tiempo en cosas cosméticas como CSS que en cosas esenciales como plantillas. Mientras que la codificación es, a menudo, más fácil que la documentación.

Mostraremos el código fuente del tema XMin. Debido a que el tema puede actualizarse ocasionalmente en el futuro, puede seguir este enlace para obtener una versión fija de la que hablaremos en esta sección: <https://github.com/yihui/hugo-xmin/tree/4bb305>. A continuación se muestra una vista en árbol de todos los archivos y directorios del tema:

```
hugo-xmin/  
├─ LICENSE.md  
├─ README.md  
├─ archetypes  
│   └─ default.md  
├─ layouts  
└─ 404.html
```

<sup>17</sup><https://github.com/yihui/hugo-xmin>

```

|   |   | _default
|   |   | | list.html
|   |   | | single.html
|   |   | | terms.html
|   |   | | partials
|   |   | |   | foot_custom.html
|   |   | |   | footer.html
|   |   | |   | head_custom.html
|   |   | |   | header.html
|   |   | static
|   |   | | css
|   |   | |   | fonts.css
|   |   | |   | style.css
|   |   | exampleSite
|   |   | | config.toml
|   |   | | content
|   |   | | | _index.md
|   |   | | | about.md
|   |   | | | note
|   |   | | |   | 2017-06-13-a-quick-note.md
|   |   | | |   | 2017-06-14-another-note.md
|   |   | | | post
|   |   | | |   | 2015-07-23-lorem-ipsum.md
|   |   | | |   | 2016-02-14-hello-markdown.md
|   |   | | layouts
|   |   | | | partials
|   |   | | |   | foot_custom.html
|   |   | | public
|   |   | | ...

```

LICENSE.md y README.md no son componentes necesarios de un tema, pero definitivamente debe elegir una licencia para su código fuente para que otras personas puedan usar su código correctamente, y un README puede ser la breve documentación de su software.

El archivo `archetypes/default.md` define la plantilla predeterminada

en función de qué usuarios pueden crear nuevas publicaciones. En este tema, `default.md` solo proporcionaba metadatos YAML vacíos:

```
---  
---
```

Los directorios más importantes de un tema son `layouts/` y `static/`. Las plantillas HTML se almacenan en `layouts/`, y los archivos se almacenan en `static/`.

Para comprender `layouts/`, debe conocer algunos conceptos básicos sobre HTML (consulte la sección [B.1](#)) porque las plantillas en este directorio son, en su mayoría, documentos o fragmentos HTML. Hay muchos tipos posibles de subdirectorios en `layouts/`, pero solo vamos a introducir dos aquí: `_default/` y `partials/`.

- El directorio `_default/` es donde almacena las plantillas predeterminadas para sus páginas web. En el tema XMin, tenemos tres plantillas: `single.html`, `list.html`, y `terms.html`.
  - `single.html` es una plantilla para presentar páginas individuales. Una sola página básicamente corresponde a un documento de Markdown bajo `content/`, y contiene tanto los metadatos (YAML) como el contenido. Por lo general, queremos mostrar el título de la página, el autor, la fecha y el contenido. A continuación se muestra el código fuente de `single.html` de XMin:

```
{{ partial "header.html" . }}  
<div class="article-meta">  
<h1><span class="title">{{ .Title }}</span></h1>  
{{ with .Params.author }}  
<h2 class="author">{{ . }}</h2>  
{{ end }}  
{{ if .Params.date }}  
<h2 class="date">{{ .Date.Format "2006/01/02" }}</h2>
```

```
{{ end }}  
</div>  
  
<main>  
{{ .Content }}  
</main>  
  
{{ partial "footer.html" . }}
```

Verá muchos pares de corchetes `{{}}`, y así es como se programan las plantillas usando las variables y funciones de Hugo.

La plantilla comienza con una plantilla parcial `header.html`, para la cual verá el código fuente pronto. Por ahora, puede imaginarlo como todas las etiquetas HTML antes del cuerpo de su página (e.g., `<html><head>`). Las plantillas parciales se usan, principalmente, para reutilizar código HTML. Por ejemplo, todas las páginas HTML pueden compartir tags muy similares `<head></head>`, y puede factorizar las partes comunes en plantillas parciales.

Los metadatos de una página se incluyen en un elemento `<div>` con la clase `article-meta`. Recomendamos que asigne clases a elementos HTML al diseñar plantillas, de modo que sea más fácil aplicar estilos CSS a estos elementos usando nombres de clase. En una plantilla, tiene acceso a muchas variables proporcionadas por Hugo, por ejemplo, la variable `.Title` almacena el valor del título de la página, y escribimos el título en `<span>` en un encabezado de primer nivel `<h1>`. De forma similar, el autor y la fecha se escriben en `<h2>`, pero solo si se proporcionan en los metadatos YAML. La sintaxis `{{ con FOO }}{{ . }}{{ end }}` es una abreviatura de `{{if FOO }}{{ FOO }}{{ end }}`, es decir, le ahorra el esfuerzo de digitar la expresión `FOO` dos veces usando `{{ . }}`. El método `.Format` se puede aplicar a un objeto de fecha, y en este tema, formateamos las fechas en el formato `YYYY/mm/dd` (`2006/01/02` es la forma de especificar el formato en Go) .



Luego mostramos el contenido de una página, que se almacena en la variable `.content`. El contenido está envuelto en una etiqueta HTML semántica `<main>`.

La plantilla finaliza después de incluir otra plantilla parcial `footer.html` (código fuente que se mostrará en breve).

Para que sea más fácil de entender cómo funciona una plantilla, mostramos un mínimo ejemplo de publicación a continuación:

```
---
title: Hello World
author: Frida Gomam
date: 2017-06-19
---

A single paragraph.
```

Con la plantilla `single.html`, se convertirá en una página HTML con un código fuente que se parece más o menos a esto (con el encabezado y el pie de página omitidos):

```
<div class="article-meta">
  <h1><span class="title">Hello World</span></h1>
  <h2 class="author">Frida Gomam</h2>
  <h2 class="date">2017/06/19</h2>
</div>

<main>
  <p>A single paragraph.</p>
</main>
```

Para un ejemplo completo de una página sencilla, puede ver <https://xmin.yihui.name/about/>.

- `list.html` es la plantilla para generar listas de páginas, como una lista de publicaciones de blog, o una lista de páginas

dentro de una categoría o etiqueta. Aquí está su código fuente:

```
{{ partial "header.html" . }}

{{if not .IsHome }}
<h1>{{ .Title }}</h1>
{{ end }}

{{ .Content }}

<ul>
  {{ range (where .Data.Pages "Section" "!=" "") }}
    <li>
      <span class="date">{{ .Date.Format "2006/01/02" }}</span>
      <a href="{{ .URL }}">{{ .Title }}</a>
    </li>
  {{ end }}
</ul>

{{ partial "footer.html" . }}
```

Nuevamente, usa dos plantillas parciales `header.html` y `footer.html`. La expresión `{{if not .IsHome}}` significa, si esta lista no es la página de inicio, muestre el título de la página. Esto es porque no quiero mostrar el título en la página de inicio. Es solo mi preferencia personal. Sin duda, puede mostrar el título en `<h1>` en la página de inicio, si lo desea. El `{{.Content}}` muestra el contenido de la lista. Tenga en cuenta que típicamente `.Content` está vacío, lo que puede sorprender. Esto se debe a que una página de lista no se genera a partir de un archivo de marca de origen de forma predeterminada. Como sea, hay una excepción. Cuando se escribe un archivo Markdown especial `_index.md` en un directorio correspondiente al nombre de la lista, el `.Content` de la lista será el contenido de este archivo Markdown. Por ejemplo, puede definir el contenido de su página de inicio

en `content/_index.md`, y el contenido de la página de la lista de publicaciones en `content/post/_index.md`.

A continuación, generamos la lista utilizando un bucle (`range`) a través de todas las páginas filtradas por la condición de que la sección de una página no debe estar vacía. “Section” en Hugo significa el nombre del subdirectorío de primer nivel bajo `content/`. Por ejemplo, la sección de `content/post/foo.md` es `post`. Por lo tanto, el filtro significa que enumeraremos todas las páginas bajo subdirectoríos de `content/`. Esto excluirá las páginas debajo del directorío raíz `content/`, como `content/about.md`.

Tenga en cuenta que la variable `.data` es dinámica y su valor cambia de acuerdo con la lista específica que desea generar. Por ejemplo, la página de la lista <https://xmin.yihui.name/post/> solo contiene páginas bajo `content/post/`, y <https://xmin.yihui.name/note/> solo contiene páginas bajo `content/note/`. Estas páginas de lista son generadas automáticamente por Hugo, y no necesita pasar explícitamente por las secciones publicación y nota. Es decir, una sola plantilla `list.html` generará múltiples listas de páginas según las secciones y los términos de taxonomía (por ejemplo, `categories` y `tags`) que tenga en su sitio web.

Los elementos de la lista están representados por las etiquetas HTML `<li>` en `<ul>`. Cada elemento consta de la fecha, el enlace y el título de una página. Puede ver <https://xmin.yihui.name/post/> para obtener un ejemplo completo de una página de lista.

- `terms.html` es la plantilla para la página de inicio de los términos de la taxonomía. Por ejemplo, puede usarlo para generar la lista completa de categorías o etiquetas. El código fuente está a continuación:

```
{{ partial "header.html" . }}  
  
<h1>{{ .Title }}</h1>
```

```

<ul class="terms">
  {{ range $key, $value := .Data.Terms }}
    <li>
      <a href='{{ (print "/" $.Data.Plural "/" $key) | relURL }}'>
        {{ $key }}
      </a>
      ({{ len $value }})
    </li>
  {{ end }}
</ul>

{{ partial "footer.html" . }}

```

Similar a `list.html`, también usa un bucle. La variable `.Data.Terms` almacena todos los términos bajo una taxonomía, por ejemplo, todos los nombres de categorías. Puede considerarlo como una lista con nombre en R (llamado ‘map’ en Go), cuyos nombres son los términos y los valores son listas de páginas. La variable `$key` denota el término y `$value` denota la lista de páginas asociadas con este término. Lo que presentamos en cada `<li>` es un enlace al término página, así como el recuento de publicaciones que utilizan este término (`len` es una función Go que devuelve la longitud de un objeto).

Hugo representa automáticamente todas las páginas de taxonomía, y los nombres de ruta son las formas plurales de las taxonomías, por ejemplo, <https://xmin.yihui.name/categories/> y <https://xmin.yihui.name/tags/>. Ese es el significado de `.Data.Plural`. El `$` inicial es obligatorio porque estamos dentro de un bucle y necesitamos acceder a variables del alcance externo. El enlace del término se pasa a la función Hugo `relURL` a través de un conector `|` para hacerlo relativo, lo cual es una buena práctica porque los enlaces relativos son más portátiles (independientemente del nombre de dominio).

- El directorio `partials/` es el lugar para poner los fragmentos

HTML para ser reutilizados por otras plantillas a través de la función `partial`. Tenemos cuatro plantillas parciales bajo este directorio:

- `header.html` define la etiqueta `<head>` y el menú de navegación en la etiqueta `<nav>`.

```
<!DOCTYPE html>
<html lang="{{ .Site.LanguageCode }}">
  <head>
    <meta charset="utf-8">
    <title>{{ .Title }} | {{ .Site.Title }}</title>
    <link href='{{ "/css/style.css" | relURL }}'
          rel="stylesheet" />
    <link href='{{ "/css/fonts.css" | relURL }}'
          rel="stylesheet" />
    {{ partial "head_custom.html" . }}
  </head>

  <body>
    <nav>
      <ul class="menu">
        {{ range .Site.Menus.main }}
          <li><a href="{{ .URL | relURL }}">{{ .Name }}</a></li>
        {{ end }}
      </ul>
      <hr/>
    </nav>
```

El área `<head>` debe ser fácil de entender si está familiarizado con HTML. Tenga en cuenta que también incluimos una plantilla parcial `head_custom.html`, que está vacía en este tema, pero hará que sea mucho más fácil para los usuarios agregar código personalizado a `<head>` sin reescribir toda la plantilla. Ver la sección [@ref\(layouts personalizados\)](#) para más detalles.

El menú de navegación es esencialmente una lista, y cada

elemento de la lista se lee de la variable `.Site.Menus.main`. Esto significa que los usuarios pueden definir el menú en `config.toml`, e.g.,

```
[[menu.main]]
  name = "Home"
  url = "/"
[[menu.main]]
  name = "About"
  url = "/about/"
```

Esto generará un menú como:

```
<ul class="menu">
  <li><a href="/">Home</a></li>
  <li><a href="/about/">About</a></li>
</ul>
```

Hugo tiene un poderoso sistema de menú, y solo usamos el tipo más simple de menú en este tema. Si está interesado en más funciones como menús anidados, consulte la documentación completa en <http://gohugo.io/extras/menus/>.

\* `footer.html` define el área footer de una página y cierra el documento HTML:

```
<footer>
{{ partial "foot_custom.html" . }}
{{ with .Site.Params.footer }}
<hr/>
{{ . | markdownify }}
{{ end }}
</footer>
</body>
</html>
```

El propósito de la plantilla parcial `foot_custom.html` es el mismo que `head_custom.html`; es decir, para permitir que el usuario agregue código personalizado al `<footer>` sin volver a escribir la plantilla completa.

Por último, usamos la variable `.Site.Params.footer` para generar un pie de página. Tenga en cuenta que utilizamos la función `with` nuevamente. Recuerde que la sintaxis `{{with .Site.Params.footer}}{{ . }}{{ end }}` es una abreviatura de `{{if .Site.Params.footer }}{{.Site.Params.footer }}{{ end }}`. Esta sintaxis le evita escribir dos veces la expresión `.Site.Params.footer` usando `{{ . }}` como un marcador de posición para la variable `footer`, que se define como un parámetro de sitio en nuestro archivo `config.toml`. La función adicional `markdownify` puede convertir Markdown a HTML (es decir, `{{ . | markdownify }}`). En conjunto, esta secuencia significa que podemos definir una opción `footer` usando Markdown bajo `params` en `config.toml`, e.g.,

```
[params]
  footer = "&copy; [Yihui Xie](https://yihui.name) 2017"
```

Hay una plantilla especial `404.html`, que Hugo usa para crear la página 404 (cuando no se encuentra una página, se muestra esta página):

```
{{ partial "header.html" . }}

404 NOT FOUND

{{ partial "footer.html" . }}
```

Con todas las plantillas anteriores, podremos generar un sitio web a partir de los archivos fuente de Markdown. Sin embargo, es poco probable que esté satisfecho con el sitio web porque los elementos HTML no tienen ningún estilo y la apariencia predeterminada

puede no parecer atractiva para la mayoría de las personas. Puede haber notado que en `header.html`, hemos incluido dos archivos CSS, `/css/style.css` y `/css/fonts.css`.

Puede encontrar muchos frameworks CSS de código abierto existentes que se pueden aplicar a un tema de Hugo. Por ejemplo, el framework CSS más popular puede ser Bootstrap: <http://getbootstrap.com>. Cuando estaba diseñando XMin, me preguntaba hasta dónde podría llegar sin usar ninguno de estos frameworks existentes, porque generalmente son muy grandes. Por ejemplo, `bootstrap.css` tiene casi 10000 líneas de código cuando no se minimiza. Resultó que pude obtener una apariencia satisfactoria con aproximadamente 50 líneas de CSS, que explicaré en detalle a continuación:

- `style.css` define todos los estilos excepto las fuentes tipográficas:

```
body {  
    max-width: 800px;  
    margin: auto;  
    padding: 1em;  
    line-height: 1.5em;  
}
```

El ancho máximo del cuerpo de la página se establece en 800 píxeles porque una página excesivamente ancha es difícil de leer (800 es un umbral arbitrario que elegí). El cuerpo se centra utilizando el truco de CSS `margin: auto`, lo que significa que los márgenes superior, derecho, inferior y izquierdo son automáticos. Cuando los márgenes izquierdo y derecho de un elemento de bloque son `auto`, estará centrado.

```
/* header and footer areas */  
.menu li { display: inline-block; }  
.article-meta, .menu a {
```



```
    text-decoration: none;
    background: #eee;
    padding: 5px;
    border-radius: 5px;
}
.menu, .article-meta, footer { text-align: center; }
.title { font-size: 1.1em; }
footer a { text-decoration: none; }
hr {
    border-style: dashed;
    color: #ddd;
}
```

Recuerde que nuestro elemento de menú es una lista `<ul class="menu">` definida en `header.html`. Cambié el estilo de visualización predeterminado de `<li>` dentro del menú a `inline-block`, de modo que se distribuyan de izquierda a derecha como elementos en línea, en lugar de apilarse verticalmente como una lista de viñetas (el comportamiento predeterminado))

Para los enlaces (‘’) en el menú y el área de metadatos de un artículo, se elimina la decoración de texto predeterminada (subrayados) y se aplica un color de fondo claro. El radio del borde se establece en 5 píxeles para que pueda ver un rectángulo sutil de esquina redondeada detrás de cada enlace.

La regla horizontal (`<hr>`) se establece en una línea discontinua de color gris claro para que sea menos prominente en una página. Estas reglas se utilizan para separar el cuerpo del artículo de las áreas de encabezado y pie de página.

```
/* code */
pre {
    border: 1px solid #ddd;
    box-shadow: 5px 5px 5px #eee;
    padding: 1em;
}
```

```

    overflow-x: auto;
}
code { background: #f9f9f9; }
pre code { background: none; }

```

Para bloques de código (<pre>), aplico bordes gris claro con efectos de sombra paralela. Cada elemento de código en línea tiene un fondo gris muy claro. Estas decoraciones son simplemente por mi propio interés y énfasis peculiares en el código.

```

/* misc elements */
img, iframe, video { max-width: 100%; }
main { hyphens: auto; }
blockquote {
    background: #f9f9f9;
    border-left: 5px solid #ccc;
    padding: 3px 1em 3px;
}

table {
    margin: auto;
    border-top: 1px solid #666;
    border-bottom: 1px solid #666;
}
table thead th { border-bottom: 1px solid #ddd; }
th, td { padding: 5px; }
tr:nth-child(even) { background: #eee }

```

Los elementos incrustados, como las imágenes y los videos que exceden el margen de la página, a menudo son desagradables, por lo que restrinjo su ancho máximo al 100%. La separación silábica está activada para palabras en <main>. Las citas en bloque tienen una barra lateral izquierda gris y un fondo gris claro. Las tablas están centradas de manera predeterminada, con solo tres reglas horizontales: los bordes superior e inferior de la tabla y el

borde inferior de la tabla. Las filas de la tabla están rayadas para facilitar la lectura de la tabla, especialmente cuando la tabla es ancha.

- `fonts.css` es una hoja de estilo separada porque juega un papel crítico en la apariencia de un sitio web, y es muy probable que desee personalizar este archivo. En la mayoría de los casos, sus lectores dedicarán la mayor parte del tiempo a leer el texto en sus páginas, por lo que es importante hacer que el texto sea cómodo de leer. No soy un experto en diseño web, y acabo de elegir Palatino para el cuerpo y Lucida Console o Monaco (cualquiera que esté disponible en su sistema) para el código. Es común usar las fuentes web de Google hoy en día. Puede probar algunas fuentes web y ver si le gusta alguno de ellos.

```
body {  
    font-family: "Palatino Linotype", "Book Antiqua", Palatino, serif;  
}  
code {  
    font-family: "Lucida Console", Monaco, monospace;  
    font-size: 85%;  
}
```

Los dos archivos CSS se colocan bajo el directorio `static/css/` del tema. En la plantilla HTML `header.html`, la ruta `/css/style.css` se refiere al archivo `static/css/style.css`.

Por último, este tema proporcionó un sitio de ejemplo en `exampleSite/`. La estructura del directorio puede ser un poco confusa porque este es un tema en lugar de un sitio web. En la práctica, todo lo que se encuentra debajo de `exampleSite/` debe estar debajo del directorio raíz de un sitio web, y el directorio `hugo-xmin/` de nivel superior debe estar bajo el directorio `themes/` de este sitio web, i.e.,



- **Activar comentarios Disqus.** Similar a Google Analytics, puedes agregar la plantilla incorporada

```
{{ template "_internal/disqus.html" . }}
```

a `foot_custom.html`, y configurar el nombre corto Disqus en `config.toml`. Vea <https://github.com/yihui/hugo-xmin/pull/4> para detalles, y una vista previa en <https://deploy-preview-4--hugo-xmin.netlify.com>.

- **Configurar sintaxis resaltada mediante highlight.js.** Agregue esto a `head_custom.html`

```
<link href="//YOUR-CDN-LINK/styles/github.min.css" rel="stylesheet">
```

and this to `foot_custom.html`:

```
<script src="//YOUR-CDN-LINK/highlight.min.js"></script>
<script src="//YOUR-CDN-LINK/languages/r.min.js"></script>

<script>
hljs.configure({languages: []});
hljs.initHighlightingOnLoad();
</script>
```

Recuerde reemplazar `YOUR-CDN-LINK` con el enlace al host de CDN preferido de highlight.js, por ejemplo, `cdn.bootcss.com/highlight.js/9.12.0`. Para obtener más información sobre highlight.js, consulte su página principal: <https://highlightjs.org>. Si necesita usar otros hosts CDN, `cdnjs.com` es una buena opción: <https://cdnjs.com/libraries/highlight.js>. También puede ver qué idiomas y temas CSS son compatibles allí.

Puede ver <https://github.com/yihui/hugo-xmin/pull/5> para una implementación real, y una página de muestra con resaltado de sintaxis en <https://deploy-preview-5--hugo-xmin.netlify.com/post/2016/02/14/a-plain-markdown-post/>.

- **Soporte para expresiones matemáticas a través de MathJax.** Agregue el código de abajo a `foot_custom.html`.

```
<script src="//yihui.name/js/math-code.js"></script>
<script async
src="//cdn.bootcss.com/mathjax/2.7.1/MathJax.js?config=TeX-MML-AM_CHTML">
</script>
```

Esto requiere un conocimiento sustancial de JavaScript y la familiaridad con MathJax para comprender completamente el código anterior, y dejaremos la explicación del código en la sección B.3.

Tenga en cuenta que bootcss.com es solo un posible servidor CDN de MathJax, y usted es libre de usar otros hosts.

- **Mostrar la tabla de contenidos (TOC).** Para mostrar una TOC para las publicaciones R Markdown, solo necesita agregar el formato de salida `blogdown::html_page` con la opción `toc: true` al YAML:

```
output:
  blogdown::html_page:
    toc: true
```

Para las publicaciones simples de Markdown, debe modificar la plantilla `single.html`. La TOC de una publicación se almacena en la variable de plantilla Hugo `.TableOfContents`. Es posible que desee una opción para controlar si mostrar la TOC, por ejemplo, puede agregar una opción `toc: true` a los metadatos YAML de una publicación de marcado para mostrar la TOC. El código a

continuación se puede agregar antes del contenido de una publicación en `single.html`:

```
{{ if .Params.toc }}
{{ .TableOfContents }}
{{ end }}
```

Vea <https://github.com/yihui/hugo-xmin/pull/7> para una implementación con ejemplos.

- **Mostrar categorías y etiquetas en una publicación si se proporciona en su YAML.** Agregue el código de abajo donde usted quiera ubicar las categorías y etiquetas en `single.html`, e.g., en `<div class="article-meta"></div>`.

```
<p class="terms">
  {{ range $i := (slice "categories" "tags") }}
  {{ with ($.Param $i) }}
  {{ $i | title }}:
  {{ range $k := . }}
  <a href="{{ relURL (print "/" $i "/" $k | urlize) }}">{{ $k }}</a>
  {{ end }}
  {{ end }}
  {{ end }}
</p>
```

Básicamente, el código recorre los campos de metadatos de YAML `categories` y `tags`, y para cada campo, su valor se obtiene de `.Param`, luego usamos un bucle interno para escribir los términos con enlaces de la forma `<a href="/tags/foo/">foo</a>`.

Puede ver <https://github.com/yihui/hugo-xmin/pull/2> para la implementación completa y una previsualización en <https://deploy-preview-2--hugo-xmin.netlify.com/post/2016/02/14/a-plain-markdown-post/>.

- **Agregar paginación.** Cuando tiene una gran cantidad de publicaciones en un sitio web, es posible que no desee mostrar la lista completa en una sola página, pero muestre N publicaciones (por ejemplo,  $N = 10$ ) por página. Es fácil agregar paginación a un sitio web usando las funciones y plantillas integradas de Hugo. En lugar de recorrer todas las publicaciones en una plantilla de lista (por ejemplo, `range .Data.Pages`), pagine la lista completa de publicaciones usando la función `.Paginate` (por ejemplo, `range (.Paginate .Data.Pages)`). A continuación se muestra un fragmento de plantilla que puede insertar en su archivo de plantilla `list.html`:

```
<ul>
  {{ $paginator := .Paginate .Data.Pages }}
  {{ range $paginator.Pages }}
    <li>
      <span class="date">{{ .Date.Format "2006/01/02" }}</span>
      <a href="{{ .URL }}">{{ .Title }}</a>
    </li>
  {{ end }}
</ul>
{{ template "_internal/pagination.html" . }}
```

See <https://github.com/yihui/hugo-xmin/pull/16> for a full implementation.

- **Agregar un botón de edición de GitHub o un link a una página.** Si ninguna de las características anteriores le parece emocionante (lo que no me sorprendería), esta pequeña característica es realmente un gran ejemplo de mostrarle el poder de los archivos de texto sin formato y los sitios web estáticos, cuando se combina con GitHub (u otros servicios que admiten la edición en línea de archivos de texto sin formato). Creo que sería difícil, si no imposible, implementar esta característica en marcos de sitios web dinámicos como WordPress.

Básicamente, cuando navega por cualquier archivo de texto en un



repositorio en GitHub, puede editarlos directamente en la página presionando el botón Editar (mire la figura @ref(fig: github-edit) para ver un ejemplo) si tiene una cuenta de GitHub. Si tiene acceso de escritura al repositorio, puede confirmar los cambios directamente en línea, de lo contrario, GitHub bifurca automáticamente el repositorio para que pueda editar el archivo en su propio repositorio, y GitHub lo guiará para crear un pull request al repositorio original. Cuando el propietario original ve el pull request, puede ver los cambios que ha realizado y decidir si los acepta o no, o le pide que haga más cambios. Aunque la terminología “pull request” es muy confusa para los principiantes,<sup>18</sup>. Es probablemente la característica más importante inventada por GitHub, porque hace que sea mucho más fácil para las personas hacer contribuciones.

Lo que realmente es útil es que todo lo que necesita es una URL de forma fija para editar un archivo en GitHub: `https://github.com/USER/REPO/edit/BRANCH/PATH/TO/FILE`. Por ejemplo, `https://github.com/rbind/yihui/edit/master/content/knitr/faq.md`, donde USER es rbind, REPO es yihui, BRANCH es master, y la ruta del archivo es content/knitr/faq.md.

La clave para implementar esta característica es la variable `.File.Path`, que nos da la ruta del archivo fuente de una página bajo `content/`, por ejemplo, `post/foo.md`. Si su sitio web solo utiliza archivos de Markdown simples, la implementación será muy simple. Omití la URL completa de GitHub en ... a continuación, de la cual un ejemplo podría ser `https://github.com/rbind/yihui/edit/master/content/`.

```
{{ with .File.Path }}
<a href="https://github.com/.../{{ . }}">Edit this page</a>
{{ end }}
```

Sin embargo, el caso es un poco más complicado para los usuarios de **blogdown**, cuando se trata de publicaciones de R Markdown. No se puede usar `.File.Path` porque apunta al archivo de

<sup>18</sup>En mi opinión, en realidad debería llamarse “solicitud de fusión”

salida `.html` de un archivo `.Rmd`, mientras que el archivo `.Rmd` es el archivo fuente real. El botón o enlace Edit no debe apuntar al archivo `.html`. A continuación se muestra la implementación completa que puede agregar a un archivo de plantilla dependiendo de dónde desee mostrar el enlace Edit (por ejemplo, `footer.html`):

```
{{ if .File.Path }}

{{ $Rmd := (print .File.BaseFileName ".Rmd") }}

{{ if (where (readDir (print "content/" .File.Dir)) "Name" $Rmd) }}
  {{ $.Scratch.Set "FilePath" (print .File.Dir $Rmd) }}
{{ else }}
  {{ $.Scratch.Set "FilePath" .File.Path }}
{{ end }}

{{ with .Site.Params.GithubEdit}}
<a href='{{ . }}{{ $.Scratch.Get "FilePath" }}'>Edit this page</a>
{{ end }}

{{ end }}
```

La lógica básica es que, para un archivo, si existe el mismo nombre de archivo con la extensión `.Rmd`, señalaremos el enlace Edit al archivo `Rmd`. Primero, definimos una variable `$ Rmd` para que sea el nombre de archivo con la extensión `.Rmd`. Luego verificamos si existe. Desafortunadamente, no hay ninguna función en Hugo como `file.exists()` en R, así que tenemos que usar un truco: liste todos los archivos bajo el directorio y vea si el archivo `Rmd` está en la lista. `$ .Scratch`<sup>19</sup> es la forma de almacenar dinámicamente y obtener variables en las plantillas de Hugo. La mayoría de las variables en Hugo son de solo lectura, y usted tiene que usar `$.Scratch` cuando quiera modificar una variable. Establecemos una variable `FilePath` en `$.Scratch`, cuyo valor es la ruta

<sup>19</sup><http://gohugo.io/extras/scratch/>

completa al archivo `Rmd` cuando existe el archivo `Rmd`, y la ruta al archivo fuente de Markdown de lo contrario. Finalmente, concatenamos una opción personalizada `GithubEdit` en `config.toml` con la ruta del archivo para completar el enlace `Edit` <a>. Aquí hay un ejemplo de la opción en `config.toml`:

```
[params]
GithubEdit = "https://github.com/rbind/yihui/edit/master/content/"
```

Tenga en cuenta que si utiliza Hugo en Windows para compilar e implementar su sitio, es posible que tenga que cambiar los separadores de ruta de archivos de las barras diagonales inversas a barras diagonales, por ejemplo, puede necesitar `{{$.Scratch.Set "FilePath" (replace ($.Scratch.Get"FilePath") "\\\" "/" )}}` en la plantilla. Para evitar esta complicación, no recomendamos que implemente su sitio a través de Windows (consulte el capítulo ?? para conocer los métodos de implementación).

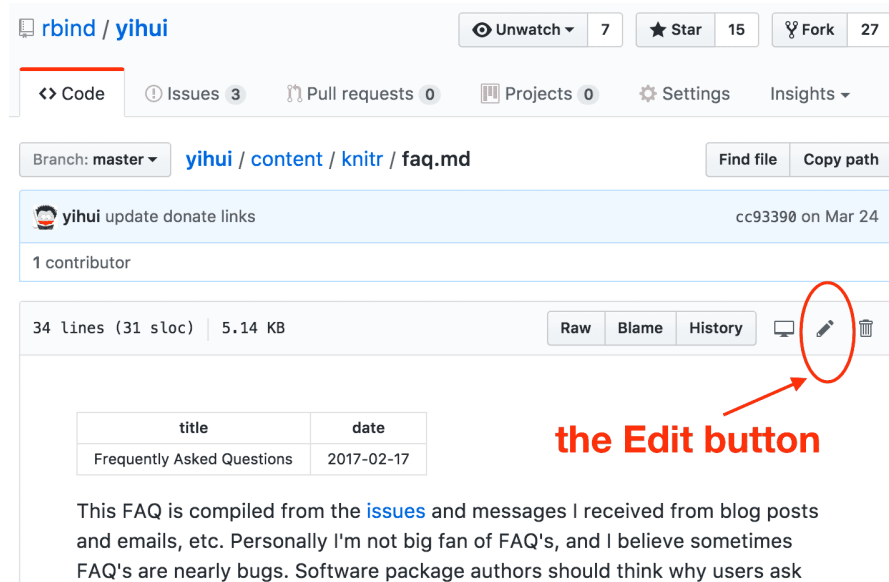
Puede ver <https://github.com/yihui/hugo-xmin/pull/6> para una implementación real con ejemplos de R Markdown, y ver el pie de página de esta página para el enlace Edit: <https://deploy-preview-6--hugo-xmin.netlify.com>. En realidad, puede ver un enlace en el pie de página de cada página, excepto las listas de páginas (porque no tienen archivos fuente).

Después de digerir el tema XMin y las implementaciones de funciones adicionales, debería ser mucho más fácil entender las plantillas de otras personas. Hay una gran cantidad de temas de Hugo, pero las principales diferencias entre ellos suelen ser estilos. Los componentes básicos de las plantillas son a menudo similares.

---

## 2.6 Layouts personalizados

Es muy probable que desee personalizar un tema a menos que lo haya diseñado. La forma más directa es simplemente hacer cambios



**FIGURE 2.3:** Editar un archivo de texto en línea en GitHub.

directamente en el tema,<sup>20</sup> Pero el problema es que un tema de Hugo puede ser actualizado constantemente por su autor original para mejoras o correcciones de errores. De manera similar a la política “la rompes, la compras” (la regla de Pottery Barn<sup>21</sup>), una vez que toca el código fuente de otra persona, será responsable de su mantenimiento futuro, y el autor original no debería ser responsable de los cambios que haya realizado de su lado. Eso significa que puede no ser fácil extraer actualizaciones futuras de este tema a su sitio web (debe leer cuidadosamente los cambios y asegurarse de que no entren en conflicto con sus cambios), pero si está completamente satisfecho con el estado actual del tema y no quiere actualizaciones futuras, está bien modificar los archivos de tema directamente.

Un autor de tema que tenga en cuenta el hecho de que los usar-

<sup>20</sup>Si es nuevo en el desarrollo web, tenga cuidado de cambiar el contenido dentro del tema. Pequeños cambios como colores y tamaños de fuente se pueden encontrar dentro de los archivos CSS del tema y pueden modificarse simplemente con el mínimo riesgo de romper la funcionalidad del tema.

<sup>21</sup>[https://en.wikipedia.org/wiki/Pottery\\_Barn\\_rule](https://en.wikipedia.org/wiki/Pottery_Barn_rule)

ios pueden personalizar su tema generalmente proporcionará dos maneras: una es proporcionar opciones en `config.toml`, para que pueda cambiar estas opciones sin tocar los archivos de la plantilla; la otra es dejar unos pocos archivos de plantilla livianos en ‘`layouts/`’ en el tema, para que pueda anularlos sin tocar los archivos de la plantilla principal. Tome el tema XMin por ejemplo:

Tengo dos archivos HTML vacíos `head_custom.html` y `foot_custom.html` en `layouts/partials/` en el tema. El primero se agregará dentro de `<head>` `</head>` de una página, por ejemplo, puede cargar librerías de JavaScript o incluir hojas de estilo CSS mediante `<link>`. Este último se agregará antes del pie de página de una página, por ejemplo, puede cargar librerías de JavaScript adicionales o incrustar comentarios de Disqus allí.

La forma en que personaliza estos dos archivos no es para editarlos directamente en la carpeta de temas, sino para crear un directorio `layouts/partials/` en el directorio raíz de su sitio web, por ejemplo, su estructura de directorios puede verse así:

```
your-website/
├── config.toml
├── ...
├── themes/
│   └── hugo-xmin/
│       ├── ...
│       └── layouts/
│           ├── ...
│           └── partials
│               ├── foot_custom.html
│               ├── footer.html
│               ├── head_custom.html
│               └── header.html
└── layouts
    └── partials
        ├── foot_custom.html
        └── head_custom.html
```

Todos los archivos en `layouts/` en el directorio raíz anularán los archivos con las mismas rutas relativas en `themes/hugo-xmin/layouts/`, por ejemplo, el archivo `layouts/partials/foot_custom.html`, cuando se proporcione, anulará `themes/hugo-xmin/layouts/partials/foot_custom.html`. Eso significa que solo necesita crear y mantener como máximo dos archivos en `layouts/` en lugar de mantener todos los archivos bajo `themes/`. Tenga en cuenta que este mecanismo de anulación se aplica a todos los archivos en `layouts/`, y no está limitado al directorio `partials/`. También se aplica a cualquier tema de Hugo que utilice realmente para su sitio web, y no se limita a `hugo-xmin`.

---

## 2.7 Archivos estáticos

Todos los archivos bajo el directorio `static/` se copian a `public/` cuando Hugo procesa un sitio web. Este directorio se usa a menudo para almacenar archivos web estáticos como imágenes, CSS y archivos de JavaScript. Por ejemplo, una imagen `static/foo/bar.png` se puede incrustar en su publicación usando la sintaxis Markdown ``.<sup>22</sup>

Por lo general, un tema tiene una carpeta `static/`, y puede anular parcialmente sus archivos utilizando el mismo mecanismo que reemplaza a los `layouts/` archivos, es decir, `static/file` anulará `themes/theme-name/static/file`. En el tema XMin, tengo dos archivos CSS `style.css` y `fonts.css`. El primero es la hoja de estilo principal, y el último es un archivo bastante pequeño para definir tipos de letra solamente. Es posible que desee definir sus propios tipos de letra, y solo puede proporcionar un `static/css/fonts.css` para anular el del tema, por ejemplo,

---

<sup>22</sup>El enlace de la imagen depende de su configuración `baseurl` en `config.toml`. Si no contiene un subtrayecto, `/foo/bar.png` será el enlace de la imagen; de lo contrario, tendrá que ajustarlo, por ejemplo, para `baseurl = "http://example.com/subpath/"`, el enlace a la imagen debe ser `/subpath/foo/bar.png`.

```
body {  
  font-family: "Comic Sans MS", cursive, sans-serif;  
}  
code {  
  font-family: "Courier New", Courier, monospace;  
}
```

Para los usuarios de R Markdown, otra aplicación importante del directorio `static/` es construir documentos Rmd con formatos de salida personalizados, es decir, documentos Rmd que no utilizan el formato `blogdown::html_page()` (ver sección ??). Por ejemplo, puede generar un PDF o presentaciones de documentos Rmd en este directorio, para que Hugo no los postprocesa, sino que simplemente los copie en `public/` para su publicación. Para compilar estos archivos Rmd, debe proporcionar un script de compilación personalizado `R/build.R` (consulte la sección ??). Puede escribir una sola línea de código en este script:

```
blogdown::build_dir("static")
```

La función `build_dir()` busca todos los archivos Rmd bajo un directorio y llama a `rmarkdown::render()` para compilarlos en los formatos de salida especificados en los metadatos YAML de los archivos Rmd. Si sus archivos Rmd no se deben presentar con una simple llamada `rmarkdown::render()`, puede proporcionar su propio código para presentarlos en `R/build.R`. Hay un mecanismo de caché integrado en la función `dir_desarrollo()`: un archivo Rmd no se compilará si es anterior a su archivo(s) de salida. Si no desea este comportamiento, puede obligar a todos los archivos Rmd a volver a compilarse cada vez: `build_dir(force = TRUE)`.

He proporcionado un ejemplo mínimo en el repositorio de GitHub `yihui/blogdown-static`,<sup>23</sup> donde puede encontrar dos ejemplos de

<sup>23</sup><https://github.com/yihui/blogdown-static>

Rmd en el directorio `static/`. Una es una presentación HTML5 basada en el paquete **xaringan**, y la otra es un documento PDF basado en **bookdown**.

Debe tener precaución con los archivos arbitrarios en `static/`, debido al mecanismo predominante de Hugo. Es decir, todo en `static/` se copiará en `public/`. Debe asegurarse de que los archivos que procesa en `static/` no entrarán en conflicto con los archivos generados automáticamente por Hugo a partir de `content/`. Por ejemplo, si tiene un archivo fuente `content/about.md` y un archivo Rmd `static/about/index.Rmd` al mismo tiempo, el resultado HTML de este último sobrescribirá el anterior (tanto Hugo como usted generarán un archivo de salida con el mismo nombre `public/about/index.html`).



# 3

## *Implementación*

Dado que el sitio web es básicamente una carpeta que contiene archivos estáticos, es mucho más fácil de implementar que los sitios web que requieren lenguajes dinámicos en el servidor, como PHP o bases de datos. Todo lo que necesita es subir los archivos a un servidor, y generalmente su sitio web estará en funcionamiento en breve. La pregunta clave es qué servidor web quiere usar. Si no tiene su propio servidor, puede probar los que figuran en este capítulo. La mayoría de ellos son gratuitos (excepto Amazon S3), o al menos ofrecen planes gratuitos. Descargo de responsabilidad: los autores de este libro no están afiliados a ninguno de estos servicios o compañías, y no hay garantía de que estos servicios se presten para siempre.<sup>1</sup>

Teniendo en cuenta el costo y la amabilidad de los principiantes, actualmente recomendamos Netlify (<https://www.netlify.com>). Proporciona un plan gratuito que en realidad tiene muchas funciones útiles. Si no tiene experiencia en publicar sitios web antes, solo inicie sesión con su cuenta GitHub u otras cuentas, arrastre la carpeta `public/` creada por **blogdown** para su sitio web a la página de Netlify, y su sitio web estará en línea en unos segundos con un nombre de subdominio aleatorio del formulario `random-word-12345.netlify.com` proporcionado por Netlify (puede personalizar el nombre). Puede automatizar fácilmente este proceso (consulte la sección 3.1 para obtener más información). Ya no necesita luchar con `ssh` o `rsync-zrvce`, si sabe lo que significan estos comandos.

La segunda solución más fácil puede ser Updog (<https://updog.co>), que cuenta con la integración de Dropbox. Publicar un sitio web

---

<sup>1</sup>Puede encontrar fácilmente otros servicios similares si usa su motor de búsqueda

puede ser tan fácil como copiar los archivos en la carpeta `public/` de su sitio web **blogdown** en una carpeta de Dropbox. El plan gratuito de Updog solo ofrece funciones limitadas, y su plan de pago le dará acceso a funciones mucho más ricas.

Si no le importa utilizar herramientas de línea de comandos o está familiarizado con GIT/GitHub, puede considerar servicios como GitHub Pages, Travis CI o Amazon S3 para construir o alojar sus sitios web. No importa qué servicio use, tenga en cuenta que ninguno de ellos realmente puede encerrarlo y siempre puede cambiar el servicio. Como mencionamos anteriormente, una gran ventaja de **blogdown** es que su sitio web será una carpeta de archivos estáticos que puede mover a cualquier servidor web.

---

### 3.1 Netlify

Como acabamos de mencionar, Netlify le permite publicar rápidamente un sitio web cargando la carpeta `public/` a través de su interfaz web, y se le asignará un subdominio aleatorio `*.netlify.com`.<sup>2</sup> Este enfoque es bueno para los sitios web que no se actualizan con frecuencia (o no se actualizan). Sin embargo, es poco probable que no necesite actualizar su sitio web, por lo que presentamos un mejor enfoque en esta sección,<sup>3</sup>. Le llevará unos minutos más completar las configuraciones. Una vez que está configurado correctamente, todo lo que necesita hacer en el futuro es actualizar el repositorio fuente, y Netlify llamará a Hugo para que haga su sitio web automáticamente.

Básicamente, debe alojar todos los archivos fuente de su sitio web

---

<sup>2</sup>Usted no tiene que mantener el dominio `*.netlify.com`. Consulte el apéndice @ref(nombre de dominio) para obtener más información.

<sup>3</sup>Tenga en cuenta que el propósito de esta sección es describir los pasos básicos de la publicación de un sitio web con Netlify, y los detalles técnicos pueden cambiar de vez en cuando, por lo que la documentación oficial de Netlify debería ser la fuente más confiable si tiene alguna pregunta o si alguna de las cosas que presentamos aquí no funciona

en un repositorio GIT.<sup>4</sup> No necesita poner el directorio `public/` bajo control de versión<sup>5</sup> porque se generará automáticamente. Actualmente, Netlify admite repositorios GIT alojados en GitHub, GitLab y BitBucket. Con cualquiera de estas cuentas, puede iniciar sesión en Netlify desde su página de inicio y seguir la guía para crear un nuevo sitio desde su repositorio de GIT.

Netlify es compatible con varios generadores de sitios web estáticos, incluidos Jekyll y Hugo. Para un nuevo sitio, debe especificar un comando para construir su sitio web, así como también la ruta del directorio de publicación. Netlify también admite múltiples versiones de Hugo, por lo que el comando de compilación puede ser el `hugo` predeterminado. La versión predeterminada es 0.17, que es demasiado antigua. Le recomendamos que utilice al menos la versión 0.20. Para especificar una versión de Hugo mayor o igual a 0.20, debe crear una variable de entorno `HUGO_VERSION` en Netlify. Consulte la documentación de Netlify<sup>6</sup> para obtener más información. El directorio de publicación debe ser `public` a menos que lo haya cambiado en su `config.toml`. La figura @ref(fig: configuración-netlify) muestra la configuración del sitio web <https://t.yihui.name>. No tiene que seguir la configuración exacta para su propio sitio web; en particular, es posible que necesite cambiar el valor de la variable de entorno `HUGO_VERSION` a una versión reciente de Hugo.<sup>7</sup>

Puede tardar uno o dos minutos en implementar su sitio web en Netlify por primera vez, pero puede ser mucho más rápido más adelante (unos segundos) cuando actualice el origen de su sitio web, porque Netlify implementa cambios incrementales en el directorio `public/`, es decir, solo se despliegan los archivos más nuevos en comparación con la última vez.

Después de que su repositorio de GIT esté conectado con Netlify, el último problema que puede querer resolver es el nombre de do-

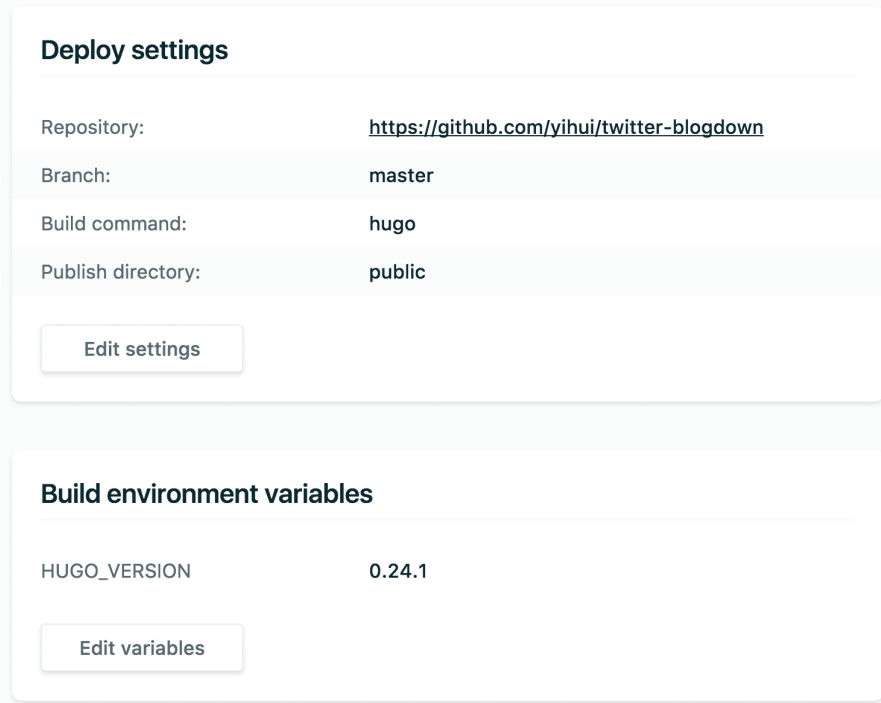
---

<sup>4</sup>Si el contenido de su sitio `blogdown` no está en el directorio raíz de su repositorio GIT, Netlify no se compilará.

<sup>5</sup>Puede agregar `public` a `.gitignore` para ignorarlo en GIT.

<sup>6</sup><https://www.netlify.com/docs/continuous-deployment/>

<sup>7</sup>Para el momento en que se publique este libro, la versión 0.24.1 puede ser demasiado antigua.



The image shows two sections of a Netlify configuration interface. The top section, titled 'Deploy settings', contains a table with the following information:

Repository:	<a href="https://github.com/yihui/twitter-blogdown">https://github.com/yihui/twitter-blogdown</a>
Branch:	master
Build command:	hugo
Publish directory:	public

Below the table is a button labeled 'Edit settings'.

The bottom section, titled 'Build environment variables', contains a table with the following information:

HUGO_VERSION	0.24.1
--------------	--------

Below the table is a button labeled 'Edit variables'.

**FIGURE 3.1:** Configuraciones de ejemplo de un sitio web presentado en Netlify.

minio, a menos que esté satisfecho con el subdominio gratuito de Netlify. Si desea utilizar un dominio diferente, debe configurar algunos registros DNS del dominio para dirigirlo al servidor de Netlify. Consulte el apéndice @ref(nombre de dominio) para obtener información general sobre los nombres de dominio.

Si no está familiarizado con los nombres de dominio o no quiere aprender más sobre ellos, debe tener en cuenta un subdominio gratuito \* .rbind.io ofrecido por RStudio, Inc. Visite el sitio web de soporte de Rbind <https://support.rbind.io> para aprender cómo solicitar un subdominio. De hecho, la organización Rbind también ofrece ayuda gratuita sobre cómo configurar un sitio web basado en **blogdown**, gracias a una gran cantidad de voluntarios de la comunidad de R y de estadística.

Netlify es la única solución en este capítulo que no requiere prein-

stalar su sitio web. Solo necesita actualizar los archivos fuente, enviarlos a GitHub y Netlify creará el sitio web para usted.<sup>8</sup>. El resto de las soluciones de este capítulo requerirán que cree su sitio web localmente. y cargue la carpeta `public/` explícita o implícitamente. Dicho esto, ciertamente puede preconstruir su sitio web utilizando cualquier herramienta, enviarlo a GitHub, y aún así Netlify lo implementará para usted. Lo que debe hacer es dejar el comando de compilación en blanco y decirle a Netlify su directorio de publicación (por ejemplo, `public/` por defecto de Hugo, pero si su sitio web preconstruído está bajo el directorio raíz, especifique `.` como el directorio de publicación). Entonces Netlify simplemente carga todos los archivos de este directorio a sus servidores sin reconstruir su sitio web.

---

## 3.2 Updog

Updog (<https://updog.co>) proporciona un servicio simple: convierte una carpeta de Dropbox (o Google Drive) especificada en un sitio web. La idea es que le conceda a Updog el permiso para leer la carpeta, y actuará como intermediario para mostrar sus archivos en esta carpeta a sus visitantes. Se debe acceder a esta carpeta a través de un nombre de dominio, y Updog ofrece un subdominio gratuito `*.updog.co`. Por ejemplo, si ha asignado el dominio `example.updog.co` a su carpeta de Dropbox, y un visitante desea ver la página `https://example.updog.co/foo/index.html`, Updog leerá el archivo `foo/index.html` en su carpeta de Dropbox y lo mostrará al visitante.

Por el momento, el plan gratuito de Updog solo permite un sitio web por cuenta e insertará un pie de página “Hosted on Updog” en sus páginas web. Puede que no le gusten estas limitaciones. La principal ventaja de usar Updog es que la publicación de un sitio web se vuelve implícita, ya que Dropbox sincronizará

---

<sup>8</sup>Esto se denomina “implementación continua”

archivos continuamente. Todo lo que necesita hacer es asegurarse de que su sitio web se genere en la carpeta correcta de Dropbox. Esto se puede lograr fácilmente estableciendo la opción `publishDir` en `config.toml`. Por ejemplo, supongamos que la carpeta que asigna a Updog es `~/Dropbox/Apps/updog/my-website/`, y su carpeta fuente está en `~/Dropbox/Apps/updog/my-source/`, entonces puede establecer `publishDir: "../my-website"` en `~/Dropbox/Apps/updog/my-source/config.toml`.

También puede usar su nombre de dominio personalizado si no desea el subdominio Updog predeterminado, y solo necesita apuntar el registro CNAME de su nombre de dominio al subdominio Updog.<sup>9</sup>

---

### 3.3 GitHub Pages

GitHub Pages (<https://pages.github.com>) es una forma muy popular de alojar sitios web estáticos (especialmente los creados con Jekyll), pero sus ventajas no son obvias ni atractivas en comparación con Netlify. Le recomendamos que considere Netlify + Hugo debido a estas razones:

- Actualmente, GitHub Pages no es compatible con HTTPS para nombres de dominio personalizados. HTTPS solo funciona para los subdominios `*.github.io`. Esta limitación no existe en Netlify. Puede leer el artículo “¿Por qué HTTPS para todo?”<sup>10</sup> para comprender por qué es importante y se le anima a activar HTTPS para su sitio web siempre que sea posible.
- Redirigir URLs es incómodo con GitHub Pages pero mucho más sencillo con Netlify.<sup>11</sup> Esto es importante especialmente cuando

---

<sup>9</sup>Vea el apéndice @ref(nombre de dominio) para obtener más información.

<sup>10</sup><https://https.cio.gov/everything/>

<sup>11</sup>GitHub Pages utiliza un plugin Jekyll para escribir una metaetiqueta `HTTP-REFRESH` para redirigir páginas, y Netlify puede hacer redirecciones 301 o 302

tiene un sitio web antiguo que desea migrar a Hugo; algunos enlaces pueden estar rotos, en cuyo caso puede redireccionarlos fácilmente con Netlify.

- Una de las mejores características de Netlify que no está disponible en GitHub Pages es que Netlify puede generar un sitio web único para la vista previa cuando se envía un pull request de GitHub a su repositorio de GitHub. Esto es extremadamente útil cuando otra persona (o incluso usted mismo) propone cambios en su sitio web, ya que tiene la oportunidad de ver cómo se vería el sitio web antes de fusionar el pull request.

Básicamente, Netlify puede hacer todo lo que GitHub Pages puede, pero todavía hay una pequeña característica que falta, que está estrechamente vinculada a GitHub, que es GitHub Project Pages.<sup>12</sup> Esta función le permite tener sitios web de proyectos en repositorios separados, por ejemplo, puede tener dos sitios web independientes `https://username.github.io/proj-a/` y `https://username.github.io/proj-b/`, que corresponde a los repositorios de GitHub `username/proj-a` y `username/proj-b`, respectivamente. Sin embargo, dado que puede conectar cualquier repositorio de GitHub con Netlify, y cada repositorio puede asociarse con un nombre de dominio o subdominio, puede reemplazar las páginas de proyecto de GitHub con diferentes subdominios como `proj-a.netlify.com` y `proj-b.netlify.com`. La limitación real es que no puede usar subcampos en la URL pero puede usar cualquier (sub)nombre de dominio.

Aunque GitHub no es compatible oficialmente con Hugo (solo es compatible con Jekyll), puede publicar cualquier archivo HTML estático en GitHub Pages, incluso si no están compiladas con Jekyll. El primer requisito para usar GitHub Pages es que debe crear un repositorio de GitHub llamado `username.github.io` debajo de su cuenta (reemplace `username` con su nombre de usuario GitHub real), y lo que queda es llevar los archivos de su sitio web a este

basadas en patrones, que puede notificar a los motores de búsqueda que ciertas páginas se han movido (de forma permanente o temporal).

<sup>12</sup><https://help.github.com/articles/user-organization-and-project-pages/>

repositorio. La documentación completa de GitHub Pages está en <https://pages.github.com>, y por favor ignore todo lo relacionado con Jekyll a menos que realmente use Jekyll en lugar de Hugo. Para asegurarse de que GitHub no reconstruya su sitio web utilizando Jekyll y simplemente publique los archivos que envía al repositorio, debe crear un archivo (oculto) llamado `.nojekyll` en el repositorio.<sup>13</sup> GitHub ofrece un subdominio gratuito `username.github.io`, y puede usar su propio nombre de dominio configurando sus registros A o CNAME para apuntarlo a GitHub Pages (consulte la documentación de GitHub Pages para obtener instrucciones).

Su directorio `public/` debe ser el repositorio de GIT. Tienes dos opciones posibles para configurar este repositorio localmente. La primera opción es seguir la estructura predeterminada de un sitio web de Hugo como el siguiente diagrama e inicializar el repositorio de GIT bajo el directorio `public/`:

```
source/
|
├─ config.toml
├─ content/
├─ themes/
├─ ...
└─ public/
    |
    ├─ .git/
    ├─ .nojekyll
    ├─ index.html
    ├─ about/
    └─ ...
```

Si sabe cómo usar la línea de comandos, cambie el directorio de trabajo a `public/`, e inicialice el repositorio de GIT allí:

---

<sup>13</sup>Puede usar la función en R `file.create('.nojekyll')` para crear este archivo si no sabe cómo hacerlo.



```
cd public
git init
git remote add origin https://github.com/username/username.github.io
```

La otra opción es clonar el repositorio de GitHub que creó en el mismo directorio que el origen de su sitio web:

```
git clone https://github.com/username/username.github.io
```

Y la estructura debería lucir más o menos así:

```
source/
|
├─ config.toml
├─ content/
├─ themes/
└─ ...

username.github.io/
|
├─ .git/
├─ .nojekyll
├─ index.html
├─ about/
└─ ...
```

El directorio de origen y el directorio `username.github.io` están bajo el mismo directorio principal. En este caso, debe establecer la opción `publishDir: "../username.github.io"` en `source/config.toml`.

---

### 3.4 Travis + GitHub

Si decide no seguir nuestra recomendación de usar Netlify para implementar su sitio web, debemos advertirle que el enfoque de esta sección requerirá un conocimiento sustancial sobre GIT, GitHub, Travis CI (<https://travis-ci.org>), y la línea de comandos de Linux, que dejaremos que aprenda por su cuenta. La principal ventaja de publicar a través de Travis CI es que puede compilar todas sus publicaciones de Rmd en Travis CI (en la nube) en lugar de su computadora local.

En caso de que no esté familiarizado con Travis, este es un servicio de verificación continua de su software en una máquina virtual cada vez que haga push a cambios en GitHub. Es principalmente para probar software, pero dado que puede ejecutar muchos comandos en su máquina virtual, puede usar la máquina virtual para hacer otras cosas, por ejemplo, instalar R y el paquete **blogdown** para crear sitios web. Antes de mostrarle cómo, me gustaría mencionar dos cuestiones que debe tener en cuenta:

- Personalmente, prefiero echar un vistazo a la salida en GIT para ver los cambios cuando tengo cualquier salida que se calcula dinámicamente desde R, para que sepa con certeza qué voy a publicar exactamente. Con Travis, es algo impredecible porque es completamente automático y no tiene la oportunidad de ver el nuevo contenido o los resultados que se publicarán. Hay muchos factores que pueden afectar la construcción del sitio: la versión de R, la disponibilidad de ciertos paquetes en R, las dependencias del sistema y la conexión de red, etc.
- El tiempo requerido para compilar todos los archivos Rmd puede ser muy largo y causar tiempos de espera en Travis, dependiendo de cuánto tiempo consuma su código en R. Hay un mecanismo de almacenamiento en caché en **blogdown** para acelerar la construcción de su sitio (consulte la sección ??), y si usa Travis para construir su sitio web, no se beneficiará de este mecanismo

de almacenamiento en caché a menos que aproveche el almacenamiento en caché de Travis. Tiene que almacenar en caché los directorios `content/`, `static/`, y `blogdown/`, pero el caché de Travis es un poco frágil, en mi experiencia. Algunas veces la memoria caché puede ser purgada por razones desconocidas. Además, no puede almacenar en caché directamente `content/` y `static/`, porque Travis clona su repositorio antes de restaurar el caché, lo que significa que los archivos viejos del `content/` y `static/` almacenados en caché pueden sobrescribir los nuevos archivos que usted envió a GitHub.

El segundo problema se puede resolver, pero no quiero explicar cómo en este libro, ya que la solución es demasiado complicada. Si realmente desea usar Travis para construir su sitio web y encontrarse con este problema, puede presentar un issue en el repositorio de GitHub <https://github.com/yihui/travis-blogdown>. De hecho, este repositorio es un ejemplo mínimo que creé para mostrar cómo crear un sitio web en Travis y publicarlo en GitHub Pages.

La documentación de Travis muestra cómo implementar un sitio en GitHub Pages: <https://docs.travis-ci.com/user/deployment/pages/>, pero no muestra cómo crear un sitio. Aquí está el archivo de configuración de Travis, `.travis.yml`, para el repositorio `travis-blogdown`:

```
language: r
dist: trusty
sudo: false

branches:
  only:
    - master

cache:
  packages: yes
  directories:
    - $HOME/bin
```

```
before_script:
  - "Rscript -e 'blogdown::install_hugo()'"

script:
  - "Rscript -e 'blogdown::build_site()'"

deploy:
  provider: pages
  skip_cleanup: true
  github_token: $GITHUB_TOKEN
  on:
    branch: master
  local_dir: public
  fqdn: travis-blogdown.yihui.name
```

La clave es que instalemos Hugo a través de `blogdown::install_hugo()` y construyamos el sitio a través de `blogdown::build_site()`. Para engañar a Travis para que cree este repositorio como un paquete en R, debe tener un archivo `DESCRIPTION` en el repositorio, de lo contrario, su sitio web no se compilará.

```
Package: placeholder
Type: Website
Title: Does not matter.
Version: 0.0.1
Imports: blogdown
Remotes: rstudio/blogdown
```

Hay algunas cosas más que explicar y enfatizar en `.travis.yml`:

- La opción `branches` especifica que solo los cambios en la rama `master` activarán la construcción en Travis.

- La opción `cache` especifica todos los paquetes en R que se almacenarán en caché, por lo que la próxima vez será más rápido crear el sitio (no es necesario volver a instalar los paquetes en R desde el origen). El directorio `bin/` en el directorio de inicio también se almacena en caché porque Hugo está instalado allí, y la próxima vez que Hugo no necesite ser reinstalado.
- Para la opción `deploy`, hay una variable de entorno llamada `GITHUB_TOKEN`, y he especificado su valor para ser un token de acceso personal de GitHub a través de la configuración de Travis de este repositorio, para que Travis pueda escribir en mi repositorio después de que el sitio web está construido. La opción `on` especifica que la implementación solo ocurrirá cuando se construya la rama `master`. La opción `local_dir` es el directorio de publicación, que debe ser ‘público’ por defecto en Hugo. Por defecto, el sitio web se envía a la rama `gh-pages` de este repositorio. La opción `fqdn` especifica el dominio personalizado del sitio web. He establecido un registro CNAME (ver apéndice @ref(nombre de dominio)) para apuntar `travis-blogdown.yihui.name` a `yihui.github.io`, para que GitHub pueda servir a este sitio web a través de este dominio (de hecho, Travis escribirá un archivo `CNAME` que contiene el dominio en la rama `gh-pages`).

Si utiliza el repositorio `username.github.io` en GitHub, el sitio web debe ser enviado a su rama `master` en lugar de `gh-pages` (esta es la única excepción). Recomiendo que separe el repositorio fuente y el repositorio de salida. Por ejemplo, puede tener un repositorio `website-source` con la misma configuración que el `.travis.yml` anterior, excepto dos nuevas opciones bajo `deploy`:

```
deploy:
  ...
  repo: username/username.github.io
  target_branch: master
```

Esto significa que el sitio web será enviado a la rama `master` del

repositorio `username/username.github.io` (recuerde reemplazar `username` con su nombre de usuario real).

También puede implementar su sitio web en Amazon S3, y la configuración desde R es muy similar a lo que hemos introducido para GitHub Pages. La única diferencia está en el último paso, donde cambia el destino de GitHub Pages a Amazon S3. Para obtener más información, consulte la documentación de Travis: <https://docs.travis-ci.com/user/deployment/s3/>.

---

### 3.5 GitLab Pages

GitLab (<http://gitlab.com>) es una forma muy popular de alojar el código fuente de su proyecto. GitLab tiene un servicio de Integración y Despliegue Integrado (CI/CD)<sup>14</sup> que se puede usar para alojar sitios web estáticos, llamados Páginas de GitLab<sup>15</sup>. La principal ventaja de utilizar GitLab Pages es que podrá compilar todas sus publicaciones Rmd a través de su servicio CI/CD en lugar de su computadora local y cualquier contenido generado, como archivos HTML, se copiará automáticamente en el servidor web. Tenga en cuenta que este enfoque tiene problemas similares a los del enfoque Travis + GitHub en la sección 3.4.

El servicio CI/CD de GitLab usa las instrucciones almacenadas en el archivo YAML `.gitlab-ci.yml` en el repositorio. Aquí hay un archivo de configuración de muestra `.gitlab-ci.yml` del repositorio de ejemplo <https://gitlab.com/rgaiacs/blogdown-gitlab>:

```
image: debian:buster-slim

before_script:
  - apt-get update && apt-get -y install pandoc r-base
```

---

<sup>14</sup><https://about.gitlab.com/features/gitlab-ci-cd/>

<sup>15</sup><https://about.gitlab.com/features/pages/>

```
- R -e "install.packages('blogdown',repos='http://cran.rstudio.com')"  
- R -e "blogdown::install_hugo()"  
  
pages:  
  script:  
    - R -e "blogdown::build_site()"  
  artifacts:  
    paths:  
      - public  
  only:  
    - master
```

La opción `image` especifica qué imagen de Docker<sup>16</sup> se usará como punto de inicio. Estamos utilizando una imagen de Debian, pero se puede usar cualquier imagen de Docker Hub<sup>17</sup>. Otras configuraciones y opciones son similares a `.travis.yml` en la sección 3.4. El ejemplo anterior genera el sitio web en <https://rgaiacs.gitlab.io/blogdown-gitlab>.

---

<sup>16</sup><https://www.docker.com>

<sup>17</sup><https://hub.docker.com/>





## 4

---

### *Migration*

---

Usually, it is easier to start a new website than migrating an old one to a new framework, but you may have to do it anyway because of the useful content on the old website that should not simply be discarded. A lazy solution is to leave the old website as is, start a new website with a new domain, and provide a link to the old website. This may be a hassle to your readers, and they may not be able to easily discover the gems that you created on your old website, so I recommend that you migrate your old posts and pages to the new website if possible.

This process may be easy or hard, depending on how complicated your old website is. The bad news is that there is unlikely to be a universal or magical solution, but I have provided some helper functions in **blogdown** as well as a Shiny application to assist you, which may make it a little easier for you to migrate from Jekyll and WordPress sites.

To give you an idea about the possible amount of work required, I can tell you that it took me a whole week (from the morning to midnight every day) to migrate several of my personal Jekyll-based websites to Hugo and **blogdown**. The complication in my case was not only Jekyll, but also the fact that I built several separate Jekyll websites (because I did not have a choice in Jekyll) and I wanted to unite them in the same repository. Now my two blogs (Chinese and English), the **knitr** (Xie, 2018b) package documentation, and the **animation** package (?) documentation are maintained in the same repository: <https://github.com/rbind/yihui>. I have about 1000 pages on this website, most of which are blog posts. It used to take me more than 30 seconds to preview my blog in Jekyll, and now it takes less than 2 seconds to build the site in Hugo.

Another complicated example is the website of Rob J Hyndman (<https://robjhyndman.com>). He started his website in 1993 (12 years before me), and had accumulated a lot of content over the years. You can read the post <https://support.rbind.io/2017/05/15/converting-robjhyndman-to-blogdown/> for the stories about how he migrated his WordPress website to **blogdown**. The key is that you probably need a long international flight when you want to migrate a complicated website.

A simpler example is the Simply Statistics blog (<https://simplystatistics.org>). Originally it was built on Jekyll<sup>1</sup> and the source was hosted in the GitHub repository <https://github.com/simplystats/simplystats.github.io>. I volunteered to help them move to **blogdown**, and it took me about four hours. My time was mostly spent on cleaning up the YAML metadata of posts and tweaking the Hugo theme. They had about 1000 posts, which sounds like a lot, but the number does not really matter, because I wrote an R script to process all posts automatically. The new repository is at <https://github.com/rbind/simplystats>.

If you do not really have too many pages (e.g., under 20), I recommend that you cut and paste them to Markdown files, because it may actually take longer to write a script to process these pages.

It is likely that some links will be broken after the migration because Hugo renders different links for your pages and posts. In that case, you may either fix the permanent links (e.g., by tweaking the slug of a post), or use 301 redirects (e.g., on Netlify).

---

## 4.1 From Jekyll

When converting a Jekyll website to Hugo, the most challenging part is the theme. If you want to keep exactly the same theme, you will have to rewrite your Jekyll templates using Hugo's syntax

---

<sup>1</sup>It was migrated from WordPress a few years ago. The WordPress site was actually migrated from an earlier Tumblr blog.

(see Section ??). However, if you can find an existing theme in Hugo (<https://themes.gohugo.io>), things will be much easier, and you only need to move the content of your website to Hugo, which is relatively easy. Basically, you copy the Markdown pages and posts to the `content/` directory in Hugo and tweak these text files.

Usually, posts in Jekyll are under the `_posts/` directory, and you can move them to `content/post/` (you are free to use other directories). Then you need to define a custom rule for permanent URLs in `config.toml` like (see Section ??):

```
[permalinks]
  post = "[:year/]:month/:day/:slug/"
```

This depends on the format of the URLs you used in Jekyll (see the `permalink` option in your `_config.yml`).

If there are static assets like images, they can be moved to the `static/` directory in Hugo.

Then you need to use your favorite tool with some string manipulation techniques to process all Markdown files. If you use R, you can list all Markdown files and process them one by one in a loop. Below is a sketch of the code:

```
files = list.files(
  'content/', '[.](md|markdown)$', full.names = TRUE,
  recursive = TRUE
)
for (f in files) {
  blogdown::process_file(f, function(x) {
    # process x here and return the modified x
    x
  })
}
```

The `process_file()` function is an internal helper function in **blogdown**. It takes a filename and a processor function to manipulate the content of the file, and writes the modified text back to the file.

To give you an idea of what a processor function may look like, I provided a few simple helper functions in **blogdown**, and below are two of them:

```
blogdown:::remove_extra_empty_lines
```

```
function (f)
  process_file(f, function(x) {
    x = paste(gsub("\\s+$", "", x), collapse = "\n")
    trim_ws(gsub("\n{3,}", "\n\n", x))
  })
<environment: namespace:blogdown>
```

```
blogdown:::process_bare_urls
```

```
function (f)
  process_file(f, function(x) {
    gsub("\\\\([^[^]]+)]\\\\(\\1/?\\\\)", "<\\1>", x)
  })
<environment: namespace:blogdown>
```

The first function substitutes two or more empty lines with a single empty line. The second function replaces links of the form `[url](url)` with `<url>`. There is nothing wrong with excessive empty lines or the syntax `[url](url)`, though. These helper functions may make your Markdown text a little cleaner. You can find all such helper functions at <https://github.com/rstudio/blogdown/blob/master/R/clean.R>. Note they are not exported from **blogdown**, so you need triple colons to access them.

The YAML metadata of your posts may not be completely clean, especially when your Jekyll website was actually converted from an earlier WordPress website. The internal helper function `blogdown::modify_yaml()` may help you clean up the metadata. For example, below is the YAML metadata of a blog post of Simply Statistics when it was built on Jekyll:

```
---
id: 4155
title: Announcing the JHU Data Science Hackathon 2015
date: 2015-07-28T13:31:04+00:00
author: Roger Peng
layout: post
guid: http://simplystatistics.org/?p=4155
permalink: /2015/07/28/announcing-the-jhu-data-science-hackathon-2015
pe_theme_meta:
  - '0:8:"stdClass":2:{s:7:"gallery";0:8:"stdClass":...}'
al2fb_facebook_link_id:
  - 136171103105421_837886222933902
al2fb_facebook_link_time:
  - 2015-07-28T17:31:11+00:00
al2fb_facebook_link_picture:
  - post=http://simplystatistics.org/?al2fb_image=1
dsq_thread_id:
  - 3980278933
categories:
  - Uncategorized
---
```

You can discard the YAML fields that are not useful in Hugo. For example, you may only keep the fields `title`, `author`, `date`, `categories`, and `tags`, and discard other fields. Actually, you may also want to add a `slug` field that takes the base filename of the post (with the leading date removed). For example, when the post filename is `2015-07-28-announcing-the-jhu-data-science-hackathon-2015.md`, you may want to add `slug: announcing-the-jhu-`

data-science-hackathon-2015 to make sure the URL of the post on the new site remains the same.

Here is the code to process the YAML metadata of all posts:

```
for (f in files) {
  blogdown:::modify_yaml(f, slug = function(old, yaml) {
    # YYYY-mm-dd-name.md -> name
    gsub('^\\d{4}-\\d{2}-\\d{2}-|\\.|(md|markdown)', '', f)
  }, categories = function(old, yaml) {
    # remove the Uncategorized category
    setdiff(old, 'Uncategorized')
  }, .keep_fields = c(
    'title', 'author', 'date', 'categories', 'tags', 'slug'
  ), .keep_empty = FALSE)
}
```

You can pass a file path to `modify_yaml()`, define new YAML values (or functions to return new values based on the old values), and decide which fields to preserve (`.keep_fields`). You may discard empty fields via `.keep_empty = FALSE`. The processed YAML metadata is below, which looks much cleaner:

```
---
title: Announcing the JHU Data Science Hackathon 2015
author: Roger Peng
date: '2015-07-28T13:31:04+00:00'
slug: announcing-the-jhu-data-science-hackathon-2015
---
```

---

## 4.2 From WordPress

From our experience, the best way to import WordPress blog posts to Hugo is to import them to Jekyll, and write an R script to

clean up the YAML metadata of all pages if necessary, instead of using the migration tools listed on the official guide,<sup>2</sup> including the WordPress plugin `wordpress-to-hugo-exporter`.

To our knowledge, the best tool to convert a WordPress website to Jekyll is the Python tool `Exitwp`.<sup>3</sup> Its author has provided detailed instructions on how to use it. You have to know how to install Python libraries and execute Python scripts. If you do not, I have provided an online tool at <https://github.com/yihui/travis-exitwp>. You can upload your WordPress XML file there, and get a download link to a ZIP archive that contains your posts in Markdown.

The biggest challenge in converting WordPress posts to Hugo is to clean up the post content in Markdown. Fortunately, I have done this for three different WordPress blogs,<sup>4</sup> and I think I have managed to automate this process as much as possible. You may refer to the pull request I submitted to Karl Broman to convert his WordPress posts to Markdown ([https://github.com/kbroman/oldblog\\_xml/pull/1](https://github.com/kbroman/oldblog_xml/pull/1)), in which I provided both the R script and the Markdown files. I recommend that you go to the “Commits” tab and view all my GIT commits one by one to see the full process.

The key is the R script [https://github.com/yihui/oldblog\\_xml/blob/master/convert.R](https://github.com/yihui/oldblog_xml/blob/master/convert.R), which converts the WordPress XML file to Markdown posts and cleans them. Before you run this script on your XML file, you have to adjust a few parameters, such as the XML filename, your old WordPress site’s URL, and your new blog’s URL.

Note that this script depends on the `Exitwp` tool. If you do not know how to run `Exitwp`, please use the online tool I mentioned before (`travis-exitwp`), and skip the R code that calls `Exitwp`.

The Markdown posts should be fairly clean after the conversion,

---

<sup>2</sup><https://gohugo.io/tools/>

<sup>3</sup><https://github.com/thomasf/exitwp>

<sup>4</sup>The RViews blog (<https://rviews.rstudio.com>), the RStudio blog (<https://blog.rstudio.com>), and Karl Broman’s blog (<http://kbroman.org>). The RViews blog took me a few days. The RStudio blog took me one day. Karl Broman’s blog took me an hour.

but there may be remaining HTML tags in your posts, such as `<table>` and `<blockquote>`. You will need to manually clean them, if any exist.

---

### 4.3 From other systems

If you have a website built by other applications or systems, your best way to go may be to import your website to WordPress first, export it to Jekyll, and clean up the Markdown files. You can try to search for solutions like “how to import blogger.com to WordPress” or “how to import Tumblr to WordPress.”

If you are very familiar with web scraping techniques, you can also scrape the HTML pages of your website, and convert them to Markdown via Pandoc, e.g.,

```
rmarkdown::pandoc_convert(  
  'foo.html', to = 'markdown', output = 'foo.md'  
)
```

I have actually tried this way on a website, but was not satisfied, since I still had to heavily clean up the Markdown files. If your website is simpler, this approach may work better for you.



# 5

---

## *Other Generators*

---

We mentioned the possibility to bypass Hugo and use your own building method in Section D.9. Basically you have to build the site using `blogdown::build_site(method = "custom")`, and provide your own building script `/R/build.R`. In this chapter, we show you how to work with other popular static site generators like Jekyll and Hexo. Besides these static site generators written in other languages, there is actually a simple site generator written in R provided in the **rmarkdown** package (Allaire et al., 2018), and we will introduce it in Section 5.3.

---

### 5.1 Jekyll

For Jekyll (<https://jekyllrb.com>) users, I have prepared a minimal example in the GitHub repository `yihui/blogdown-jekyll`.<sup>1</sup> If you clone or download this repository and open `blogdown-jekyll.Rproj` in RStudio, you can still use all addins mentioned in Section 1.3, such as “New Post,” “Serve Site,” and “Update Metadata,” but it is Jekyll instead of Hugo that builds the website behind the scenes now.

I assume you are familiar with Jekyll, and I’m not going to introduce the basics of Jekyll in this section. For example, you should know what the `_posts/` and `_site/` directories mean.

The key pieces of this **blogdown-jekyll** project are the files `.Rproj`

---

<sup>1</sup><https://github.com/yihui/blogdown-jekyll>

file, `R/build.R`, and `R/build_one.R`. I have set some global R options for this project in `.Rprofile`:<sup>2</sup>

```
options(
  blogdown.generator = "jekyll",
  blogdown.method = "custom",
  blogdown.subdir = "_posts"
)
```

First, the website generator was set to `jekyll` using the option `blogdown.generator`, so that **blogdown** knows that it should use Jekyll to build the site. Second, the build method `blogdown.method` was set to `custom`, so that we can define our custom R script `R/build.R` to build the Rmd files (I will explain the reason later). Third, the default subdirectory for new posts was set to `_posts`, which is Jekyll's convention. After you set this option, the “New Post” addin will create new posts under the `_posts/` directory.

When the option `blogdown.method` is `custom`, **blogdown** will call the R script `R/build.R` to build the site. You have full freedom to do whatever you want in this script. Below is an example script:

```
build_one = function(io) {
  # if output is not older than input, skip the compilation
  if (!blogdown:::require_rebuild(io[2], io[1])) return()

  message('* knitting ', io[1])
  if (blogdown:::Rscript(shQuote(c('R/build_one.R', io))) != 0) {
    unlink(io[2])
    stop('Failed to compile ', io[1], ' to ', io[2])
  }
}
```

---

<sup>2</sup>If you are not familiar with this file, please read Section ??.

```
# Rmd files under the root directory
rmds = list.files('.', '[.]Rmd$', recursive = T, full.names = T)
files = cbind(rmds, xfun::with_ext(rmds, '.md'))

for (i in seq_len(nrow(files))) build_one(files[i, ])

system2('jekyll', 'build')
```

- Basically it contains a function `build_one()` that takes an argument `io`, which is a character vector of length 2. The first element is the input (Rmd) filename, and the second element is the output filename.
- Then we search for all Rmd files under the current directory, prepare the output filenames by substituting the Rmd file extensions with `.md`, and build the Rmd files one by one. Note there is a caching mechanism in `build_one()` that makes use of an internal **blogdown** function `require_rebuild()`. This function returns `FALSE` if the output file is not older than the input file in terms of the modification time. This can save you some time because those Rmd files that have been compiled before will not be compiled again every time. The key step in `build_one()` is to run the R script `R/build_one.R`, which we will explain later.
- Lastly, we build the website through a system call of the command `jekyll build`.

The script `R/build_one.R` looks like this (I have omitted some non-essential settings for simplicity):

```
local({
  # fall back on "/" if baseurl is not specified
  baseurl = blogdown::get_config2("baseurl", default = "/")
  knitr::opts_knit$set(base.url = baseurl)
  knitr::render_jekyll() # set output hooks
```

```

# input/output filenames as two arguments to Rscript
a = commandArgs(TRUE)
d = gsub("^_|[.][a-zA-Z]+$", "", a[1])
knitr::opts_chunk$set(
  fig.path = sprintf("figure/%s/", d),
  cache.path = sprintf("cache/%s/", d)
)
knitr::knit(
  a[1], a[2], quiet = TRUE, encoding = "UTF-8",
  envir = globalenv()
)
})

```

- The script is wrapped in `local()` so that an Rmd file is knitted in a clean global environment, and the variables such as `baseUrl`, `a`, and `d` will not be created in the global environment, i.e., `globalenv()` used by `knitr::knit()` below.
- The **knitr** package option `base.url` is a URL to be prepended to figure paths. We need to set this option to make sure figures generated from R code chunks can be found when they are displayed on a web page. A normal figure path is often like `figure/foo.png`, and it may not work when the image is rendered to an HTML file, because `figure/foo.png` is a relative path, and there is no guarantee that this image file will be copied to the directory of the final HTML file. For example, for an Rmd source file `_posts/2015-07-23-hello.Rmd` that generates `figure/foo.png` (under `_posts/`), the final HTML file may be `_site/2015/07/23/hello/index.html`. Jekyll knows how to render an HTML file to this location, but it does not understand the image dependency and will not copy the image file to this location. To solve this issue, we render figures at the root directory `/figure/`, which will be copied to `_site/` by Jekyll. To refer to an image under `_site/figure/`, we need the leading slash (`baseUrl`), e.g., ``. This is an absolute path, so no matter where the HTML is rendered, this path always works.

- What `knitr::render_jekyll()` does is mainly to set up some **knitr** output hooks so that source code and text output from R code chunks will be wrapped in Liquid tags `{% highlight %}` and `{% end highlight %}`.
- Remember in `build.R`, we passed the variable `io` to the Rscript call `blogdown::Rscript`. Here in `build_one.R`, we can receive them from `commandArgs(TRUE)`. The variable `a` contains an `.Rmd` and an `.md` file path. We removed the possible leading underscore (`^_`) and the extension (`[a-zA-Z]$`) in the path. Next we set figure and cache paths using this string. For example, for a post `_posts/foo.Rmd`, its figures will be written to `figure/foo/` and its cache databases (if there are any) will be stored under `cache/foo/`. Both directories are under the root directory of the project.
- Lastly, we call `knitr::knit()` to knit the Rmd file to a Markdown output file, which will be processed by Jekyll later.

A small caveat is that since we have both `.Rmd` and `.md` files, Jekyll will treat both types of files as Markdown files by default. You have to ask Jekyll to ignore `.Rmd` files and only build `.md` files. You can set the option `exclude` in `_config.yml`:

```
exclude: ['*.Rmd']
```

Compared to the Hugo support in **blogdown**, this approach is limited in a few aspects:

1. It does not support Pandoc, so you cannot use Pandoc's Markdown. Since it uses the **knitr** package instead of **rmarkdown**, you cannot use any of **bookdown**'s Markdown features, either. You are at the mercy of the Markdown renderers supported by Jekyll.
2. Without **rmarkdown**, you cannot use HTML widgets. Basically, all you can have are dynamic text output and R graphics output from R code chunks. They may or may not suffice, depending on your specific use cases.

It may be possible for us to remove these limitations in a future version of **blogdown**, if there are enough happy Jekyll users in the R community.

---

## 5.2 Hexo

The ideas of using Hexo (<https://hexo.io>) are very similar to what we have applied to Jekyll in the previous section. I have also prepared a minimal example in the GitHub repository [yihui/blogdown-hexo](https://github.com/yihui/blogdown-hexo).<sup>3</sup>

The key components of this repository are still `.Rprofile`, `R/build.R`, and `R/build_one.R`. We set the option `blogdown.generator` to `hexo`, the `build.method` to `custom`, and the default subdirectory for new posts to `source/_posts`.

```
options(  
  blogdown.generator = 'hexo',  
  blogdown.method = 'custom',  
  blogdown.subdir = 'source/_posts'  
)
```

The script `R/build.R` is similar to the one in the `blogdown-jekyll` repository. The main differences are:

1. We find all Rmd files under the `source/` directory instead of the root directory, because Hexo's convention is to put all source files under `source/`.
2. We call `system2('hexo', 'generate')` to build the website.

For the script `R/build_one.R`, the major difference with the script in the `blogdown-jekyll` repository is that we set the `base.dir` option for

---

<sup>3</sup><https://github.com/yihui/blogdown-hexo>

**knitr**, so that all R figures are generated to the `source/` directory. This is because Hexo copies everything under `source/` to `public/`, whereas Jekyll copies everything under the root directory to `_site/`.

```
local({
  # fall back on '/' if baseurl is not specified
  baseurl = blogdown::get_config2('root', '/')
  knitr::opts_knit$set(
    base.url = baseurl, base.dir = normalizePath('source')
  )

  # input/output filenames as two arguments to Rscript
  a = commandArgs(TRUE)
  d = gsub('^source/_?|[.][a-zA-Z]+$', '', a[1])
  knitr::opts_chunk$set(
    fig.path = sprintf('figure/%s/', d),
    cache.path = sprintf('cache/%s/', d)
  )
  knitr::knit(
    a[1], a[2], quiet = TRUE, encoding = 'UTF-8', envir = .GlobalEnv
  )
})
```

This repository is also automatically built and deployed through Netlify when I push changes to it. Since Hexo is a Node package, and Netlify supports Node, you can easily install Hexo on Netlify. For example, this example repository uses the command `npm install && hexo generate` to build the website; `npm install` will install the Node packages specified in `packages.json` (a file under the root directory of the repository), and `hexo generate` is the command to build the website from `source/` to `public/`.

---

### 5.3 Default site generator in **rmarkdown**

Before **blogdown** was invented, there was actually a relatively simple way to render websites using **rmarkdown**. The structure of the website has to be a flat directory of Rmd files (no subdirectories for Rmd files) and a configuration file in which you can specify a navigation bar for all your pages and output format options.

You can find more information about this site generator in its documentation at [http://rmarkdown.rstudio.com/rmarkdown\\_websites.html](http://rmarkdown.rstudio.com/rmarkdown_websites.html), and we are not going to repeat the documentation here, but just want to highlight the major differences between the default site generator in **rmarkdown** and other specialized site generators like Hugo:

- The **rmarkdown** site generator requires all Rmd files to be under the root directory. Hugo has no constraints on the site structure, and you can create arbitrary directories and files under `/content/`.
- Hugo is a general-purpose site generator that is highly customizable, and there are a lot of things that **rmarkdown**'s default site generator does not support, e.g., RSS feeds, metadata especially common in blogs such as categories and tags, and customizing permanent links for certain pages.

There are still legitimate reasons to choose the **rmarkdown** default site generator, even though it does not appear to be as powerful as Hugo, including:

- You are familiar with generating single-page HTML output from R Markdown, and all you want is to extend this to generating multiple pages from multiple Rmd files.
- It suffices to use a flat directory of Rmd files. You do not write a blog or need RSS feeds.



- You prefer the Bootstrap styles. In theory, you can also apply Bootstrap styles to Hugo websites, but it will require you to learn more about Hugo. Bootstrap is well supported in **rmarkdown**, and you can spend more time on the configurations instead of learning the technical details about how it works.
- There are certain features in **rmarkdown** HTML output that are missing in **blogdown**. For example, currently you cannot easily print data frames as paged tables, add a floating table of contents, or fold/unfold code blocks dynamically in the output of **blogdown**. All these could be implemented via JavaScript and CSS, but it is certainly not as simple as specifying a few options in **rmarkdown** like `toc_float: true`.

Please note that the **rmarkdown** site generator is extensible, too. For example, the **bookdown** package (Xie, 2018a) is essentially a custom site generator to generate books as websites.

---

## 5.4 **pkgdown**

The **pkgdown** package (?, <https://github.com/hadley/pkgdown>) can help you quickly turn the R documentation of an R package (including help pages and vignettes) into a website. It is independent of **blogdown** and solves a specific problem. It is not a general-purpose website generator. We want to mention it in this book because it is very easy to use, and also highly useful. You can find the instructions on its website or in its GitHub repository.



# A

## *R Markdown*

R Markdown (Allaire et al., 2018) is a plain-text document format consisting of two components: R (or other computing languages) and Markdown. Markdown makes it easy for authors to write a document due to its simple syntax. Program code (such as R code) can be embedded in a source Markdown document to generate an output document directly: when compiling the source document, the program code will be executed and its output will be intermingled with the Markdown text.

R Markdown files usually use the filename extension `.Rmd`. Below is a minimal example:

```
---
title: A Simple Linear Regression
author: Yihui Xie
---

We build a linear regression below.

```{r}
fit = lm(dist ~ speed, data = cars)
b = coef(summary(fit))
plot(fit)
```

The slope of the regression is `r b[2, 1]`.
```

Such a document can be compiled using the function `rmarkdown::render()`, or equivalently, by clicking the `knit` button in RStudio. Under the hood, an R Markdown document is first compiled

to Markdown through **knitr** (Xie, 2018b), which executes all program code in the document. Then the Markdown output document is compiled to the final output document through Pandoc, such as an HTML page, a PDF document, a Word document, and so on. It is important to know this two-step process, otherwise you may not know which package documentation to look up when you have questions. Basically, for anything related to the (R) code chunks, consult the **knitr** documentation (<https://yihui.name/knitr/>); for anything related to Markdown, consult the Pandoc documentation (<https://pandoc.org>).

An R Markdown document typically consists of YAML metadata (optional) and the document body. YAML metadata are written between a pair of `---` to set some attributes of the document, such as the title, author, and date, etc. In the document body, you can mix code chunks and narratives. A code block starts with a chunk header ```{r}` and ends with ````. There are many possible chunk options that you can set in the chunk header to control the output, e.g., you can set the figure height to 4 inches using ```{r fig.height=4}`. For all possible chunk options, see <https://yihui.name/knitr/options/>.

Pandoc supports a large variety of output document formats. For **blogdown**, the output format is set to HTML (`blogdown::html_page`), since a website typically consists of HTML pages. If you want other formats, please see Section ???. To create an R Markdown post for **blogdown**, it is recommended that you use the RStudio “New Post” (Figure 1.2) or the function `blogdown::new_post()`, instead of the RStudio menu `File -> New File -> R Markdown`.

You are strongly recommended to go through the documentation of **knitr** chunk options and Pandoc’s manual at least once to have an idea of all possibilities. The basics of Markdown are simple enough, but there are many less well-known features in Pandoc’s Markdown, too. As we mentioned in Section 1.5, **blogdown**’s output format is based on **bookdown** (Xie, 2018a), which contains several other Markdown extensions, such as numbered equations

and theorem environments, and you need to read Chapter 2 of the **bookdown** book (Xie, 2016) to learn more about these features.

You can find an R Markdown cheat sheet and a reference guide at <https://www.rstudio.com/resources/cheatsheets/>, which can be handy after you are more familiar with R Markdown.

With R Markdown, you only need to maintain the source documents; all output pages can be automatically generated from source documents. This makes it much easier to maintain a website, especially when the website is related to data analysis or statistical computing and graphics. When the source code is updated (e.g., the model or data is changed), your web pages can be updated accordingly and automatically. There is no need to run the code separately and cut-and-paste again. Besides the convenience, you gain reproducibility at the same time.



# B

## *Website Basics*

If you want to tweak the appearance of your website, or even design your own theme, you must have some basic knowledge of web development. In this appendix, we briefly introduce HTML, CSS, and JavaScript, which are the most common components of a web page, although CSS and JavaScript are optional.

We only aim at getting you started with HTML, CSS, and JavaScript. HTML is relatively simple to learn, but CSS and JavaScript can be much more complicated, depending on how much you want to learn and what you want to do with them. After reading this appendix, you will have to use other resources to teach yourself. When you search for these technologies online, the most likely websites that you may hit are MDN<sup>1</sup> (Mozilla Developer Network), w3schools.com,<sup>2</sup> and StackOverflow.<sup>3</sup> Among these websites, w3schools often provides simple examples and tutorials that may be friendlier to beginners, but we often hear negative comments<sup>4</sup> about it, so please use it with caution. I often read all three websites when looking for solutions.

If we were only allowed to give one single most useful tip about web development, it would be: use the Developer Tools of your web browser. Most modern web browsers provide these tools. For example, you can find these tools from the menu of Google Chrome View -> Developer, Firefox's menu Tools -> Web Developer, or Safari's menu Develop -> Show Web Inspector. Figure B.1 is a screenshot of using the Developer Tools in Chrome.

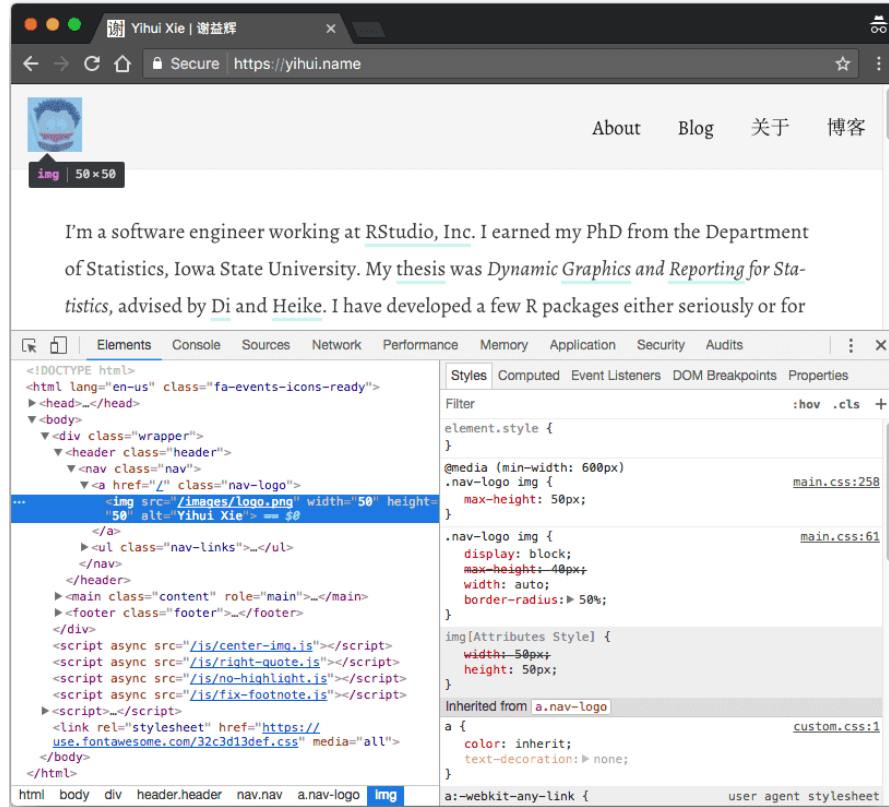
---

<sup>1</sup><https://developer.mozilla.org>

<sup>2</sup><https://www.w3schools.com>

<sup>3</sup><https://stackoverflow.com>

<sup>4</sup><https://meta.stackoverflow.com/q/280478/559676>



**FIGURE B.1:** Developer Tools in Google Chrome.

Typically you can also open the Developer Tools by right-clicking on a certain element on the web page and selecting the menu item Inspect (or Inspect Element). In Figure B.1, I right-clicked on the profile image of my website <https://yihui.name> and inspected it, and Chrome highlighted its HTML source code `` in the left pane. You can also see the CSS styles associated with this `img` element in the right pane. What is more, you can interactively change the styles there if you know CSS, and see the (temporary) effects in real time on the page! After you are satisfied with the new styles, you can write the CSS code in files.

There are a lot of amazing features of Developer Tools, which make them not only extremely useful for debugging and experimentation, but also helpful for learning web development. These tools



are indispensable to me when I develop anything related to web pages. I learned a great deal about CSS and JavaScript by playing with these tools.

---

## B.1 HTML

HTML stands for Hyper Text Markup Language, and it is the language behind most web pages you see. You can use the menu `View -> View Source` or the context menu `View Page Source` to see the full HTML source of a web page in your browser. All elements on a page are represented by HTML tags. For example, the tag `<p>` represents paragraphs, and `<img>` represents images.

The good thing about HTML is that the language has only a limited number of tags, and the number is not very big (especially the number of commonly used tags). This means there is hope that you can master this language fully and quickly.

Most HTML tags appear in pairs, with an opening tag and a closing tag, e.g., `<p></p>`. You write the content between the opening and closing tags, e.g., `<p>This is a paragraph.</p>`. There are a few exceptions, such as the `<img>` tag, which can be closed by a slash `/` in the opening tag, e.g., ``. You can specify attributes of an element in the opening tag using the syntax `name=value` (a few attributes do not require `value`).

HTML documents often have the filename extension `.html` (or `.htm`). Below is an overall structure of an HTML document:

```
<html>

  <head>
  </head>

  <body>
```

```
</body>

</html>
```

Basically an HTML document consists a `head` section and `body` section. You can specify the metadata and include assets like CSS files in the `head` section. Normally the `head` section is not visible on a web page. It is the `body` section that holds the content to be displayed to a reader. Below is a slightly richer example document:

```
<!DOCTYPE html>
<html>

  <head>
    <meta charset="utf-8" />

    <title>Your Page Title</title>

    <link rel="stylesheet" href="/css/style.css" />
  </head>

  <body>
    <h1>A First-level Heading</h1>

    <p>A paragraph.</p>

    <ul>
      <li>An item.</li>
      <li>Another item.</li>
      <li>Yet another item.</li>
    </ul>

    <script src="/js/bar.js"></script>
```

```
</body>
```

```
</html>
```

In the head, we declare the character encoding of this page to be UTF-8 via a `<meta>` tag, specify the title via the `<title>` tag, and include a stylesheet via a `<link>` tag.

The body contains a first-level section heading `<h1>`,<sup>5</sup> a paragraph `<p>`, an image `<img>`, an unordered list `<ul>` with three list items `<li>`, and includes a JavaScript file in the end via `<script>`.

There are much better tutorials on HTML than this section, such as those offered by MDN and w3schools.com, so we are not going to make this section a full tutorial. Instead, we just want to provide a few tips on HTML:

- You may validate your HTML code via this service: <https://validator.w3.org>. This validator will point out potential problems of your HTML code. It actually works for XML and SVG documents, too.
- Among all HTML attributes, file paths (the `src` attribute of some tags like `<img>`) and links (the `href` attribute of the `<a>` tag) may be the most confusing to beginners. Paths and links can be relative or absolute, and may come with or without the protocol and domain. You have to understand what a link or path exactly points to. A full link is of the form `http://www.example.com/foo/bar.ext`, where `http` specifies the protocol (it can be other protocols like `https` or `ftp`), `www.example.com` is the domain, and `foo/bar.ext` is the file under the root directory of the website.
  - If you refer to resources on the same website (the same protocol and domain), we recommend that you omit the protocol and domain names, so that the links will continue to work even if you change

---

<sup>5</sup>There are six possible levels from `h1`, `h2`, ..., to `h6`.

the protocol or domain. For example, a link `<a href="/hi/there.html">` on a page `http://example.com/foo/` refers to `http://example.com/hi/there.html`. It does not matter if you change `http` to `https`, or `example.com` to `another-domain.com`.

- Within the same website, a link or path can be relative or absolute. The meaning of an absolute path does not change no matter where the current HTML file is placed, but the meaning of a relative path depends on the location of the current HTML file. Suppose you are currently viewing the page `example.com/hi/there.html`:
  - \* A absolute path `/foo/bar.ext` always means `example.com/foo/bar.ext`. The leading slash means the root directory of the website.
  - \* A relative path `../images/foo.png` means `example.com/images/foo.png` (`..` means to go one level up). However, if the HTML file `there.html` is moved to `example.com/hey/again/there.html`, this path in `there.html` will refer to `example.com/hey/images/foo.png`.
  - \* When deciding whether to use relative or absolute paths, here is the rule of thumb: if you will not move the resources referred or linked to from one subpath to another (e.g., from `example.com/foo/` to `example.com/bar/`), but only move the HTML pages that use these resources, use absolute paths; if you want to change the subpath of the URL of your website, but the relative locations of HTML files and the resources they use do not change, you may use relative links (e.g., you can move the whole website from `example.com/` to `example.com/foo/`).
  - \* If the above concepts sound too complicated, a better way is to either think ahead carefully about the structure of your website and avoid moving files, or use rules of redirects if supported (such as 301 or 302 redirects).
- If you link to a different website or web page, you have to include the domain in the link, but it may not be neces-

sary to include the protocol, e.g., `//test.example.com/foo.css` is a valid path. The actual protocol of this path matches the protocol of the current page, e.g., if the current page is `https://example.com/`, this link means `https://test.example.com/foo.css`. It may be beneficial to omit the protocol because HTTP resources cannot be embedded on pages served through HTTPS (for security reasons), e.g., an image at `http://example.com/foo.png` cannot be embedded on a page `https://example.com/hi.html` via ``, but `` will work if the image can be accessed via HTTPS, i.e., `https://example.com/foo.png`. The main drawback of not including the protocol is that such links and paths do not work if you open the HTML file locally without using a web server, e.g., only double-click the HTML file in your file browser and show it in the browser.<sup>6</sup>

- A very common mistake that people make is a link without the leading double slashes before the domain. You may think `www.example.com` is a valid link. It is not! At least it does not link to the website that you intend to link to. It works when you type it in the address bar of your browser because your browser will normally autocomplete it to `http://www.example.com`. However, if you write a link `<a href="www.example.com">See this link</a>`, you will be in trouble. The browser will interpret this as a relative link, and it is relative to the URL of the current web page, e.g., if you are currently viewing `http://yihui.name/cn/`, the link `www.example.com` actually means `http://yihui.name/cn/www.example.com`! Now you should know the Markdown text `[Link](www.example.com)` is typically a mistake, unless you really mean to link to a sub-

---

<sup>6</sup>That is because without a web server, an HTML file is viewed via the protocol `file`. For example, you may see a URL of the form `file://path/to/the/file.html` in the address bar of your browser. The path `//example.com/foo.png` will be interpreted as `file://example.com/foo.png`, which is unlikely to exist as a local file on your computer.

directory of the current page or a file with literally the name `www.example.com`.

---

## B.2 CSS

The Cascading Stylesheets (CSS) language is used to describe the look and formatting of documents written in HTML. CSS is responsible for the visual style of your site. CSS is a lot of fun to play with, but it can also easily steal your time.

In the Hugo framework (<https://gohugo.io/tutorials/creating-a-new-theme/>), CSS is one of the major “non-content” files that shapes the look and feel of your site (the others are images, JavaScript, and Hugo templates). What does the “look and feel”<sup>7</sup> of a site mean? “Look” generally refers to static style components including, but not limited to

- color palettes,
- images,
- layouts/margins, and
- fonts.

whereas “feel” relates to dynamic components that the user interacts with like

- dropdown menus,
- buttons, and
- forms.

There are 3 ways to define styles. The first is in line with HTML. For example, this code

---

<sup>7</sup>[https://en.wikipedia.org/wiki/Look\\_and\\_feel](https://en.wikipedia.org/wiki/Look_and_feel)

```
<p>Marco! <em>Polo!</em></p>
```

would produce text that looks like this: Marco! *Polo!*

However, this method is generally not preferred for numerous reasons.<sup>8</sup>

A second way is to internally define the CSS by placing a style section in your HTML:

```
<html>
<style>
#favorite {
    font-style: italic;
}
</style>
<ul id="tea-list">
    <li>Earl Grey</li>
    <li>Darjeeling</li>
    <li>Oolong</li>
    <li>Chamomile</li>
    <li id="favorite">Chai</li>
</ul>
</html>
```

In this example, only the last tea listed, *Chai*, is italicized.

The third method is the most popular because it is more flexible and the least repetitive. In this method, you define the CSS in an external file that is then referenced as a link in your HTML:

```
<html>
    <link rel="stylesheet" href="/css/style.css" />
</html>
```

---

<sup>8</sup><https://stackoverflow.com/q/12013532/559676>

What goes inside the linked CSS document is essentially a list of rules (the same list could appear internally between the style tags, if you are using the second method). Each rule must include both a selector or group of selectors, and a declarations block within curly braces that contains one or more `property: value;` pairs. Here is the general structure for a rule<sup>9</sup>:

```
/* CSS pseudo-code */
selectorlist {
    property: value;
    /* more property: value; pairs*/
}
```

Selectors<sup>10</sup> can be based on HTML element types or attributes, such as `id` or `class` (and combinations of these attributes):

```
/* by element type */
li {
    color: yellow; /* all <li> elements are yellow */
}

/* by ID with the # symbol */
#my-id {
    color: yellow; /* elements with id = "my-id" are yellow */
}

/* by class with the . symbol */
.my-class {
    color: yellow; /* elements with class = "my-class" are yellow */
}
```

Because each HTML element may match several different selectors,

---

<sup>9</sup><https://developer.mozilla.org/en-US/docs/Web/CSS/Reference>

<sup>10</sup><https://developer.mozilla.org/en-US/docs/Web/CSS/Reference#Selectors>



the CSS standard determines which set of rules has precedence for any given element, and which properties to inherit. This is where the cascade algorithm comes into play. For example, take a simple unordered list:

```
<ul id="tea-list">
  <li>Earl Grey</li>
  <li>Darjeeling</li>
  <li>Oolong</li>
  <li>Chamomile</li>
  <li>Chai</li>
</ul>
```

Now, let's say we want to highlight our favorite teas again, so we'll use a class attribute.

```
<ul id="tea-list">
  <li>Earl Grey</li>
  <li class="favorite">Darjeeling</li>
  <li>Oolong</li>
  <li>Chamomile</li>
  <li class="favorite">Chai</li>
</ul>
```

We can use this class attribute as a selector in our CSS. Let's say we want our favorite teas to be in bold and have a background color of yellow, so our CSS would look like this:

```
.favorite {
  font-weight: bold;
  background-color: yellow;
}
```

Now, if you want every list item to be italicized with a white background, you can set up another rule:

```
li {  
    font-style: italic;  
    background-color: white;  
}
```

If you play with this code (which you can do easily using sites like <http://jsbin.com>, <https://jsfiddle.net>, or <https://codepen.io/pen/>), you'll see that the two favorite teas are still highlighted in yellow. This is because the CSS rule about `.favorite` as a class is more specific than the rule about `li` type elements. To override the `.favorite` rule, you need to be as specific as you can be when choosing your group of selectors:

```
ul#tea-list li.favorite {  
    background-color: white;  
}
```

This example just scratches the surface of cascade and inheritance.<sup>11</sup>

For any Hugo theme that you install, you can find the CSS file in the `themes/` folder. For example, the default lithium theme is located in: `themes/hugo-lithium-theme/static/css/main.css`. Once you are familiar with CSS, you can understand how each set of rules work to shape the visual style of your website, and how to alter the rules. For some themes (i.e., the hugo-academic theme<sup>12</sup>), you have the option of linking to a custom CSS,<sup>13</sup> which you can use to further customize the visual style of your site.

---

<sup>11</sup>[https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction\\_to\\_CSS/Cascade\\_and\\_inheritance](https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/Cascade_and_inheritance)

<sup>12</sup><https://github.com/gcushen/hugo-academic>

<sup>13</sup><https://gist.github.com/gcushen/d5525a4506b9ccf83f2bce592a895495>

A few one-line examples illustrate how simple CSS rules can be used to make dramatic changes:

- To make circular or rounded images, you may assign a class `img-circle` to images (e.g., ``) and define the CSS:

```
.img-circle {  
  border-radius: 50%;  
}
```

- To make striped tables, you can add background colors to odd or even rows of the table, e.g.,

```
tr:nth-child(even) {  
  background: #eee;  
}
```

- You can append or prepend content to elements via pseudo-elements `::after` and `::before`. Here is an example of adding a period after section numbers: <https://github.com/rstudio/blogdown/issues/80>.

---

## B.3 JavaScript

It is way more challenging to briefly introduce JavaScript than HTML and CSS, since it is a programming language. There are many books and tutorials about this language. Anyway, we will try to scratch the surface for R users in this section.

In a nutshell, JavaScript is a language that is typically used to manipulate elements on a web page. An effective way to learn it

is through the JavaScript console in the Developer Tools of your web browser (see Figure B.1), because you can interactively type code in the console and execute it, which feels similar to executing R code in the R console (e.g., in RStudio). You may open any web page in your web browser (e.g., <https://yihui.name>), then open the JavaScript console, and try the code below on any web page:

```
document.body.style.background = 'orange';
```

It should change the background color of the page to orange, unless the page has already defined background colors for certain elements.

To effectively use JavaScript, you have to learn both the basic syntax of JavaScript and how to select elements on a page before you can manipulate them. You may partially learn the former from the short JavaScript snippet below:

```
var x = 1; // assignments
1 + 2 - 3 * 4 / 5; // arithmetic

if (x < 2) console.log(x); // "print" x

var y = [9, 1, 0, 2, 1, 4]; // array

// function
var sum = function(x) {
  var s = 0;
  // a naive way to compute the sum
  for (var i=0; i < x.length; i++) {
    s += x[i];
  }
  return s;
};
```

```
sum(y);

var y = "Hello World";
y = y.replace(" ", ", "); // string manipulation
```

You may feel the syntax is similar to R to some degree. JavaScript is an object-oriented language, and usually there are several methods that you can apply to an object. The string manipulation above is a typical example of the `Object.method()` syntax. To know the possible methods on an object, you can type the object name in your JavaScript console followed by a dot, and you should see all candidates.

R users have to be extremely cautious because JavaScript objects are often mutable, meaning that an object could be modified anywhere. Below is a quick example:

```
var x = {"a": 1, "b": 2}; // like a list in R
var f = function(z) {
  z.a = 100;
};
f(x);
x; // modified! x.a is 100 now
```

There are many mature JavaScript libraries that can help you select and manipulate elements on a page, and the most popular one may be jQuery.<sup>14</sup> However, you should know that sometimes you can probably do well enough without these third-party libraries. There are some basic methods to select elements, such as `document.getElementById()` and `document.getElementsByClassName()`. For example, you can select all paragraphs using `document.querySelectorAll('p')`.

Next we show a slightly advanced application, in which you will see

---

<sup>14</sup><https://jquery.com>

anonymous functions, selection of elements by HTML tag names, regular expressions, and manipulation of HTML elements.

In Section 2.5.2, we mentioned how to enable MathJax on a Hugo website. The easy part is to include the script `MathJax.js` via a `<script>` tag, and there are two hard parts:

1. How to protect the math content from the Markdown engine (Blackfriday), e.g., we need to make sure underscores in math expressions are not interpreted as `<em></em>`. This problem only exists in plain Markdown posts, and has been mentioned in Section 1.5 without explaining the solution.
2. By default, MathJax does not recognize a pair of single dollar signs as the syntax for inline math expressions, but most users are perhaps more comfortable with the syntax `$x$` than `\(x\)`.

The easiest solution to the first problem may be adding backticks around math expressions, e.g., ``$x_i$``, but the consequence is that the math expression will be rendered in `<code></code>`, and MathJax ignores `<code>` tags when looking for math expressions on the page. We can force MathJax to search for math expressions in `<code>`, but this will still be problematic. For example, someone may want to display inline R code ``list$x$y``, and `$x$` may be recognized as a math expression. MathJax ignores `<code>` for good reasons. Even if you do not have such expressions in `<code>`, you may have some special CSS styles attached to `<code>`, and these styles will be applied to your math expressions, which can be undesired (e.g., a light gray background).

To solve these problems, I have provided a solution in the JavaScript code at <https://yihui.name/js/math-code.js>:

```
(function() {  
  var i, text, code, codes = document.getElementsByTagName('code');  
  for (i = 0; i < codes.length; i) {
```

```

code = codes[i];
if (code.parentNode.tagName !== 'PRE' &&
    code.childElementCount === 0) {
    text = code.textContent;
    if (/^\$[\^$]/.test(text) && /^[^\$]\$\$/ .test(text)) {
        text = text.replace(/^\$/ , '\\(').replace(/\$\$/ , '\\)');
        code.textContent = text;
    }
    if (/^\\((.|\s)+\\)\$/ .test(text) ||
        /^\\[(.|\s)+\\]\$/ .test(text) ||
        /^\\$(.|\s)+\$\$/ .test(text) ||
        /^\\begin\{([^\}]+\)\}(.|\s)+\\end\{([^\}]+\)\}\$/ .test(text)) {
        code.outerHTML = code.innerHTML; // remove <code></code>
        continue;
    }
}
i++;
}
})();

```

It is not a perfect solution, but it should be very rare that you run into problems. This solution identifies possible math expressions in `<code>`, and strips the `<code>` tag, e.g., replaces `<code>$x$</code>` with `\(x\)`. After this script is executed, we load the MathJax script. This way, we do not need to force MathJax to search for math expressions in `<code>` tags, and your math expressions will not inherit any styles from `<code>`. The JavaScript code above is not too long, and should be self-explanatory. The trickiest part is `i++`. I will leave it to readers to figure out why the `for` loop is not the usual form `for (i = 0; i < codes.length; i++)`. It took me quite a few minutes to realize my mistake when I wrote the loop in the usual form.

## B.4 Useful resources

### B.4.1 File optimization

Although static websites are fast in general, you can certainly further optimize them. You may search for “CSS and JavaScript minifier,” and these tools can compress your CSS and JavaScript files, so that they can be loaded faster. Since there are a lot of tools, I will not recommend any here.

You can also optimize images on your website. I frequently use a command-line tool named `optipng`<sup>15</sup> to optimize PNG images. It is a lossless optimizer, meaning that it reduces the file size of a PNG image without loss of quality. From my experience, it works very well on PNG images generated from R, and can reduce the file size by at least 30% (sometimes even more than 50%). Personally I also use online tools like <http://optimizilla.com> to optimize PNG and JPEG images. For GIF images, I often use <https://ezgif.com/optimize> to reduce the file sizes if they are too big.

Note that Netlify has provided the optimization features on the server side for free at the moment, so you may just want to enable them there instead of doing all the hard work by yourself.

### B.4.2 Helping people find your site

Once your site is up and running, you probably want people to find it. SEO — Search Engine Optimization — is the art of making a website easy for search engines like Google to understand. And, hopefully, if the search engine knows what you are writing about, it will present links to your site high in results when someone searches for topics you cover.

Entire books have been written about SEO, not to mention the many companies that are in the business of offering (paid) technical and strategic advice to help get sites atop search-engine rank-

---

<sup>15</sup><http://optipng.sourceforge.net>



ings. If you are interested in finding out more, a good place to start is the Google Search Engine Optimization Starter Guide (<http://bit.ly/google-seo-starter>). Below are a few key points:

1. The title that you select for each page and post is a very important signal to Google and other search engines telling them what that page is about.
2. Description tags are also critical to explain what a page is about. In HTML documents, description tags<sup>16</sup> are one way to provide metadata about the page. Using **blog-down**, the description may end up as text under the page title in a search-engine result. If your page's YAML does not include a description, you can add one like `description: "A brief description of this page."`; the HTML source of the rendered page would have a `<meta>` tag in `<head>` like `<meta name="description" content="A brief description of this page.">`. Not all themes support adding the description to your HTML page (although they should!)
3. URL structure also matters. You want your post's slug to have informative keywords, which gives another signal of what the page is about. Have a post with interesting things to do in San Francisco? `san-francisco-events-calendar` might be a better slug than `my-guide-to-fun-things-to-do`.

---

<sup>16</sup>[https://www.w3schools.com/tags/tag\\_meta.asp](https://www.w3schools.com/tags/tag_meta.asp)



# C

---

## *Domain Name*

---

While you can use the free subdomain names like those provided by GitHub or Netlify, it may be a better idea to own a domain name of your own. The cost of an apex domain is minimal (typically the yearly cost is about US\$10), and you will enter a much richer world after you purchase a domain name. For example, you are free to point your domain to any web servers, you can create as many subdomain names as you want, and you can even set up your own email accounts using the domain or subdomains. In this appendix, we will explain some basic concepts of domain names, and mention a few (free) services to help you configure your domain name.

Before we dive into the details, we want to outline the big picture of how a URL works in your web browser. Suppose you typed or clicked a link `http://www.example.com/foo/index.html` in your web browser. What happens behind the scenes before you see the actual web page?

First, the domain name has to be resolved through the nameservers associated with it. A nameserver knows the DNS (Domain Name System) records of a domain. Typically it will look up the “A records” to point the domain to the IP address of a web server. There are several other types of DNS records, and we will explain them later. Once the web server is reached, the server will look for the file `foo/index.html` under a directory associated with the domain name, and return its content in the response. That is basically how you can see a web page.

---

## C.1 Registration

You can purchase a domain name from many domain name registrars. To stay neutral, we are not going to make recommendations here. You can use your search engine to find a registrar by yourself, or ask your friends for recommendations. However, we would like to remind you of a few things that you should pay attention to when looking for a domain name registrar:

- You should have the freedom to transfer your domain from the current registrar to other registrars, i.e., they should not lock you in their system. To transfer a domain name, you should be given a code known as the “Transfer Auth Code” or “Auth Code” or “Transfer Key” or something like that.
- You should be able to customize the nameservers (see Section [C.2](#)) of your domain. By default, each registrar will assign their own nameservers to you, and these nameservers typically work very well. However, there are some special nameservers that provide services more than just DNS records, and you may be interested in using them.
- Other people can freely look up your personal information, such as your email or postal address, after you register a domain and submit this information to the registrar. This is called the “WHOIS Lookup.” You may want to protect your privacy, but your registrar may require an extra payment.

---

## C.2 Nameservers

The main reason why we need nameservers is that we want to use domains instead of IP addresses, although a domain is not strictly necessary for you to be able to access a website. You could use the

IP address if you have your own server with a public IP, but there are many problems with this approach. For example, IP addresses are limited (in particular IPv4), not easy to memorize, and you can only host one website per IP address (without using other ports).

A nameserver is an engine that directs the DNS records of your domain. The most common DNS record is the A record, which maps a domain to an IP address, so that the hosting server can be found via its IP address when a website is accessed through a domain. We will introduce two more types of DNS records in Section C.3: CNAME and MX records.

In most cases, the default nameservers provided by your domain registrar should suffice, but there is a special technology missing in most nameservers: CNAME flattening. You only need this technology if you want to set a CNAME record for your apex domain. The only use case to my knowledge is when you host your website via Netlify, but want to use the apex domain instead of the `www` subdomain, e.g., you want to use `example.com` instead of `www.example.com`. To make use of this technology, you could consider Cloudflare,<sup>1</sup> which provides this DNS feature for free. Basically, all you need to do is to point the nameservers of your domain to the nameservers provided by Cloudflare (of the form `*.ns.cloudflare.com`).

---

### C.3 DNS records

There are many types of DNS records, and you may see a full list on Wikipedia.<sup>2</sup> The most commonly used types may be A, CNAME, and MX records. Figure C.1 shows a subset of DNS records of my domain `yihui.name` on Cloudflare, which may give you an idea of what DNS records look like. You may query DNS records using

---

<sup>1</sup><https://www.cloudflare.com>

<sup>2</sup>[https://en.wikipedia.org/wiki/List\\_of\\_DNS\\_record\\_types](https://en.wikipedia.org/wiki/List_of_DNS_record_types)

command-line tools such as `dig`<sup>3</sup> or an app provided by Google: <https://toolbox.googleapps.com/apps/dig/>.

DNS Records

A, AAAA, and CNAME records can have their traffic routed through the Cloudflare system. Add more records using this form, and click the cloud next to each record to toggle Cloudflare on or off.

A



Automatic TTL

Add Record

Type	Name	Value	TTL	Status
CNAME	db	is an alias of updog.co	Automatic	
CNAME	slides	is an alias of updog.co	Automatic	
CNAME	t	is an alias of twitter-yihui.netlify.com	Automatic	
CNAME	www	is an alias of yihui.netlify.com	Automatic	
CNAME	xran	is an alias of yihui.github.io	Automatic	
CNAME	yihui.name	is an alias of yihui.netlify.com	Automatic	
MX	yihui.name	mail handled by alt1.aspmx.l.google.com (5)	Automatic	
MX	yihui.name	mail handled by alt2.aspmx.l.google.com (5)	Automatic	
MX	yihui.name	mail handled by aspmx2.googlemail.com (10)	Automatic	
MX	yihui.name	mail handled by aspmx3.googlemail.com (10)	Automatic	
MX	yihui.name	mail handled by aspmx.l.google.com (1)	Automatic	

**FIGURE C.1:** Some DNS records of the domain yihui.name on Cloudflare.

An apex domain can have any number of subdomains. You can set DNS records for the apex domain and any subdomains. You can see from Figure C.1 that I have several subdomains, e.g., `slides.yihui.name` and `xran.yihui.name`.

As we have mentioned, an A record points a domain or subdomain to an IP address of the host server. I did not use any A records for my domains since all services I use, such as Updog, GitHub Pages, and Netlify, support CNAME records well. A CNAME

<sup>3</sup>[https://en.wikipedia.org/wiki/Dig\\_\(command\)](https://en.wikipedia.org/wiki/Dig_(command))

record is an alias, pointing one domain to another domain. The advantage of using CNAME over A is that you do not have to tie a domain to a fixed IP address. For example, the CNAME record for `t.yihui.name` is `twitter-yihui.netlify.com`. The latter domain is provided by Netlify, and I do not need to know where they actually host the website. They are free to move the host of `twitter-yihui.netlify.com`, and I will not need to update my DNS record. Every time someone visits the website `t.yihui.name`, the web browser will route the traffic to the domain set in the CNAME record. Note that this is different from redirection, i.e., the URL `t.yihui.name` will not be explicitly redirected to `twitter-yihui.netlify.com` (you still see the former in the address bar of your browser).

Normally, you can set any DNS records for the apex domain except CNAME, but I set a CNAME record for my apex domain `yihui.name`, and that is because Cloudflare supports CNAME flattening. For more information on this topic, you may read the post “To WWW or not WWW,”<sup>4</sup> by Netlify. Personally, I prefer not using the subdomain `www.yihui.name` to keep my URLs short, so I set a CNAME record for both the apex domain `yihui.name` and the `www` subdomain, and Netlify will automatically redirect the `www` subdomain to the apex domain. That said, if you are a beginner, it may be a little easier to configure and use the `www` subdomain, as suggested by Netlify. Note `www` is a conventional subdomain that sounds like an apex domain, but really is not; you can follow this convention or not as you wish.

For email services, I was an early enough “netizen”,<sup>5</sup> and when I registered my domain name, Google was still offering free email services to custom domain owners. That is how I can have a custom mailbox `xie@yihui.name`. Now you will have to pay for G Suite.<sup>6</sup> In Figure C.1 you can see I have set some MX (stands for “mail exchange”) records that point to some Google mail servers. Of course, Google is not the only possible choice when it comes to cus-

---

<sup>4</sup><https://www.netlify.com/blog/2017/02/28/to-www-or-not-www/>

<sup>5</sup><https://en.wikipedia.org/wiki/Netizen>

<sup>6</sup><https://gsuite.google.com>

tom mailboxes. Migadu<sup>7</sup> claims to be the “most affordable email hosting.” You may try its free plan and see if you like it. Unless you are going to use your custom mailbox extensively and for professional purposes, the free plan may suffice. In fact, you may create an alias address on Migadu to forward emails to your other email accounts (such as Gmail) if you do not care about an actual custom mailbox. Migadu has provided detailed instructions on how to set the MX records for your domain.

---

<sup>7</sup><https://www.migadu.com>



# D

## *Advanced Topics*

In this appendix, we talk about a few advanced topics that may be of interest to developers and advanced users.

### D.1 More global options

There are a few more advanced global options in addition to those introduced in Section ??, and they are listed in Table D.1.

If you want to install Hugo to a custom path, you can set the global option `blogdown.hugo.dir` to a directory to store the Hugo executable before you call `install_hugo()`, e.g., `options(blogdown.hugo.dir = '~/Downloads/hugo_0.20.1/')`. This may be useful for you to use a specific version of Hugo for a specific website,<sup>1</sup> or store a copy of Hugo on a USB Flash drive along with your website.

The option `blogdown.method` is explained in Section D.9.

<sup>1</sup>You can set this option per project. See Section ?? for details.

**TABLE D.1:** A few more advanced global options.

Option name	Default	Meaning
<code>blogdown.hugo.dir</code>		The directory of the Hugo executable
<code>blogdown.method</code>	html	The building method for R Markdown
<code>blogdown.publishDir</code>		The publish dir for local preview
<code>blogdown.widgetsID</code>	TRUE	Incremental IDs for HTML widgets?

When your website project is under version control in the RStudio IDE, continuously previewing the site can be slow, if it contains hundreds of files or more. The default publish directory is `public/` under the project root directory, and whenever you make a change in the source that triggers a rebuild, RStudio will be busy tracking file changes in the `public/` directory. The delay before you see the website in the RStudio Viewer can be 10 seconds or even longer. That is why we provide the option `blogdown.publishDir`. You may set a temporary publish directory to generate the website, and this directory should not be under the same RStudio project, e.g., `options(blogdown.publishDir = '../public_site')`, which means the website will be generated to the directory `public_site/` under the parent directory of the current project.

The option `blogdown.widgetsID` is only relevant if your website source is under version control and you have HTML widgets on the website. If this option is `TRUE` (default), the random IDs of HTML widgets will be changed to incremental IDs in the HTML output, so these IDs are unlikely to change every time you recompile your website; otherwise, every time you will get different random IDs.

---

## D.2 LiveReload

As we briefly mentioned in Section ??, you can use `blogdown::serve_site()` to preview a website, and the web page will be automatically rebuilt and reloaded in your web browser when the source file is modified and saved. This is called “LiveReload.”

We have provided two approaches to LiveReload. The default approach is through `servr::http()`, which will continuously watch the website directory for file changes, and rebuild the site when changes are detected. This approach has a few drawbacks:

1. It is relatively slow because the website is fully regenerated every time. This may not be a real problem for Hugo, because Hugo is often fast enough: it takes about

a millisecond to generate one page, so a website with a thousand pages may only take about one second to be fully regenerated.

2. The daemonized server (see Section ??) may not work.

If you are not concerned about the above issues, we recommend that you use the default approach, otherwise you can set the global option `options(blogdown.generator.server = TRUE)` to use an alternative approach to LiveReload, which is based on the native support for LiveReload from the static site generator. At the moment, this has only been tested against Hugo-based websites. It does not work with Jekyll and we were not successful with Hexo, either.

This alternative approach requires two additional R packages to be installed: **processx** (?) and **later** (?). You may use this approach when you primarily work on plain Markdown posts instead of R Markdown posts, because it can be much faster to preview Markdown posts using the web server of Hugo. The web server can be stopped by `blogdown::stop_server()`, and it will always be stopped when the R session is ended, so you can restart your R session if `stop_server()` fails to stop the server for some reason.

The web server is established via the command `hugo server` (see its documentation<sup>2</sup> for details). You can pass command-line arguments via the global option `blogdown.hugo.server`. The default value for this option is `c('-D', '-F')`, which means to render draft and future posts in the preview. We want to highlight a special argument `--navigateToChanged` in a recent version of Hugo, which asks Hugo to automatically navigate to the changed page. For example, you can set the options:

```
options(blogdown.hugo.server = c('-D', '-F', '--navigateToChanged'))
```

Then, when you edit a source file under `content/`, Hugo will automatically show you the corresponding output page in the web browser.

---

<sup>2</sup>[https://gohugo.io/commands/hugo\\_server/](https://gohugo.io/commands/hugo_server/)

Note that Hugo renders and serves the website from memory by default, so no files will be generated to `public/`. If you need to publish the `public/` folder manually, you will have to manually build the website via `blogdown::hugo_build()` or `blogdown::build_site()`.

---

### D.3 Building a website for local preview

The function `blogdown::build_site()` has an argument `local` that defaults to `FALSE`, which means building the website for publishing instead of local previewing. The mode `local = TRUE` is primarily for `blogdown::serve_site()` to serve the website locally. There are three major differences between `local = FALSE` and `TRUE`. When `local = TRUE`:

- The `baseurl` option in `config.toml` is temporarily overridden by `"/` even if you have set it to a full URL like `"http://www.example.com/"`.<sup>3</sup> This is because when a website is to be previewed locally, links should refer to local files. For example, `/about/index.html` should be used instead of the full link `http://www.example.com/about/index.html`; the function `serve_site()` knows that `/about/index.html` means the file under the `public/` directory, and can fetch it and display the content to you, otherwise your browser will take you to the website `http://www.example.com` instead of displaying a local file.
- Draft and future posts are always rendered when `local = TRUE`, but not when `local = FALSE`. This is for you to preview draft and future posts locally. If you know the Hugo command line,<sup>4</sup> it means the `hugo` command is called with the flags `-D -F`, or equivalently, `--buildDrafts --buildFuture`.

---

<sup>3</sup>If your `baseurl` contains a subdirectory, it will be overridden by the subdirectory name. For example, for `baseurl = "http://www.example.com/project/"`, `build_site(local = TRUE)` will temporarily remove the domain name and only use the value `/project/`.

<sup>4</sup><https://gohugo.io/commands/hugo/>

- There is a caching mechanism to speed up building your website: an Rmd file will not be recompiled when its `*.html` output file is newer (in terms of file modification time). If you want to force `build_site(local = TRUE)` to recompile the Rmd file even if it is older than the HTML output, you need to delete the HTML output, or edit the Rmd file so that its modification time will be newer. This caching mechanism does not apply to `local = FALSE`, i.e., `build_site(local = FALSE)` will always recompile all Rmd files, because when you want to publish a site, you may need to recompile everything to make sure the site is fully regenerated. If you have time-consuming code chunks in any Rmd files, you have to use either of these methods to save time:
  - Turn on **knitr**'s caching for time-consuming code chunks, i.e., the chunk option `cache = TRUE`.
  - Do not call `build_site()`, but `blogdown::hugo_build()` instead. The latter does not compile any Rmd files, but simply runs the `hugo` command to build the site. Please use this method only if you are sure that your Rmd files do not need to be recompiled.

You do not need to worry about these details if your website is automatically generated from source via a service like Netlify, which will make use of `baseurl` and not use `-D -F` by default. If you manually publish the `public/` folder, you need to be more careful:

- If your website does not work without the full `baseurl`, or you do not want the draft or future posts to be published, you should not publish the `public/` directory generated by `serve_site()`. Always run `blogdown::build_site()` or `blogdown::hugo_build()` before you upload this directory to a web server.
- If your drafts and future posts contain (time-)sensitive information, you are strongly recommended to delete the `/public/` directory before you rebuild the site for publishing every time, because Hugo never deletes it, and your sensitive information may be rendered by a certain `build_site(local = TRUE)` call last

time and left in the directory. If the website is really important, and you need to make sure you absolutely will not screw up anything every time you publish it, put the `/public/` directory under version control, so you have a chance to see which files were changed before you publish the new site.

---

## D.4 Functions in the **blogdown** package

There are about 20 exported functions in the **blogdown** package, and many more non-exported functions. Exported functions are documented and you can use them after `library(blogdown)` (or via `blogdown::`). Non-exported functions are not documented, but you can access them via `blogdown:::` (the triple-colon syntax). This package is not very complicated, and consists of only about 1800 lines of R code (the number is given by the word-counting command `wc`):

You may take a look at the source code (<https://github.com/rstudio/blogdown>) if you want to know more about a non-exported function. In this section, we selectively list some exported and non-exported functions in the package for your reference.

### D.4.1 Exported functions

Installation: You can install Hugo with `install_hugo()`, update Hugo with `update_hugo()`, and install a Hugo theme with `install_theme()`.

Wrappers of Hugo commands: `hugo_cmd()` is a general wrapper of `system2('hugo', ...)`, and all later functions execute specific Hugo commands based on this general wrapper function; `hugo_version()` executes the command `hugo version` (i.e., `system2('hugo', 'version')` in R); `hugo_build()` executes `hugo` with optional parameters; `new_site()` executes `hugo new site`; `new_content()` executes `hugo new` to create a new content file, and `new_post()` is a wrapper

based on `new_content()` to create a new blog post with appropriate YAML metadata and filename; `hugo_convert()` executes `hugo convert`; `hugo_server()` executes `hugo server`.

Output format: `html_page()` is the only R Markdown output format function in the package. It inherits from `bookdown::html_document2()`, which in turn inherits from `rmarkdown::html_document()`. You need to read the documentation of these two functions to know the possible arguments. Section 1.5 has more detailed information about it.

Helper functions: `shortcode()` is a helper function to write a Hugo shortcode `{{% %}}` in an Rmd post; `shortcode_html()` writes out `{{< >}}`.

Serving a site: `serve_site()` starts a local web server to build and preview a site continuously; you can stop the server via `stop_server()`, or restart your R session.

Dealing with YAML metadata: `find_yaml()` can be used to find content files that contain a specified YAML field with specified values; `find_tags()` and `find_categories()` are wrapper functions based on `find_yaml()` to match specific tags and categories in content files, respectively; `count_yaml()` can be used to calculate the frequencies of specified fields.

#### D.4.2 Non-exported functions

Some functions are not exported in this package because average users are unlikely to use them directly, and we list a subset of them below:

- You can find the path to the Hugo executable via `blogdown::find_hugo()`. If the executable can be found via the `PATH` environment variable, it just returns `'hugo'`.
- The helper function `modify_yaml()` can be used to modify the YAML metadata of a file. It has a `...` argument that takes arbitrary YAML fields, e.g., `blogdown::modify_yaml('foo.md', author = 'Frida Gomam', date = '2015-07-23')` will change the `author` field

in the file `foo.md` to Frida Gomar, and date to 2015-07-23. We have shown the advanced usage of this function in Section 4.1.

- We have also mentioned a series of functions to clean up Markdown posts in Section 4.1. They include `process_file()`, `remove_extra_empty_lines()`, `process_bare_urls()`, `normalize_chars()`, `remove_highlight_tags()`, and `fix_img_tags()`.
- In Section D.3, we mentioned a caching mechanism based on the file modification time. It is implemented in `blogdown::require_rebuild()`, which takes two arguments of filenames. The first file is the output file, and the second is the source file. When the source file is older than the output file, or the output file does not exist or is empty, this function returns `TRUE`.
- The function `blogdown::Rscript()` is a wrapper function to execute the command `Rscript`, which basically means to execute an R script in a new R session. We mentioned this function in Chapter 5.

---

## D.5 Paths of figures and other dependencies

One of the most challenging tasks in developing the **blogdown** package is to properly handle dependency files of web pages. If all pages of a website were plain text without dependencies like images or JavaScript libraries, it would be much easier for me to develop the **blogdown** package.

After **blogdown** compiles each Rmd document to HTML, it will try to detect the dependencies (if there are any) from the HTML source and copy them to the `static/` folder, so that Hugo will copy them to `public/` later. The detection depends on the paths of dependencies. By default, all dependencies, like R plots and libraries for HTML widgets, are generated to the `foo_files/` directory if the Rmd is named `foo.Rmd`. Specifically, R plots are generated to



`foo_files/figure-html/` and the rest of files under `foo_files/` are typically from HTML widgets.

R plots under `content/*/foo_files/figure-html/` are copied to `static/*/foo_files/figure-html/`, and the paths in HTML tags like `` are substituted with `/*/foo_files/figure-html/bar.png`. Note the leading slash indicates the root directory of the published website, and the substitution works because Hugo will copy `/*/foo_files/figure-html/` from `static/` to `public/`.

Any other files under `foo_files/` are treated as dependency files of HTML widgets, and will be copied to `static/rmarkdown-lib/`. The original paths in HTML will also be substituted accordingly, e.g., from `<script src="foo_files/jquery/jquery.min.js">` to `<script src="/rmarkdown-lib/jquery/jquery.min.js">`. It does not matter whether these files are generated by HTML widgets or not. The links on the published website will be correct and typically hidden from the readers of the pages.<sup>5</sup>

You should not modify the **knitr** chunk option `fig.path` or `cache.path` unless the above process is completely clear to you, and you want to handle dependencies by yourself.

In the rare cases when **blogdown** fails to detect and copy some of your dependencies (e.g., you used a fairly sophisticated HTML widget package that writes out files to custom paths), you have two possible choices:

- Do not ignore `_files$` in the option `ignoreFiles` in `config.toml`, do not customize the `permalinks` option, and set the option `uglyURLs` to `true`. This way, **blogdown** will not substitute paths that it cannot recognize, and Hugo will copy these files to `public/`. The relative file locations of the `*.html` file and its dependencies will remain the same when they are copied to `public/`, so all links will continue to work.

---

<sup>5</sup>For example, a reader will not see the `<script>` tag on a page, so it does not really matter what its `src` attribute looks like as long as it is a path that actually exists.

- If you choose to ignore `_files$` or have customized the `perma-links` option, you need to make sure **blogdown** can recognize the dependencies. One approach is to use the path returned by the helper function `blogdown::dep_path()` to write out additional dependency files. Basically this function returns the current `fig.path` option in **knitr**, which defaults to `*_files/figure-html/`. For example, you can generate a plot manually under `dep_path()`, and **blogdown** will process it automatically (copy the file and substitute the image path accordingly).

If you do not understand all these technical details, we recommend that you use the first choice, and you will have to sacrifice custom permanent links and clean URLs (e.g., `/about.html` instead of `/about/`). With this choice, you can also customize the `fig.path` option for code chunks if you want.

---

## D.6 HTML widgets

We do not recommend that you use different HTML widgets from many R packages on the same page, because it is likely to cause conflicts in JavaScript. For example, if your theme uses the jQuery library, it may conflict with the jQuery library used by a certain HTML widget. In this case, you can conditionally load the theme's jQuery library by setting a parameter in the YAML metadata of your post and revising the Hugo template that loads jQuery. Below is the example code to load jQuery conditionally in a Hugo template:

```
{{ if not .Params.exclude_jquery }}  
<script src="path/to/jquery.js"></script>  
{{ end }}
```

Then if you set `exclude_jquery: true` in the YAML metadata of a

post, the theme's jQuery will not be loaded, so there will not be conflicts when your HTML widgets also depend on jQuery.

Another solution is the **widgetframe** package<sup>6</sup> (?). It solves this problem by embedding HTML widgets in `<iframe></iframe>`. Since an iframe is isolated from the main web page on which it is embedded, there will not be any JavaScript conflicts.

A widget is typically not of full width on the page. To set its width to 100%, you can use the chunk option `out.width = "100%"`.

---

## D.7 Version control

If your website source files are under version control, we recommend that you add at least these two folder names to your `.gitignore` file:

```
blogdown
public
```

The `blogdown/` directory is used to store cache files, and they are most likely to be useless to the published website. Only **knitr** may use them, and the published website will not depend on these files.

The `public/` directory should be ignored if your website is to going to be automatically (re)built on a remote server such as Netlify.

As we mentioned in Section D.5, R plots will be copied to `static/`, so you may see new files in GIT after you render an Rmd file that has graphics output. You need to add and commit these new files in GIT, because the website will use them.

Although it is not relevant to **blogdown**, macOS users should remember to ignore `.DS_store` and Windows users should ignore `Thumbs.db`.

---

<sup>6</sup><https://github.com/bhaskarvk/widgetframe>

If you are relatively familiar with GIT, there is a special technique that may be useful for you to manage Hugo themes, which is called “GIT submodules.” A submodule in GIT allows you to manage a particular folder of the main repository using a different remote repository. For example, if you used the default `hugo-lithium-theme` from my GitHub repository, you might want to sync it with my repository occasionally, because I may update it from time to time. You can add the GIT submodule via the command line:

```
git submodule add \  
  https://github.com/yihui/hugo-lithium-theme.git \  
  themes/hugo-lithium-theme
```

If the folder `themes/hugo-lithium-theme` exists, you need to delete it before adding the submodule. Then you can see a SHA string associated with the “folder” `themes/hugo-lithium-theme` in the GIT status of your main repository indicating the version of the submodule. Note that you will only see the SHA string instead of the full content of the folder. Next time when you want to sync with my repository, you may run the command:

```
git submodule update --recursive --remote
```

In general, if you are happy with how your website looks, you do not need to manage the theme using GIT submodules. Future updates in the upstream repository may not really be what you want. In that case, a physical and fixed copy of the theme is more appropriate for you.

---

## D.8 The default HTML template

As we mentioned in Section 1.5, the default output format for an Rmd document in **blogdown** is `blogdown::html_page`. This format passes a minimal HTML template to Pandoc by default:

```
$for(header-includes)$
$header-includes$
$endfor$
$if(highlighting-css)$
<style type="text/css">
$highlighting-css$
</style>
$endif$
$for(css)$
  <link rel="stylesheet" href="$css$" type="text/css" />
$endfor$

$for(include-before)$
$include-before$
$endfor$
$if(toc)$
<div id="$idprefix$TOC">
$toc$
</div>
$endif$

$body$

$for(include-after)$
$include-after$
$endfor$
```

You can find this template file via `blogdown::pkg_file('resources',`

'template-minimal.html') in R, and this file path is the default value of the `template` argument of `html_page()`. You may change this default template, but you should understand what this template is supposed to do first.

If you are familiar with Pandoc templates, you should realize that this is not a complete HTML template, e.g., it does not have the tags `<html>`, `<head>`, or `<body>`. That is because we do not need or want Pandoc to return a full HTML document to us. The main thing we want Pandoc to do is to convert our Markdown document to HTML, and give us the body of the HTML document, which is in the template variable `$body$`. Once we have the body, we can further pass it to Hugo, and Hugo will use its own template to embed the body and generate the full HTML document. Let's explain this by a minimal example. Suppose we have an R Markdown document `foo.Rmd`:

```
---
title: "Hello World"
author: "Yihui Xie"
---

I found a package named blogdown.
```

It is first converted to an HTML file `foo.html` through `html_page()`, and note that YAML metadata are ignored for now:

```
I found a package named blogdown.
```

Then **blogdown** will read the YAML metadata of the Rmd source file, and insert the metadata into the HTML file so it becomes:

```
---
title: "Hello World"
author: "Yihui Xie"
---

I found a package named blogdown.
```

This is the file to be picked up by Hugo and eventually converted to an HTML page of a website. Since the Markdown body has been processed to HTML by Pandoc, Hugo will basically use the HTML. That is how we bypass Hugo's Markdown engine BlackFriday.

Besides `$body$`, you may have noticed other Pandoc template variables like `$header-includes$`, `$css$`, `$include-before$`, `$toc$`, and `$include-after$`. These variables make it possible to customize the `html_page` format. For example, if you want to generate a table of contents, and apply an additional CSS stylesheet to a certain page, you may set `toc` to `true` and pass the stylesheet path to the `css` argument of `html_page()`, e.g.,

```
---
title: "Hello World"
author: "Yihui Xie"
output:
  blogdown::html_page:
    toc: true
    css: "/css/my-style.css"
---
```

---

## D.9 Different building methods

If your website does not contain any Rmd files, it is very straightforward to render it — just a system call to the `hugo` command.

When your website contains Rmd files, **blogdown** has provided two rendering methods to compile these Rmd files. A website can be built using the function `blogdown::build_site()`:

```
build_site(local = FALSE, method = c("html", "custom"),
  run_hugo = TRUE)
```

As mentioned in Section ??, the default value of the `method` argument is determined by the global option `blogdown.method`, and you can set this option in `.Rprofile`.

For `method = 'html'`, `build_site()` renders `*.Rmd` to `*.html`, and `*.Rmarkdown` to `*.markdown`, and keeps the `*.html/*.markdown` output files under the same directory as `*.Rmd/*.Rmarkdown` files.

An Rmd file may generate two directories for figures (`*_files/`) and cache (`*_cache/`), respectively, if you have plot output or HTML widgets (Vaidyanathan et al., 2018) in your R code chunks, or enabled the chunk option `cache = TRUE` for caching. In the figure directory, there will be a subdirectory `figure-html/` that contains your plot output files, and possibly other subdirectories containing HTML dependencies from HTML widgets (e.g., `jquery/`). The figure directory is moved to `/static/`, and the cache directory is moved to `/blogdown/`.

After you run `build_site()`, your website is ready to be compiled by Hugo. This gives you the freedom to use deploying services like Netlify (Chapter ??), where neither R nor **blogdown** is available, but Hugo is.

For `method = 'custom'`, `build_site()` will not process any R Markdown files, nor will it call Hugo to build the site. No matter which method you choose to use, `build_site()` will always look for an R script `/R/build.R` and execute it if it exists. This gives you the complete freedom to do anything you want for the website. For example, you can call `knitr::knit()` to compile Rmd to Markdown (`*.md`) in this R script instead of using `rmarkdown::render()`. This feature is designed for advanced users who are really familiar with



the **knitr** package<sup>7</sup> and Hugo or other static website generators (see Chapter 5).

When `R/build.R` exists and `method = 'html'`, the R Markdown files are knitted first, then the script `R/build.R` is executed, and lastly Hugo is called to build the website.

---

<sup>7</sup>Honestly, it was originally designed for Yihui himself to build his own website, but he realized this feature could actually free users from Hugo. For example, it is possible to use Jekyll (another popular static site generator) with **blogdown**, too.



# E

---

## *Personal Experience*

---

I started blogging at blogchina.com in 2005, moved to blog.com.cn, then MSN Space, and finally purchased my own domain yihui.name and a virtual host. I first used a PHP application named Bo-Blog, then switched to WordPress, and then Jekyll. Finally I moved to Hugo. Although I have moved several times, all my posts have been preserved, and you can still see my first post in Chinese in 2005. I often try my best not to introduce broken links (which lead to the 404 page) every time I change the backend of my website. When it is too hard to preserve the original links of certain pages, I will redirect the broken URLs to the new URLs. That is why it is important for your system to support redirections, and in particular, 301 redirections (Netlify does a nice job here). Here are some of my redirection rules: [https://github.com/rbind/yihui/blob/master/static/\\_redirects](https://github.com/rbind/yihui/blob/master/static/_redirects). For example, <http://yihui.name/en/feed/> was the RSS feed of my old WordPress and Jekyll blogs in English, and Hugo generates the RSS feed to `/en/index.xml` instead, so I need to redirect `/en/feed/` to `/en/index.xml`.

Google has provided several tools to help you know more information about your website. For example, Google Analytics<sup>1</sup> can collect visitor statistics and give speed suggestions for your website. Google Webmasters<sup>2</sup> can show you the broken links it finds. I use these tools frequently by myself.

I firmly believe in the value of writing. Over the years, I have written more than 1000 posts in Chinese and English. Some are long, and most are short. The total size of these text files is about 5

---

<sup>1</sup><https://analytics.google.com>

<sup>2</sup><https://www.google.com/webmasters/>

Mb. In retrospect, most posts are probably not valuable to general readers (some are random thoughts, and some are my rants), but I feel I benefitted a lot from writing in two aspects:

1. If I sit down and focus on writing a small topic for a while, I often feel my thoughts will become clearer. A major difference between writing and talking is that you can always reorganize things and revise them when writing. I do not think writing on social media counts. 140 characters may well be thoughtful, but I feel there is so much chaos there. It is hard to lay out systematic thoughts only through short messages, and these quick messages are often quickly forgotten.
2. I know some bloggers are very much against comments, so they do not open comments to the public. I have not had a very negative experience with comments yet. On the contrary, I constantly find inspirations from comments. For example, I was thinking<sup>3</sup> if it was possible to automatically check R packages on the cloud through Travis CI. At that time (April 2013), I believe not many people in the R community had started using Travis CI, although I'm not sure if I was the first person experimenting with this idea. I felt Travis CI could be promising, but it did not support R back then. Someone named Vincent Arel-Bundock (I still do not know him) told me a hack in a comment, which suddenly lit up my mind and I quickly figured out a solution. In October 2013, Craig Citro started more solid work on the R support on Travis CI. I do not know if he saw my blog post. Anyway, I think Travis CI has made substantial impact on R package developers, which is a great thing for the R community.

Yet another relatively small benefit is that I often go to my own posts to learn some technical stuff that I have forgotten. For example, I find it difficult to remember the syntax of different types of zero-width assertions in Perl-like regular expressions: `(?=...)`,

---

<sup>3</sup><https://yihui.name/en/2013/04/travis-ci-for-r/>

(?!...), (?<=...), and (?<!...). So I wrote a short blog post and gave myself a few minimal examples. After going back to that post a few times, finally I can remember how to use these regular expressions.



---

## ***Bibliography***

---

- Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., and Chang, W. (2018). *rmarkdown: Dynamic Documents for R*. R package version 1.9.
- R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.
- Vaidyanathan, R., Xie, Y., Allaire, J., Cheng, J., and Russell, K. (2018). *htmlwidgets: HTML Widgets for R*. R package version 1.0.
- Xie, Y. (2016). *bookdown: Authoring Books and Technical Documents with R Markdown*. Chapman and Hall/CRC, Boca Raton, Florida. ISBN 978-1138700109.
- Xie, Y. (2018a). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.7.
- Xie, Y. (2018b). *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.20.
- Xie, Y. (2018c). *servr: A Simple HTTP Server to Serve Static Files or Dynamic Documents*. R package version 0.9.
- Xie, Y. (2018d). *xaringan: Presentation Ninja*. R package version 0.6.





---

## *Index*

---

`_default/`, 45  
`404.html`, 53  
  
, 16, 33–35  
  
Amazon S3, 84  
  
`baseUrl`, 31, 39  
Blackfriday, 14  
`blogdown::build_dir()`, 69  
`blogdown::build_one()`, 97  
`blogdown::build_site()`, 138  
`blogdown::hugo_build()`, 139  
`blogdown::install_theme()`, 20  
`blogdown::new_site()`, 3  
`blogdown::serve_site()`, 3  
  
CNAME Record, 132  
Comentarios de Disqus, 40, 59  
Complementos de RStudio, 6  
`config.toml`, 4, 27  
CSS, 116  
  
Dependency Files, 142  
Directories, 5  
Directorio Static, 68  
DNS Records, 131  
Domain Name, 129  
  
Emoji, 31  
  
`fonts.css`, 57  
`footer.html`, 52  
  
Functions, 140  
  
GIT Submodules, 146  
GitHub Pages, 76  
Global Options, 12, 135  
Google Analytics, 40, 58  
  
`hasCJKLanguage`, 33  
`header.html`, 51  
Hexo, 100  
HTML, 111  
HTML Widgets, 144  
Hugo, 2, 25  
Hugo Lithium Theme, 38  
  
`ignoreFiles`, 33  
  
JavaScript, 121  
Jekyll, 88, 95  
jQuery, 123, 144  
  
`knitr::render_jekyll()`, 99  
  
`list.html`, 47  
LiveReload, 3, 136  
Logo, 42  
  
Markdown, 14, 106  
MathJax, 42, 60, 124  
  
Nameservers, 130  
Netlify, 24, 72, 101  
`nombre de dominio`, 74

opciones, 31  
Optimization, 126  
  
Pandoc, 1, 15, 147  
params, 41  
Parciales, 46  
permalinks, 31  
Peso del Post, 35  
pkgdown, 103  
Publicar fecha, 35  
Página de edición de GitHub,  
62  
  
R Markdown, 14, 105  
R Markdown Site Generator,  
102  
Resaltar sintaxis, 59  
  
Shortcode, 35  
single.html, 45  
Site Migration, 87  
Sitio estático, 25  
Slug, 32  
Syntax Highlighting, 41  
  
Tabla de contenidos, 60  
Tags, 61  
Tema XMin, 43  
Temas, 20, 37  
Templates, 42  
terms.html, 49  
TOML, 28  
Travis CI, 80  
  
Updog, 75  
  
Version Control, 145  
  
WordPress, 92  
  
YAML, 28, 106