Setting up a basic website

Håkon Robbestad Gylterud

There are lot of different ways to make a website. You can write your own HTML code (the code your web-browser reads when it draws a webpage), use a WYSIWYG editor or install Wordpress... This guide is for those who wish to focus on the textual content of their site, and keep the number of distractions down. The following are the key points, each of which might warrant further explainations.

- Keep the barrier for writing and editing the webpages low.
- Do the editing on a local machine.
- Make it easy to deploy changes to the server.

Now let us discuss the technical tools I have chosen to implement these key points. They are each in them selves quite simple, but might take a bit of effort to get comfortable with[1]. Here is the list.

- Write the content in Markdown.
- Use a version control system such as darcs.
- Automate your operations using mk or make.
- Mount the server file system locally using sshfs.

We will go through all these. At the end there is a worked example, and a discussion. Before you get started, you need some hosting for your web page. If you can have a machine running at all times, I recommend hosting your own web site, but any hosting which allows you to upload simple html files will work for this tutorial.

Markdown and Pandoc

Writing your own HTML code need not be very stressful, and gives complete flexibility — but with this flexibility comes distractions. If you are a perfectionist

[1] I work a lot with the command line, which for many people seems like an odd instrument from a forgotten past, but it has the advantage that I can completely write down my actions, which would be impossible if I used some graphical application.

you easily get caught up in the struggle of making your page look perfect on every browser. HTML is growing bigger and bigger with new standards popping up every year. Markdown however lets you write your page in a text file, and just slightly hint at the markup — using newlines to separate paragraphs, and #s to mark headings, for example. After writing the text you can convert the file to HTML and put it online.

You can use any text editor to write your markdown files. I recommend any text editor which allows you to write without too many distractions. This page was written in Acme from Plan9Port on Linux.

Pandoc is the document format swiss-army knife. It converts text documents between a lot of formats, including Markdown and HTML. This document is written in Markdown, but you are reading it as a HTML page. The following command was used to convert it:

```
pandoc --standalone --smart -T '' -f markdown -t html5 --css /markdown.css\
 -o tutorials/basic-website.html tutorials/basic-website
```

The `--standalone` gives the html file a header, so that it can stand on its own. The `--smart` tag corrects some typographical quirks. Finally the `-T "` gives the page an automatic title header. As you can see from the command, I have specified a custom css file to be used. The CSS file I use for Markdown can be found here.

Darcs

A *version control* system is a program which keeps track of a collection of files and revisions there of. Typically all versions previous versions are preserved, so that you can go back to an earlier revision, or keep update several, independent versions of your files[2].

Use a version contol system. I recomend darcs, but there are other alternatives. A quick introduction to darcs is found here.

For the purpose of making a web-page the advantage of using a version control system is that one can keep track of all revisions one made. Also, being able to have separate branches means that you can work on bigger changes while at the same time keeping the old version up to date. It also allows undoing changes more easily and cleanly.

One can of course setup a darcs repository where the site is deployed to. This has the advantage of being able to use "darcs push" to update the web-page. There are a couple of reasons why I do not do it this way. The most basic one is that since it is really just the generated HTML we need to put on the server

[2] Or, rather, several *dependent* versions of your files — since darcs makes it easy to selectively import changes from one version to another — which is far more useful.

and revision control is only needed for the Markdown-files, we might as well just copy the HTML-files once they are generated.

mk

Once you have started writing your website you might get tired of typing that long `pandoc` command every time. Also, you might grow tired of copying the HTML files to the server every time you made a minor edit. This is when writing a mkfile (or makefile) is useful. A mkfile contains the commands needed to build a given set of files. Once the mkfile is written, you call on the command `mk` to build them.

I have written a small mkfile which takes care of these things. It uses the Plan 9 from User Space userland (`rc` and `mk` to be precise), so be sure to have those installed.

It depends on having a file called `deployment-index` which lists all the HTML- and CSS files and all directories you want to deploy. Just write a list of the files and make sure that directories end with `/`.

Here is my `mkfile`:

```
MKSHELL=$PLAN9/bin/rc

# This names a file which we read as index.
index='deployment-index'

# Generate HTML-files from markdown

%.html: %
   pandoc --standalone --smart -T '' -f markdown\
    -t html5 --css /markdown.css -o $stem.html $stem

# Virtual targets

all:QV: `{grep -v '/$' $index}


# Deploy all files in $index to $destination

deploy:QV: `{grep -v '/$' $index}
  for (f in `{grep '/$' $index})
    mkdir $destination/$f >[2] /dev/null \
     || echo $destination/$f exists. Omitting .
  for(i in `{grep -v '/$' $index})
    cp -v $i $destination/$i
```

Then the following command:

```
mk deploy 'destination=/path/to/deployed-site'
```

will deploy all the files in `deployment-index` to `/path/to/deployed-site`.

sshfs

When deploying the site can be useful to mount the server using sshfs.

rsync

Alternatively, you can use `rsync(1)` to copy the files to a remote server. The following mk rule assumes you have set the host and destination enviromentvariables:

```
rsyncdeploy:QV: all
  time -f 'Total deployment time: %e seconds' rc -c '
    build=`{mktemp}
    destination=$build mk deploy
    echo ''Synchronising remote copy.''
    rsync -rcvz $build/ $host:$destination
    echo ''Removing local copy.''
    rm -r $build'
```

A word on backup

Keep backups of your files! Use an online service or keep your own backup disks. If you loose the markdown files, you can convert the HTML files back to markdown, but you will loose earlier revisions, and work-in-process. If the host of your web-site has some space for you which do not end up on the internet, it might be a good place to keep the backup.

Having our files managed by Darcs makes it easy to push the current state to a backup — so there are no excuses for not making frequent backups.

Extra credits

Test suite

One thing you can do for extra credits is to add a test suite to your web-page. We do not have any «dynamic content» or fancy programming on the web page,

so a test suite might sound unnessesairy. However, you could make a test suite looking for anti-patterns in your writing. Or maybe you want to spell check your writing before publishing?

As a useful example, the following mk rule will look for dead links:

```
links-check:QV: ‘{grep -v ’/$’ $index | grep ’\.html$’}
  if (~ $base ’’)
     base=‘{pwd}
# Get the absolute path for base
  wd=‘{pwd}
  cd $base
  base=‘{pwd}
  cd $wd
# Get the search and replace string for absolute server links
  ss=’s/^\//’‘{echo $base | ssam -e ’s/\//\\\//g’}^’\//g’
# Iterate through html files and check remote links with curl
  for(i in ‘{grep -v ’/$’ $index | grep ’\.html$’}) {
     cd $base
     newdir=‘{echo $i | ssam -e ’x/[^\/]*$/ d’}
     if (! ~ $newdir ’’)
        cd $newdir
     for (u in ‘{ssam -e ’y/href\=\"[^"]*\"/ d’ < $base/$i \
               | ssam -e ’x/href=/ d’ \
           | ssam -e ’x/\"[^\"]*\"/ y/[^\"]*/ c/\n/’ \
           | grep -v ’^$’ | grep -v ’^#’ | grep -o ’^[^#]*’ \
           | ssam -e $ss } ) {
        if (! test -e $u)
           if (! curl -k -s -r 0-255 -o /dev/null $u) {
              echo $u is invalid in $i
              cd $base
              exit 1}
         }
     cd $base }
```

Darcs can be made aware that we have tests we want to perform by running the following command:

```
% darcs setpref test ’mk links-check’
```

Exercise: Add a test which checks spelling of documents.

Generating other formats

Pandoc can generate many formats. One of the formats it generates is Latex, a powerful document markup language, from which we can generate PDFs.

Assuming you have a LaTeX distribution installed, extending the mkfile with the following lines will add PDF-targets.

```
# Generate PDF-files via tex files.

TEXINPUTS=.:‘{grep ’/$’ $index \
  | ssam -e ’x/\n/ c/:/’ }

%.tex: %
   pandoc --standalone --smart -T ’’ -f markdown\
    -t latex -o $stem.tex $stem

%.pdf: %.tex
   pdflatex -output-directory ‘{echo $stem |\
    ssam ’x/\/[^\/]*$/ d’} $stem.tex
```

Just `mk your-file.pdf` and a PDF-file will be constructed from the `your-file` markdown. Remember to add the pdf to the deployment index!

You can read this tutorial as a PDF here — it was generated in the way outlined above.

Exercise: Read the pandoc documentation, find another useful format to convert a webpage to and add an mk-target for it.

Disussion

This above technique will produce a website which might strike the reader as a primitive web page — limited in its technical abilities compared to the modern interactive ideals in which every page at least has a comment section. This guide is to be to taken as the tiniest grain of sand, put on the other side of the scale — my tiny contribution to the counter-weight of this ideal.

Why do I believe such a counter-weight is needed? It is not because I think that user-interaction is not valuable. If webpages can facilitate fruitful discussions between people who would otherwise remain unconfronted with each other it contributes something valuable to the world. However, contributions by the user is always at the mercy of the administrator and the moderator. What can appear as a free discussion is in reality only as free as the moderator is benevolent. In the best case the moderator is the same as the website author, whose agenda is transparent — at the worst it is a corporation or the state. The worst case seems by far the most common, as many sites, such as blogs, Facebook pages and Twitter accounts, are hosted as ready-made services, where the author is merely a slightly priviledged user.

A simple website, such as one constructed in the manner described here, can even be hosted on an outdated home computer. To censor it would involve

blocking Internet access or physically confiscate the computer, an effort much bigger than the button-click of a moderator of Facebook. Not to mention that it can be restored by simply moving to a new host, given that a backup is availble.

A page without a «like counter» is also not susceptible to the tyranny of the local majority we see so many places. On say, Reddit, your «contribution» is immediately evaluated by other users. The resulting grade colours the way the reader receives your words — the resulting fragmentation into cliques clearly expected by the founders by the facility of «sub-reddits».

The central tension of a dynamic discussion forum is the need for moderators (lest the site becomes infected by spam) and the need for freedom from moderators (lest the site becomes infected by moderator power games and censorship).

How then to facilitate discussion, if not by atomated discussion services? My guess is that a partial solution to this question lies in the realisation that what is called a «static webpage» (for lack of «dynamic» content), need not be static in the real sense of the word, as it can be changed by the owner — and new content and pages may be added. If making your own site is simple, each can have his own to publish his arguments to a discussion. Links between sites enables the reader to pass from one side of the argument to the other.