
grc Documentation

Release 1.0

Michel Albert

Oct 15, 2021

Contents

1	Generic Colorizer	1
2	WARNING - Pending Project Rename	3
3	Alternatives	5
3.1	The original <code>grc</code>	5
3.2	<code>sed</code> or <code>awk</code>	5
4	Usage	7
4.1	Read lines from <code>stdin</code> and emit modified/colorized lines on <code>stdout</code>	7
4.2	Spawn a subprocess, capture it's output	7
5	Configuration	9
5.1	Syntax	9
5.2	Basic structure	9
5.3	Annotated Example	10
6	Config Reference	13
6.1	Main Level	13
6.2	Contexts	13
6.3	Rules	13
7	Screenshots	15
8	Footnotes	17
9	Indices and tables	19

CHAPTER 1

Generic Colorizer

Note: Inspired by <http://kassiopeia.juls.savba.sk/~garabik/software/grc.html>

`grc` allows you to colorize (even transform) shell output.

WARNING - Pending Project Rename

In order to make room on pypi for the original `grc` project, this project will be renamed to `strec` soon. The new name was chosen to be very different from `grc` to show that it's a different project. While both *do* the same, they do it in a very different manner, and more importantly, the config format is different.

3.1 The original `grc`

Available at <http://kassiopeia.juls.savba.sk/~garabik/software/grc.html>

While the original `grc` is a bit smarter with subprocesses, this rewrite focuses on ease of use (including Installation, *Configuration* and source-code access).

Installation should also honour the [Linux FHS](#)

3.2 `sed` or `awk`

`sed` and `awk` are extremely powerful tools, and can certainly do what `grc` does. They will certainly perform better on large streams. It's their intended use afterall. *However*, they both use an archaic and arcane syntax for their scripts. Additionally, if you would like to colorize your output with these, you need to work with ANSI escape sequences. `grc` aims to simplify this by having a more readable *Configuration* syntax, and by hiding the ANSI escape sequences.

See the installation document for more information.

4.1 Read lines from `stdin` and emit modified/colorized lines on `stdout`

Note: This is the best supported mode of operation.

4.1.1 Synopsis

```
<some_process> | grc -c <config>
```

4.1.2 Example

```
tail -f /var/log/apache2/access.log | grc -c apache_access
```

Advantages

- Only the stream you are sending to `grc` is affected.
- No known side-effects

Disadvantages

- As `grc` only sees a stream, it cannot determine what application is emitting the stream. You have to specify the config manually.

4.2 Spawn a subprocess, capture it's output

Note: Use this if you don't care about the downsides, and are lazy to type.

4.2.1 Synopsis

```
grc <some_proc>
```

4.2.2 Example

```
grc aptitude search python
```

Advantages

- Much less to type
- Can auto-detect the config by using the sub-process application name.

Disadvantages

- Spawning a subprocess and interacting with its IO is non-trivial on a TTY/PTY. To simplify the code, `grc` uses `pexpect` to do the IO magic.
- `stdout` and `stderr` of the subprocess are combined into one stream, which is then emitted on `grc`'s `stdout`.¹
- The output may not use all of the available terminal width.¹

¹ `grc` uses `pyexpect` to deal with TTY peculiarities. This will however have two side-effects. First, `stdout` will be combined with `stderr`. And second, terminal width may not be well respected.

`grc` searches three locations for configuration files in order:

- `~/.grc/conf.d/<confname>.yaml`
- `/etc/grc/conf.d/<confname>.yaml`
- `/usr/share/grc/conf.d/<confname>.yaml`

The first matching config file wins. This means, you can override any system-wide configs with your own concoctions.

5.1 Syntax

`grc` uses [YAML](#) as config syntax. Comparing to `.ini` and `json` files (both included in the Python `stdlib`), this syntax lends itself much better to the requirements of this application.

5.2 Basic structure

- The config file is separated into sections (contexts). It has to have at least the `root` context.
- Each context has a list of rules. These rules fire if a line contains a given regular expression. The first matching rule wins.
- The line will then be replaced with the string contained in the `replace` value. You can use back-refs if you used capture groups in your regular expressions. Colours can be inserted using `${t.color_name}`. You should always insert a `${t.normal}` after using a color, to reset to the terminal default. The colors are provided by the package `blessings`. The `t` variable is a reference to a `blessings` terminal instance so you should be able to use it as it is documented on the `blessings` homepage.
- Rules may define, that processing should *not* stop using the `continue: yes` flag. In that case, the same line will be matched with the following rule as well.
- Additionally, rules may “push” another context onto the stack. If that’s the case, the rule will be processed, and all following lines will be matched against rules contained in the context named by the `push` value.

- If in a non-root context, a rule may “pop” the current context from the stack using the `pop: yes` action.

See *Config Reference* for more details.

5.3 Annotated Example

```
# the primary context. This section must exist!
root:
- match: '^(running)(.*)'
  # demonstrating replacements /and/ colorizing
  replace: '*** ${t.green}\1${t.normal}\2'

- match: '^(writing)(.*)'
  replace: '>>> ${t.yellow}\1${t.normal}\2'

- match: '^(reading)(.*)'
  replace: '<<< ${t.blue}\1${t.normal}\2'

- match: '^(Processing dependencies for)(.*)'
  replace: '${t.green}\1${t.normal}\2'
  # switch to the "dependencies" context
  push: dependencies

- match: '^(Installing.*)'
  replace: '>>> ${t.green}\1${t.normal}'

# the "dependencies" context
dependencies:
- match: '^(Finished processing dependencies for)(.*)'
  replace: '${t.green}\1${t.normal}\2'
  # Revert back to the "root" context
  pop: yes

- match: '^(Searching for )(.*)$'
  replace: '\1${t.blue}\2${t.normal}'
  # switch to the "dependency" context
  push: dependency

# the "dependency" context
dependency:
  # Let's prepend all lines with a small indent and pipe.
  # To do this, we specify a "match-all" regex, replace the line, and
  # specify that we will continue with the next matching rule using
  # "continue"
- match: '(.*)'
  replace: ' | \1'
  continue: yes

  # Note that after the above rule, all lines are prepended with
  # additional text. We need to include this in the regex!
- match: '^ \\\| (Installing.*)'
  replace: ' | >>> ${t.green}\1${t.normal}'

- match: '^ \\\| (Running.*)'
  replace: ' | ${t.green}\1${t.normal}'
```

(continues on next page)

(continued from previous page)

```
- match: '^ \| (Best match.*)'  
  replace: ' | ${t.green}\1${t.normal}'  
  
- match: '^ \| (WARNING|warning)'  
  replace: ' | ${t.yellow}\1${t.normal}'  
  
- match: '^ \| Installed(.*)'  
  replace: ' | Installed\1\n'  
  pop: yes
```


6.1 Main Level

root Specifies the primary context

All other keys represent a context you pushed somewhere.

6.2 Contexts

A context is simply a list of rules

6.3 Rules

match *Type:* string

A [python regular expression](#). If this matches somewhere in the input line, all occurrences will be replaced with the string specified in `replace`.

Note: While YAML does not enforce you to enclose strings in quotes, I is strongly recommend you use **single** quotes for regexps to avoid trouble with string escapes (backslashes).

replace *Type:* string

If `continue` is false (the default), this string will be emitted to `stdout`. Otherwise, this string will be passed to the next matching rule. Not that the following rule sees the *modified* string!

Note: While YAML does not enforce you to enclose strings in quotes, I is recommend using **single** quotes if using backreferences (backslashes).

continue *Type:* boolean

If true, don't write the string yet to `stdout`. Instead, pass it on to the next matching rule.

push *Type:* string

Pushes a new context onto the stack. All following lines from `stdin` will be matched against rules in the new context.

Note: This may change in a future release to give you yet more control

pop *Type:* boolean

If this is set to true, then return to the previous context after this rule has been processed. If in the `root` context, this is a no-op.

Note: This may change in a future release to give you yet more control

A python setup session

Before

```

└─┬─ grc $ ./env/bin/python setup.py develop
   │ running develop
   │ running egg_info
   │ writing requirements to grc.egg-info/requirements.txt
   │ writing grc.egg-info/PKG-INFO
   │ writing top-level names to grc.egg-info/top_level.txt
   │ writing dependency links to grc.egg-info/dependency_links.txt
   │ reading manifest file 'grc.egg-info/SOURCES.txt'
   │ writing manifest file 'grc.egg-info/SOURCES.txt'
   │ running build_ext
   │ Creating /mnt/exhuma/work/grc/env/lib/python2.6/site-packages/grc.egg-link (link to .)
   │ grc 1.0dev1 is already the active version in easy-install.pth
   │ Installing grc script to /mnt/exhuma/work/grc/env/bin
   │
   │ Installed /mnt/exhuma/work/grc
   │ Processing dependencies for grc==1.0dev1
   │ Searching for PyYAML==3.10
   │ Best match: PyYAML 3.10
   │ Processing PyYAML-3.10-py2.6-linux-i686.egg
   │ PyYAML 3.10 is already the active version in easy-install.pth
   │
   │ Using /mnt/exhuma/work/grc/env/lib/python2.6/site-packages/PyYAML-3.10-py2.6-linux-i686.egg
   └─┬─ Finished processing dependencies for grc==1.0dev1

```

After

```

└─┬─ grc $ ./env/bin/python setup.py develop | ./env/bin/grc -c pysetup
   │ *** running develop
   │ *** running egg_info
   │ >>> writing requirements to grc.egg-info/requirements.txt
   │ >>> writing grc.egg-info/PKG-INFO
   │ >>> writing top-level names to grc.egg-info/top_level.txt
   │ >>> writing dependency links to grc.egg-info/dependency_links.txt
   │ >>> reading manifest file 'grc.egg-info/SOURCES.txt'
   │ >>> writing manifest file 'grc.egg-info/SOURCES.txt'
   │ *** running build_ext
   │ Creating /mnt/exhuma/work/grc/env/lib/python2.6/site-packages/grc.egg-link (link to .)
   │ grc 1.0dev1 is already the active version in easy-install.pth
   │ >>> Installing grc script to /mnt/exhuma/work/grc/env/bin
   │
   │ Installed /mnt/exhuma/work/grc
   │ Processing dependencies for grc==1.0dev1
   │ Searching for PyYAML==3.10
   │ | Best match: PyYAML 3.10
   │ | Processing PyYAML-3.10-py2.6-linux-i686.egg
   │ | PyYAML 3.10 is already the active version in easy-install.pth
   │ |
   │ | Using /mnt/exhuma/work/grc/env/lib/python2.6/site-packages/PyYAML-3.10-py2.6-linux-i686.egg
   └─┬─ Finished processing dependencies for grc==1.0dev1

```

Simple aptitude search

Before

```

└─┬─ grc $ aptitude search "git-"
   │ p git-arch - fast, scalable, distributed revision control syste
   │ p git-buildpackage - Suite to help with Debian packages in Git reposito
   │ p git-cola - highly caffeinated git gui
   │ v git-completion -
   │ i git-core - fast, scalable, distributed revision control syste
   │ p git-cvs - fast, scalable, distributed revision control syste
   │ p git-daemon-run - fast, scalable, distributed revision control syste
   │ p git-doc - fast, scalable, distributed revision control syste
   │ p git-email - fast, scalable, distributed revision control syste
   │ p git-gui - fast, scalable, distributed revision control syste
   │ p git-load-dirs - Import upstream archives into git
   │ p git-svn - fast, scalable, distributed revision control syste
   │ p libgit-ruby - Ruby implementation of the Git revision control sy
   │ p libgit-ruby1.8 - Ruby implementation of the Git revision control sy
   │ p stgit-contrib - set of contributed script to help working with stg

```

After

```

└─┬─ grc $ aptitude search "git-" | ./env/bin/grc -c aptitude
   │ git-arch - fast, scalable, distributed revision contr
   │ git-buildpackage - Suite to help with Debian packages in Git
   │ git-cola - highly caffeinated git gui
   │ git-completion -
   │ git-core - fast, scalable, distributed revision contr
   │ git-cvs - fast, scalable, distributed revision contr
   │ git-daemon-run - fast, scalable, distributed revision contr
   │ git-doc - fast, scalable, distributed revision contr
   │ git-email - fast, scalable, distributed revision contr
   │ git-gui - fast, scalable, distributed revision contr
   │ git-load-dirs - Import upstream archives into git
   │ git-svn - fast, scalable, distributed revision contr
   │ libgit-ruby - Ruby implementation of the Git revision co
   │ libgit-ruby1.8 - Ruby implementation of the Git revision co
   │ stgit-contrib - set of contributed script to help working

```

Apache access_log	
Before	After
<pre> 1 grc \$ head /var/log/apache2/access_log 192.168.1.13 - - [11/Dec/2011:09:47:48 +0100] "OPTIONS / HTTP/1.1" 200 22 0 "-" "Microsoft-WebDAV-MiniRedir/6.1.7601" 192.168.1.13 - - [11/Dec/2011:09:47:50 +0100] "PROPFIND / HTTP/1.1" 405 5 81 "-" "Microsoft-WebDAV-MiniRedir/6.1.7601" 192.168.1.13 - - [11/Dec/2011:09:47:52 +0100] "PROPFIND / HTTP/1.1" 405 574 "-" "Microsoft-WebDAV-MiniRedir/6.1.7601" 192.168.1.13 - - [11/Dec/2011:10:03:52 +0100] "OPTIONS / HTTP/1.1" 200 22 0 "-" "Microsoft-WebDAV-MiniRedir/6.1.7601" 192.168.1.13 - - [11/Dec/2011:10:03:54 +0100] "PROPFIND / HTTP/1.1" 405 5 81 "-" "Microsoft-WebDAV-MiniRedir/6.1.7601" 192.168.1.13 - - [11/Dec/2011:10:03:56 +0100] "PROPFIND / HTTP/1.1" 405 574 "-" "Microsoft-WebDAV-MiniRedir/6.1.7601" 192.168.1.13 - - [11/Dec/2011:10:06:05 +0100] "OPTIONS / HTTP/1.1" 200 22 0 "-" "Microsoft-WebDAV-MiniRedir/6.1.7601" 192.168.1.13 - - [11/Dec/2011:10:06:07 +0100] "PROPFIND / HTTP/1.1" 405 5 81 "-" "Microsoft-WebDAV-MiniRedir/6.1.7601" 192.168.1.13 - - [11/Dec/2011:10:06:09 +0100] "PROPFIND / HTTP/1.1" 405 574 "-" "Microsoft-WebDAV-MiniRedir/6.1.7601" 192.168.1.13 - - [11/Dec/2011:10:08:04 +0100] "OPTIONS / HTTP/1.1" 200 22 0 "-" "Microsoft-WebDAV-MiniRedir/6.1.7601" </pre>	<pre> 1 grc \$ head /var/log/apache2/access_log /env/bin/grc -c apache_access 192.168.1.13 - - [11/Dec/2011:09:47:48 +0100] "OPTIONS / HTTP/1.1" 200 22 20 "-" "Microsoft-WebDAV-MiniRedir/6.1.7601" 192.168.1.13 - - [11/Dec/2011:09:47:50 +0100] "PROPFIND / HTTP/1.1" 405 581 "-" "Microsoft-WebDAV-MiniRedir/6.1.7601" 192.168.1.13 - - [11/Dec/2011:09:47:52 +0100] "PROPFIND / HTTP/1.1" 405 574 "-" "Microsoft-WebDAV-MiniRedir/6.1.7601" 192.168.1.13 - - [11/Dec/2011:10:03:52 +0100] "OPTIONS / HTTP/1.1" 200 22 20 "-" "Microsoft-WebDAV-MiniRedir/6.1.7601" 192.168.1.13 - - [11/Dec/2011:10:03:54 +0100] "PROPFIND / HTTP/1.1" 405 581 "-" "Microsoft-WebDAV-MiniRedir/6.1.7601" 192.168.1.13 - - [11/Dec/2011:10:03:56 +0100] "PROPFIND / HTTP/1.1" 405 574 "-" "Microsoft-WebDAV-MiniRedir/6.1.7601" 192.168.1.13 - - [11/Dec/2011:10:06:05 +0100] "OPTIONS / HTTP/1.1" 200 22 20 "-" "Microsoft-WebDAV-MiniRedir/6.1.7601" 192.168.1.13 - - [11/Dec/2011:10:06:07 +0100] "PROPFIND / HTTP/1.1" 405 581 "-" "Microsoft-WebDAV-MiniRedir/6.1.7601" 192.168.1.13 - - [11/Dec/2011:10:06:09 +0100] "PROPFIND / HTTP/1.1" 405 574 "-" "Microsoft-WebDAV-MiniRedir/6.1.7601" 192.168.1.13 - - [11/Dec/2011:10:08:04 +0100] "OPTIONS / HTTP/1.1" 200 22 20 "-" "Microsoft-WebDAV-MiniRedir/6.1.7601" </pre>

CHAPTER 8

Footnotes

CHAPTER 9

Indices and tables

- `genindex`
- `modindex`
- `search`