Further work on jekyll org-mode support

In Working with Jekyll and Org-Mode a simple solution was given to use org-mode format files as posts.

That solution had quite a few limitations:

1. the files still needed the '`---`' *yaml header* to be present on the first lines, making it invalid org-mode documents
2. only posts were supported, not documents, collections and pages
3. the `#+key: value` syntax of org-mode was not used for settings, or at least only partially
4. working with liquid tags was cumbersome.

What follows are some notes on how I solved the issues.

Getting rid of the '`---`' requirement

Jekyll uses the '`---`' header to determine which files need to be processed. If such a header is not present, the file is either copied verbatim or ignored, depending on the settings. Having this header in an org-mode file makes it somewhat invalid. Not a bit deal, but luckily it's rather easy to get rid of.

The function in standard Jekyll that determines this is `has_yaml_header` so by extending that function we can make sure org-mode files are treated to be processed like there was a header.

```
module Utils
  def has_yaml_header?(file)
    !!((File.open(file, 'rb') { |f| f.read(2) } =~ /^#\+/) or
       (File.open(file, 'rb') { |f| f.read(5) } =~ /\A---\r?\n/))
  end
end
```

The check is just for the file to start with `#+` which is enough for me, for now.

Dealing with the yaml header settings

Getting rid of the '`---`'–marker is one thing, but between those markers are settings which are relevant for the document. At least title, tags and layout are usually present in the header. With the yaml-header gone we need a way to register those variables in some org-mode syntax.

The standard org-mode key/value pairs as mentioned above are suitable for that. The method is implemented statically in the OrgConverter class:

```
def self.process_options(content, data)
    org_text = Orgmode::Parser.new(content, {markup_file: "html.tags.yml" })

    org_text.in_buffer_settings.each_pair do |key, value|
      # We need true/false as booleans, not string.
      if key.downcase == 'published' #(any others?)
        value = value.to_b
      end
      data[key.downcase] = value
    end
    data
  end
```

The implementation is not very elegant, but it registers all org-mode buffer settings as values. An exception must be coded for values which must be boolean. (I use only the `published` property, but there may be more)

Handling liquid tags

Here's where things get a bit hairy. Formally, when writing content which requires a liquid construct, it should be enclosed in the proper org-mode block delimiters, something like:

,#+BEGIN$_{\text{HTML}}$ {%raw%}{% liquid construct here %}{%endraw%} ,#+END$_{\text{HTML}}$

such that the whole document is a *normal* org-mode document. As my current documents do not have the '`BEGIN_HTML/END_HTML`' delimiters this would be annoying to change (again). So, the first thing I tried is to implement it to avoid all that changing again. That worked, a fully working implementation is at commit cc2081

However, there are significant issues with this:

1. documents are still *odd* org-mode documents; no real improvement from the originals (just a bit less *non-org-mode* constructs)
2. build performance of jekyll is unacceptable in the implementation (roughly 3 times slower!!)
3. to solve the problems plugins need to be adapted, so it'll quickly become a nightmare.
4. the code already is more complex than it needs to be (include tags, quote replacement)

After realizing this, I started over and just coded the convert method as:

```
def convert(content)
  org_text = Orgmode::Parser.new(content, {markup_file: "html.tags.yml" })
  org_text.in_buffer_settings.delete("TITLE")  # We already captured the title on processing
```

```
  org_text.to_html
end
```

Given I made the change to all my orgmode files as mentioned above, which is not that much work with emacs' `dired-do-query-replace-regexp` command, the conversion is complete. The only thing to make sure is that for each type of source file (post, page, document) the header options are processed with the `process_options` function above.

You can view the end result in commit 40dad2