Logiciel

cli : plugin Command Line Interface pour Dokuwiki

Ce plugin formate la sortie d'une ligne de commande, par exemple pour un manuel d'utilisation ou un tutoriel en ligne. Il est conçu pour fonctionner avec la sortie d'un shell Unix standard, mais devrait convenir pour documenter d'autres types d'interaction CLI, par ex. Fenêtre de commande Windows, Python et Ruby, Matlab, etc.

Sa conception permet d'afficher une transcription CLI (par exemple, une copie de texte à partir d'une fenêtre de console) avec un minimum de balisage supplémentaire.

On peut ajuster le style de l'affichage à l'aide d'une feuille de style. Pour plus de détails, reportez-vous à style.css.

Introduction

Pré-requis

On suppose que :

- toutes les commandes utilisateur commencent par une invite
- l'invite CLI se terminera par un caractère reconnaissable (généralement **$** ou **>**)
- les commandes utilisateur suivent l'invite CLI sur la même ligne
- les lignes qui ne commencent pas par une invite sont des sorties de CLI [1)]

Avec PHP 7

Pour les plugins qui donnent des erreurs, éditez le fichier **lib/plugins/<plugin>/syntax.php** pour le modifier comme ceci :

- remplacez tous les **&$renderer** par **Doku_Renderer $renderer**
- remplacez tous les **&$handler** par **Doku_Handler $handler**

comme indiqué dans les messages d'erreur

Installation

Le code source de ce plugin est hébergé chez GitHub à https://github.com/cpjobling/plugin-cli.

Pour installer le plugin lorsque **git** est installé sur votre serveur DokuWiki, il vous suffit de lancer depuis votre serveur :

- ```
  $ cd ../dokuwiki/plugins
  $ git clone git://github.com/cpjobling/plugin-cli.git cli
  ```

Le dossier contiendra :

style.css
: tous les styles pour cli

syntax.php
: plugin script

Le plugin est maintenant installé.

Details

syntax.php

```
<?php
/**
 * Command Line Interface (CLI) Plugin
 *     Typeset transcripts of interactive sessions with mimumum
effort.
 * Syntax:
 *     <cli prompt="$ " continue="> " comment="#">
 *     user@host:~/somedir $ ls \
 *     > # List directory
 *     file1 file2
 *     </cli>
 *   prompt --- [optional] prompt character used. '$ ' is default
- note the space.
 *   comment --- [optional] comment character used. '#' is default
- note no space.
 *   continue --- [optional] regex of shell continuation '/^> /'
is the default.
 *   The defaults above match Bourne shell ${PS1} and ${PS2}
prompts and comment
 *
 * Acknowledgements:
 *  Borrows heavily from the boxes plugin!
 *  Support for continuation added by Andy Webber
 *  Improved parsing added by Stephane Chazelas
 *
```
```

```
 * @license    GPL 2 (http://www.gnu.org/licenses/gpl.html)
 * @author     Chris P. Jobling <C.P.Jobling@Swansea.ac.uk>
 */

if(!defined('DOKU_INC'))
define('DOKU_INC',realpath(dirname(__FILE__).'/../../').'/');
if(!defined('DOKU_PLUGIN'))
define('DOKU_PLUGIN',DOKU_INC.'lib/plugins/');
require_once(DOKU_PLUGIN.'syntax.php');

/**
 * All DokuWiki plugins to extend the parser/rendering mechanism
 * need to inherit from this class
 */
class syntax_plugin_cli extends DokuWiki_Syntax_Plugin {

    var $prompt_str = '$ ';
    var $prompt_cont = '/^> /'; // this is a regex
    var $prompt_continues = false;
    var $comment_str = '#';

    /**
     * return some info
     */
    function getInfo(){
        return array(
            'author' => 'Chris P. Jobling; Stephan Chazelas; Andy
Webber',
            'email'  => 'C.P.Jobling@Swansea.ac.uk',
            'date'   => '2008-13-02',
            'name'   => 'Command Line Interface (CLI) Plugin',
            'desc'   => 'Renders transcripts of command line
interactions, e.g. for shell and dynamic language interpretor
tutorials',
            'url'    =>
'http://eehope.swan.ac.uk/dokuwiki/plugins:cli',
        );
    }

    /**
     * What kind of syntax are we?
     */
    function getType(){
        return 'protected';
    }

    /**
     * What kind of syntax do we allow (optional)
     */
//    function getAllowedTypes() {
//        return array();
```

```
//    }

    // override default accepts() method to allow nesting
    // - ie, to get the plugin accepts its own entry syntax
    function accepts($mode) {
        if ($mode == substr(get_class($this), 7)) return true;
        return parent::accepts($mode);
    }

    /**
     * What about paragraphs? (optional)
     */
    function getPType(){
        return 'block';
    }

    /**
     * Where to sort in?
     */
    function getSort(){
        return 601;
    }


    /**
     * Connect pattern to lexer
     */
    function connectTo($mode) {
         $this->Lexer->addEntryPattern('<cli(?:[)]?' .
             '"(?:\\\\.|[^\\\\"])*"' .     /* double-quoted string */
             '|\'(?:\\\\.|[^\'\\\\])*\'' . /* single-quoted string */
             '|\\\\.' .                    /* escaped character */
'|[^\'"\\\\>]|[(?:])*>\r?\n?(?=.*?</cli>)',$mode,'plugin_cli');
         /*
          * The [)]? and |[(?:] is to work around a bug in lexer.php
          * wrt nested (...)
         */
    }

    function postConnect() {
       $this->Lexer->addExitPattern('\r?\n?</cli>','plugin_cli');
    }


    /**
     * Handle the match
     */
```

```
    function handle($match, $state, $pos, &$handler){
        switch ($state) {
          case DOKU_LEXER_ENTER :
            $args = substr($match, 4, -1);
            return array($state, $args);
          case DOKU_LEXER_MATCHED :
            break;
          case DOKU_LEXER_UNMATCHED :
            return array($state, $match);
          case DOKU_LEXER_EXIT :
            return array($state, '');
          case DOKU_LEXER_SPECIAL :
            break;
        }
        return array();
    }

    /**
     * Create output
     */
    function render($mode, &$renderer, $data) {
        if($mode == 'xhtml'){
             list($state, $match) = $data;
             switch ($state) {
             case DOKU_LEXER_ENTER :
                 $args = $match;
                 $this->_process_args($args);
                 $renderer->doc .= '<pre class="cli">';
                 break;
             case DOKU_LEXER_UNMATCHED :
                 $this->_render_conversation($match, $renderer);
                 break;
             case DOKU_LEXER_EXIT :
                 $renderer->doc .= "</pre>";
                 break;
             }
             return true;
        }
        return false;
    }

     function _extract($args, $param) {
         /*
          * extracts value from $args for $param
          * xxx = "foo\"bar"  -> foo"bar
          * xxx = a\ b        -> a b
          * xxx = 'a\' b'     -> a' b
          *
          * returns null if value is empty.
          */
         if (preg_match("/$param" . '\s*=\s*(' .
```

```
            '"(?:\\\\.|[^\\\\"])*"' .     /* double-quoted string
*/
            '|\'(?:\\\\.|[^\'\\\\])*\'' . /* single-quoted string
*/
            '|(?:\\\\.|[^\\\\\s])*' .     /* escaped characters
*/
            ')/', $args, $matches)) {
            switch (substr($matches[1], 0, 1)) {
            case "'":
                $result = substr($matches[1], 1, -1);
                $result = preg_replace('/\\\\([\'\\\\])/', '$1',
$result);
                break;
            case '"':
                $result = substr($matches[1], 1, -1);
                $result = preg_replace('/\\\\(["\\\\])/', '$1',
$result);
                break;
            default:
                $result = preg_replace('/\\\\(.)/', '$1',
$matches[1]);
            }
            if ($result != "")
                return $result;
        }
    }

     function _process_args($args) {
        // process args to CLI tag: sets $comment_str and
$prompt_str
        if (!is_null($prompt = $this->_extract($args, 'prompt')))
            $this->prompt_str = $prompt;
        if (!is_null($comment = $this->_extract($args,
'comment')))
            $this->comment_str = $comment;
    }

    function _render_conversation($match, &$renderer) {
        $prompt_continues = false;
        $lines = preg_split('/\n\r|\n|\r/',$match);
        if ( trim($lines[0]) == "" ) unset( $lines[0] );
        if ( trim($lines[count($lines)]) == "" ) unset(
$lines[count($lines)] );
        foreach ($lines as $line) {
            $index = strpos($line, $this->prompt_str);
            if ($index === false) {
                if ($this->prompt_continues) {
                  if (preg_match($this->prompt_cont, $line,
$promptc) === 0) $this->prompt_continues = false;
                }
```

```
                if ($this->prompt_continues) {
                    // format prompt
                    $renderer->doc .= '<span class="cli_prompt">'
. $renderer->_xmlEntities($promptc[0]) . "</span>";
                    // Split line into command + optional comment
(only end-of-line comments supported)
                    $command =  preg_split($this->prompt_cont,
$line);
                    $commands = explode($this->comment_str,
$command[1]);
                    // Render command
                    $renderer->doc .= '<span class="cli_command">'
. $renderer->_xmlEntities($commands[0]) . "</span>";
                    // Render comment if there is one
                    if ($commands[1]) {
                        $renderer->doc .= '<span
class="cli_comment">' .
$renderer->_xmlEntities($this->comment_str . $commands[1]) .
"</span>";
                    }
                    $renderer->doc .= DOKU_LF;
                } else {
                    // render as output
                    $renderer->doc .= '<span class="cli_output">' .
$renderer->_xmlEntities($line) . "</span>" . DOKU_LF;
                    $this->prompt_continues=false;
                }
            } else {
                $this->prompt_continues = true;
                // format prompt
                $prompt = substr($line, 0, $index) .
$this->prompt_str;
                $renderer->doc .= '<span class="cli_prompt">' .
$renderer->_xmlEntities($prompt) . "</span>";
                // Split line into command + optional comment
(only end-of-line comments supported)
                $commands = explode($this->comment_str,
substr($line, $index + strlen($this->prompt_str)));
                // Render command
                 $renderer->doc .= '<span class="cli_command">' .
$renderer->_xmlEntities($commands[0]) . "</span>";
                // Render comment if there is one
                if ($commands[1]) {
                     $renderer->doc .= '<span
class="cli_comment">' .
                        $renderer->_xmlEntities($this->comment_str
. $commands[1]) . "</span>";
                }
                 $renderer->doc .= DOKU_LF;
            }
        }
```

```
    }
}
//Setup VIM: ex: et ts=4 enc=utf-8 sw=4 :
?>
```

style.css

These may be modified to suit your own requirements.

style.css

```
/* plugin:cli */

.cli_output {
    color: blue;
}

.cli_comment {
    color: brown;
}

.cli_prompt {
    color: green;
}

.cli_command {
    color: red;
}

// nested CLI
pre.cli pre.cli {
    background-color: #F8F8F8;
}

/* end plugin:cli */
```

Configuration

Le plugin n'a pas de paramètres de configuration, bien que vous souhaitiez peut-être revoir le jeu de couleurs par défaut dans style.css pour vous assurer qu'il convient à votre wiki.

Utilisation

Syntaxe

Une interaction Bash simple :

```
<cli>
user@host:~/somedir $ ls # List current directory
conf      lang         README        screen.gif  ui
info.txt  manager.dat  renderer.php  syntax.php
user@host:~/somedir $
</cli>
```

s'affiche ainsi :

```
user@host:~/somedir $ ls # List current directory
conf      lang         README        screen.gif  ui
info.txt  manager.dat  renderer.php  syntax.php
user@host:~/somedir $
```

Syntaxe complète :

```
<cli prompt='prompt ' comment='comment'>
transcript
</cli>
```

s'affiche :

```
transcript
```

prompt — [facultatif]
: invite utilisée par CLI. [2] Si aucune invite n'est donnée, '$' est supposé (notez l'espace) est utilisé [3].

comment — [facultatif]
: chaîne de commentaire utilisée par CLI. Si omis, '#' est supposé [4].

continue – [facultatif]
: invite utilisée pour les marqueurs de continuation: regex '/^> /' est la valeur par défaut.

- Les valeurs par défaut ci-dessus correspondent aux invites et commentaires $ {PS1} et $ {PS2} de Bourne shell

Le <cli ... > du début doit apparaître sur une seule ligne. Le contenu de la transcription peut apparaître sur autant de lignes que nécessaire.

Exemples

Cette page fournit un ensemble de tests pour cli et sert également d'exemple de son utilisation.

Script shell de base

Texte :

```
<cli>
user@host:~/somedir $ ls
conf      lang         README        screen.gif  ui
info.txt  manager.dat  renderer.php  syntax.php
user@host:~/somedir $ wc info.txt # count words in info.txt
55 108 1032 info.txt
user@host:~/somedir $
</cli>
```

Résultat :

```
user@host:~/somedir $ ls
conf      lang         README        screen.gif  ui
info.txt  manager.dat  renderer.php  syntax.php
user@host:~/somedir $ wc info.txt # count words in info.txt
55 108 1032 info.txt
user@host:~/somedir $
```

Script shell avec commentaires

Texte :

```
<cli comment="#">
user@host:~/somedir $ ls # List current directory
conf      lang         README        screen,gif  ui
info.txt  manager.dat  renderer.php  syntax.php
user@host:~/somedir $
</cli>
```

Script shell root avec commentaires

Texte : (caractère de commentaire shell par défaut):

```
<cli prompt="#">
root@host:~user/somedir # ls # List current directory
conf      lang         README        screen,gif  ui
info.txt  manager.dat  renderer.php  syntax.php
root@host:~user/somedir #
</cli>
```

Résultat :

```
root@host:~user/somedir # ls # List current directory
conf      lang         README        screen,gif  ui
info.txt  manager.dat  renderer.php  syntax.php
root@host:~user/somedir #
```

Ceci est également valable :

```
<cli prompt="# " comment="#">
root@host:~user/somedir # ls # List current directory
conf      lang         README        screen,gif  ui
info.txt  manager.dat  renderer.php  syntax.php
root@host:~user/somedir #
</cli>
```

```
root@host:~user/somedir # ls # List current directory
conf      lang         README        screen,gif  ui
info.txt  manager.dat  renderer.php  syntax.php
root@host:~user/somedir #
```

Exemple avec une invite de continuation:

```
<cli prompt="$ " continue="> " comment="#">
user@host:~/somedir $ ls \
> # List directory
     file1 file2
</cli>
```

```
user@host:~/somedir $ ls \
> # List directory
file1 file2
```

Script shell avec commentaires

```
<cli prompt="$" comment="#">
user@host:~/somedir $ ls # List current directory
conf      lang         README        screen,gif  ui
info.txt  manager.dat  renderer.php  syntax.php
user@host:~/somedir $
</cli>
```

Fenêtre de commande Windows

Texte :

```
<cli prompt=">" comment="REM">
C:\Users\User>REM hello world!
C:\Users\User>echo 'hello world!'
'hello world!'
</cli>
```

Résultat :

```
C:\Users\User>REM hello world!
```

```
C:\Users\User>echo 'hello world!'
'hello world!'
```

Ruby irb

Une implémentation simple ne fonctionnera pas pour les résultats car la fin de l'invite est identique au marqueur de résultats !

```
<cli prompt=">">
irb(main):001:0> 2+2
=> 4
irb(main):002:0>
</cli>
```

Python

```
<cli prompt=">>>">
ActivePython 2.5.1.1 (ActiveState Software Inc.) based on
Python 2.5.1 (r251:54863, May  1 2007, 17:47:05) [MSC v.1310 32 bit (Intel)]
on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+2
4
>>>
</cli>
```

Python + Windows Shell (Nested CLIs)

Texte :

```
<cli prompt=">">
C:\Users\Chris Jobling>python
<cli prompt=">>>">
ActivePython 2.5.1.1 (ActiveState Software Inc.) based on
Python 2.5.1 (r251:54863, May  1 2007, 17:47:05) [MSC v.1310 32 bit (Intel)]
on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+2
4
>>> ^Z

</cli>
C:\Users\Chris Jobling>
</cli>
```

Résultat :

```
C:\Users\Chris Jobling>python
ActivePython 2.5.1.1 (ActiveState Software Inc.) based on
Python 2.5.1 (r251:54863, May  1 2007, 17:47:05) [MSC v.1310 32 bit (Intel)]
on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+2
4
>>> ^Z
C:\Users\Chris Jobling>
```

Cas particuliers

- Les codes suivants ne s'affichent pas correctement

```
<cli prompt="#">
# rpm -ivh darcs-1.0.9-3.fc6.i386.rpm
Preparing...                ###########################################
[100%]
   1:darcs                  ###########################################
[100%]
</cli>
```

```
# rpm -ivh darcs-1.0.9-3.fc6.i386.rpm
Preparing...                ###########################################
[100%]
   1:darcs                  ###########################################
[100%]
```

Pas sûr de pouvoir résoudre ce problème car le marqueur de progression de téléchargement utilise le même caractère que l'invite !

- CLI ne doit pas supprimer silencieusement des lignes blanches intentionnelles:

```
user@host:~/somedir $ ls # List current directory
conf      lang         README        screen,gif  ui
info.txt  manager.dat  renderer.php  syntax.php


user@host:~/somedir $ # I intended there to be two blank lines above!
```

Désinstallation

Voir aussi

- **(fr)** Développement d'extensions (plugins)
- **(en)** UNIX Tutorial for Beginners
- **(en)** https://www.dokuwiki.org/plugin:cli?s[]=command&s[]=line
- **(en)** Wiki MoinMoin
- **(fr)** TiddlyWiki

Basé sur « Article » par Auteur.

1)
c'est un problème pour le shell interactif de Ruby qui semble préfixer chaque ligne avec une invite !
2)
En pratique, seul le caractère final est nécessaire, car tout ce qui se trouve sur la ligne jusqu'au caractère final sera considéré comme faisant partie de l'invite et tout ce qui suivra sera considéré comme une commande.
3)
'$ ' est l'invite standard du shell Bash
4)
'#' est le caractère de commentaire du shell Bash

From:
https://nfrappe.fr/doc-0/ - **Documentation du Dr Nicolas Frappé**

Permanent link:
https://nfrappe.fr/doc-0/doku.php?id=logiciel:internet:dokuwiki:plugins:cli:start

Last update: **2022/10/01 23:26**