Console Syntax Highlighting

Denice Deatrich

Created: August 2023 Last update: September 1, 2023

What is Console syntax highlighting

While writing a Pandoc Markdown-based guide about creating a Linux Home Server I spent some time understanding issues and finding solutions. I then added another guide on getting started with Markdown-based technical guides.

One of the issues I have dealt with on both occasions is *console* input/output and syntax-highlighting. When you are helping others understand command line options it is very useful to **highlight** command line console sessions.

Pandoc uses the Haskell *skylighting* library for syntax highlighting. The highlighting descriptions are written in XML, using syntax descriptions designed for KDE's Kate syntax highlighting.

By adding custom Kate syntax highlighting I was able to present console sessions as fenced code blocks with more interesting syntax highlighting in Pandoc-generated Markdown-based documents.

By adding custom Vim syntax highlighting for my favourite editor, Gvim (Graphical Vim), then I was able to edit Markdown files with embedded console sessions with the same syntax highlighting as in the final document.

When the appropriate configuration files are in place then any text file named with a *.console* filename extension written in console-consistent format will be syntax-highlighted with these editors:

- a Vim-based editor
- the KDE Kate editor

I have not yet looked at syntax highlighting console sessions for other editors like gedit, pluma or emacs.

Console is not a language

Most syntax highlighting is based on highlighting the specific syntax of various computer languages. 'Console' is instead a contrived set of syntax requirements to represent console input/output for Linux and UNIX-based command line sessions.

Getting the files

The github repository at https://github.com/deatrich/console-syntax contains the needed configuration files.

This is the list of relevant configuration and test files:

- test.console
- vim.ftdetect
- vim.syntax
- vimrc.example
- console.xml

The *test.console* file can be used to test whether your local configuration works. It also explains the simple syntax requirements which invoke the highlighter.

The contents of this file are embedded in a fenced code block at the end of this document.

Configure Gvim/Vim

For the Gvim or Vim editors, the Vim files should be installed in your home directory:

```
// Change directories to your home directory:
$ cd

// Make the necessary subdirectories:
$ mkdir -p .vim/syntax .vim/ftdetect

// Copy the files into place:
$ cp /path/to/vim.syntax ~/.vim/syntax/console.vim
$ cp /path/to/vim.ftdetect ~/.vim/ftdetect/console.vim
```

To have the same console syntax highlighting embedded in Markdown files then enable the console syntax in your *.vimrc* file by copying the settings in *vimrc.example*.

```
// Create the file if it does not exist; otherwise edit the file.  Add
//  the 'fenced languages' stanza:
$ vim ~/.vimrc
$ grep fenced_languages ~/.vimrc
let g:markdown_fenced_languages = ['console']
```

Configure Kate

For the Kate editor, it will depend on the version of Kate you are using. In my case on Ubuntu LTS 22.04 the console.xml file is installed this way:

```
// Change directories to your home directory:
$ cd

// Make the necessary subdirectory and copy the XML file there:
$ mkdir -p .local/share/katepart5/syntax
$ cp /path/to/console.xml ~/.local/share/katepart5/syntax/
```

To have the same console syntax highlighting with Kate embedded in Markdown files then it is a bit more complicated than the *Vim* example. You need to copy the current Markdown XML syntax file from Kate, and add a few lines to enable fenced console code blocks. I made it work by adding the lines just above the settings for *bash* in two places in the *markdown.xml* file:

```
$ cd ~/.local/share/katepart5/syntax/
// wget the file, or just browse to this link and copy/paste it into a file
// name 'markdown.xml':
$ wget -N -nd \
https://raw.githubusercontent.com/jgm/skylighting/master/skylighting-core/xml/markdown.xml

// Keep a copy of the original file:
$ cp -p markdown.xml markdown.xml.orig

// Edit the file, adding the XML lines noted below:
$ kate markdown.xml
```

The lines to add to *markdown.xml* are these. Most of the fenced code block XML entries resemble each other. Look for *find-lang-fenced-code* to discover the languages with embedded syntax highlighting:

```
<!-- just before first 'bash-code' line add this: -->
 <RegExpr attribute="Fenced Code" context="#pop!console-code"
  String="&fcode;\s*(?:console)&end;" insensitive="true"
  beginRegion="code-block"/>
```

```
<!-- just before second 'bash-code' section add this -->
 <context attribute="Normal Text" lineEndContext="#stay" name="console-code">
     <IncludeRules context="code"/>
     <IncludeRules context="##console" includeAttrib="true"/>
 </context>
```

Example *console* session

```
// ---------------------------------------------------------------------------
// COMMENTS:

// Comments are highlighted differently from all other console input/output.
// In fact, all comments are for instruction only, and are never part of
// terminal console input/output.

// Comments are handy when you want to explain what you are doing, and why.

// Comments start with 2 forward slashes as the first 2 characters, followed
// by a manditory space character.


// ---------------------------------------------------------------------------
// NORMAL CONSOLE OUTPUT MODE:

// All console output is in default output mode, which usually means the reverse
// of your background colour.  So, if your background is black then normal text
// is white or a light colour.  If the background is white then normal text
// is black.  As an example, this is a fragment of 'dmesg' output:

[Fri Aug 18 08:00:41 2023] RPC: Registered named UNIX socket transport module.
[Fri Aug 18 08:00:41 2023] RPC: Registered udp transport module.
[Fri Aug 18 08:00:41 2023] RPC: Registered tcp transport module.
[Fri Aug 18 08:00:41 2023] RPC: Registered tcp NFSv4.1 backchannel transport module.


// ---------------------------------------------------------------------------
// REGULAR USER 'command prompts':

// By regular user, we mean NOT the 'root' superuser.

// If the command shell is a bourne-style shell in a Linux or UNIX environment
// then the prompt is represented by a 'dollar' character as the first character
// followed by a mandatory space character.

// It should be a green colour in bold font.

$
$ whoami
myloginname

// Similarly if the command shell is the C shell then the prompt is
// represented by a 'percent' character as the first character followed by
// a mandatory space character.

%
```

```
% echo $shell
/bin/tcsh


// ------------------------------------------------------------------------
// THE 'root' USER 'command prompt':
//
// For the root user we assume a bourne shell and we represent the command
// prompt by a 'pound' character (also known as the number sign or hash sign)
// as the first character followed by a manditory space character.
// It should be a red colour in bold font.

// We transition from being a regular user to the superuser with sudo here:
$ sudo /bin/sh
[sudo] password for myloginname:
# whoami
root


// ------------------------------------------------------------------------
// COMMAND LINE AND LINE CONTINUATION:

// No matter which of the command prompts is in use, all will keep their
// prompt display properties when a command spreads over more than one
// line, provided that the 'backslash' character appears as the very last
// character of any command line fragment.  It is known as the line continuation
// character.
// Here are some examples.  For bourne shells a greater-than character '>'
// sub-prompt appears, waiting for command completion.  For C shells
// a question mark '?' sub-prompt appears:

$ grep  Firewall: /var/log/messages | \
> awk '{print $11}' | sort -u | \
> sed 's/SRC=//' | head
104.100.67.156
104.103.120.57
104.103.125.29
104.103.126.18
104.103.126.229
104.103.189.128
104.103.211.131
104.193.88.77
104.254.148.251
104.254.148.252

# for i in boot etc root ; do \
> echo tar -zcf $i.tgz /$i; \
> done
tar -zcf boot.tgz /boot
tar -zcf etc.tgz /etc
tar -zcf root.tgz /root

% ps -ef | grep -i xfs | \
? wc -l
59
```

```
// ------------------------------------------------------------------------
// OTHER SHELL-LIKE COMMANDS:
//
// So far, 3 other commands which provide an interactive shell-like interface
// are syntax-highlighted as well.
// They should all be a purple colour in bold font.
// I could add more commands in the future.


// ------------------------------------------------------------------------
// 'virsh' is the 'Virtual Shell' command line interface for managing
//  virtual machines.
// It requires the following text starting in the first position on a line,
// followed by a manditory space:  'virsh # '

virsh #

// Examples:

virsh # help connect
  NAME
    connect - (re)connect to hypervisor
...

virsh # list
 Id    Name                           State
----------------------------------------------------
 1     ubuntu                         running

virsh # quit


// ------------------------------------------------------------------------
// 'mysql' is the interactive console interface for managing a MySQL
// database.
// It requires the following text starting in the first position on a line,
// followed by a manditory space.  A semicolon is required to end the command
// text:

mysql>
 ;

// Using a semi-colon to end a command is normal for traditional mysql
// commands.  A few mysql commands like 'USE', 'QUIT' and 'EXIT' do not
// require a semi-colon to terminate the command, but for the sake of syntax
// highlighting here you must use a semi-colon.

mysql> show variables like '%engine%';
+---------------------------+--------+
| Variable_name             | Value  |
+---------------------------+--------+
| default_storage_engine    | InnoDB |
...

mysql>  grant all privileges on webdb.* to 'rw'@'192.168.1.%'
```

```
    -> identified by 'some_password_here';

mysql> quit;
Bye

$ mysql -u rw -h 192.168.1.63 -p webdb
...
Type 'help;' or '\h' for help...

mysql> insert into members (email) values ('Y')
;
Query OK, 1 row affected (0.00 sec)

mysql> quit;
Bye


// ------------------------------------------------------------------------
// 'mariadb' is the interactive console interface for managing a MariaDB
// database.  The only difference for syntax highlighting purposes between
// mariadb and mysql is the prompt; otherwise the console behaviour is the same.

$ mariadb -u root -p
Enter password:
...
MariaDB> use mysql;
Database changed

MariaDB> show tables like 'time%';
+---------------------------+
| Tables_in_mysql (time%)   |
+---------------------------+
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
+---------------------------+
5 rows in set (0.00 sec)

MariaDB> exit;
Bye

// ------------------------------------------------------------------------
```