Ficha 2.2 – REGEX

Tópicos abordados:

- Expressões regulares:
  - Na linha de comandos;
  - No PERL:
    - Carateres especiais;
    - Quantificadores;
    - Grupo de carateres;
    - Delimitadores;
    - Opções;
    - Precedências;
    - Exemplos.
    - Grupo e Captura
  - Exercícios;
- Documentação;
- Bibliografia.

Informação: uma PERL Quick Reference Card orientada para expressões regulares está em: http://files.meetup.com/120908/Regexp%20Quick%20Reference.pdf.

1. Expressões regulares (*REGEX – REGular EXpressions*)

1.1. Expressões regulares na linha de comandos

A execução do comando **grep** efetua a pesquisa de todas as linhas de um ficheiro (ex: "/etc/passwd") que contém a palavra **root**, da seguinte forma:

```
$ grep "root" /etc/passwd
```

root:x:0:0:root:/root:/bin/bash

Neste caso, a expressão a procurar foi uma palavra (root), mas poderemos também especificar uma expressão mais complexa. Com o uso do mesmo comando **grep** e alterando a expressão de procura, podemos então obter resultados mais latos ou mais específicos, como por exemplo:

$ grep "^u.*home" /etc/passwd

u*sbmux:x:108:46:usbmux deamon,,,:/****home****/usbmux:/bin/false*

u*ser:x:1000:1000:user,,,:/****home****/user:/bin/bash*

Neste caso, o comando grep vai encontrar expressões que obedecem ao critério especificado, ou seja, **encontra palavras** nas linhas do ficheiro **/etc/passwd** que comecem por **'u'**, e que de seguida tenham qualquer caracter (.) zero ou mais vezes (*), e que de seguida apresentem a sequência de caracters "**home"**.

1.2. Expressões regulares no PERL

As expressões regulares (REGEX, abreviatura de REGular EXpressions) têm um papel fundamental em PERL e são usadas sempre que se deseja trabalhar com padrões de texto, substituição, etc.

O código seguinte apresenta exemplo de **pesquisa**[1] ("$a =~ **m**/ola/") e de **substituição** ("**s**/ola/bom dia/" – substituição no conteúdo da variável $a de "ola" para "bom dia"). Finalmente, o operador PERL "tr" é similar ao utilitário "tr".

```
# Verifica se a string inclui a expressão ola
if ($a =~ m/ola/){ print "ola";  }

# Verifica se a string não inclui a expressão ola
if ($a !~ m/ola/){ print "sem ola";  }

# Substituir "ola" por "bom dia"
$a =~ s/ola/bom dia/;

# Alterar a letras minúsculas para maiúsculas
$a =~ tr/a-z/A-Z/;
```

O formato de uma expressão regular é 'sempre' **/expressões-regulares/**, exceto se:

^	- Usado no **início do padrão** obriga a próxima expressão a verificar-se no início da string (ou da linha se usado o modificador 'm') - Usado no **início após o meta-caracter '['** corresponde à **negação** de todas a expressões até ao meta-caracter ']', isto é, indica todos os carateres que não pertencem ao grupo definido por [...]
[]	Permite definir um conjunto de carateres, que podem ser verificados (ou não, caso se inicie por '^'). Por exemplo, **[a-z,A-E]** ou em modo de negação: [^A-Z] (tudo exceto de **A** até **Z**)
$	Usado no final do padrão obriga a expressão anterior a verificar-se no final da string (ou da linha se usado o modificador 'm')
.	Significa qualquer caracter exceto o *newline* '\n'. Inclui também o *newline* se for usado o modificador 's'
|	Permitir definir expressões alternativas que devem ser verificadas. Funciona pois como um "or".
( )	Expressões entre parênteses correspondem à captura, agrupando expressões. Após execução da operação as variáveis pré-definidas $1, $2, $.., $9 contém o valor da string verificada (capturada) no 1º,2º,..º,9º grupo respetivamente.

[1] A pesquisa numa REGEX é indicada através de "m" que corresponde à designação Anglo-Saxónica "match".

1.2.1. Carateres especiais

\ → É o caracter de escape para testar carateres que são do ponto de vista das expressões regulares meta-carateres
\s → Um *white-space*, exemplo: um espaço ou um tab
\w → Um alfanumérico incluindo o '_'
\d → Um dígito
\S → Qualquer caracter que não seja um 'white-space'
\W → Qualquer caracter que não seja um alfanumérico ou '_'
\D → Qualquer caracter que não seja um dígito

1.2.2. Quantificadores

? → Zero ou uma ocorrência
+ → Uma ou mais ocorrências
* → Zero ou mais ocorrências
{n,} → N ou mais ocorrências
{n,m} → Entre n e m ocorrências
{n} → Exatamente n ocorrências

Exemplos:

m/Si+E?s/ # Identifica um **S** seguido por um ou mais "**i**" seguido por um ou nenhum "**E**" seguido por um "**s**". (um resultado possível seria a palavra **Sistemas**)

Os quantificadores *****, **+**, e **?** são ditos "gananciosos" (*greedy*) identificando o maior número de caracteres possível:

```
$a = "INICIO xxxxxxxxx FIM";
$a =~ s/x+/XPTO/;   # devolve: INICIO XPTO FIM
```

Isto é, toda a sequência "xxxxxxxxx" foi especificada por "x+" e substituída por XPTO

Exercício 1

Recorrendo a expressões regulares, elabore a função PERL "IsNumeric1", que receba como parâmetro de entrada a expressão a validar e devolva: 1 se a expressão for um número e 0 se a expressão não for um número.

Nota: para já devem ser apenas avaliados **números inteiros positivos**, ou seja, não deve entrar para validação nem o carater "**-**", nem o caracter "+".

Exercício 2

Com base na função "IsNumeric1", escreva a função "IsNumeric2", que possa avaliar todos os números inteiros (positivos e negativos).

1.2.3. Grupo de carateres

- Para especificar, uma gama de carateres usam-se **"[ ]"** (parênteses retos)

  Exemplos

  /[abc]/

  "Verdadeiro" para qualquer string que contenha (pelo menos) um dos carateres "a" "b" "c"

  /[0123456789]/

  "Verdadeiro" para qualquer string que contenha um algarismo

- Caso se pretenda especificar **"]"** no grupo deve-se empregar a barra de escape (\), ou em alternativa colocá-lo como o primeiro caracter do grupo:

  /[abc\]]/ # inclusão de "]" na gama de carateres

  /[]abc]/ # idem

- **"-"** para especificar, gama de carateres :

  /[0123456789]/ # Qualquer dígito

  /[0-9]/ # idem

- Para especificar **"-"** na lista, coloca-se **"\"** antes ou coloca-se **"-"** no início ou no fim:

  /[X\-Z]/ # X, -, Z

  /[XZ-]/ # X, Z, -

  /[-XZ]/ # -, X, Z

- Mais alguns exemplos:

/[0-9\-]/ # 0-9, ou sinal "menos"

/[0-9a-z]/ # dígito ou letra (minúscula)

/[a-zA-Z0-9_]/ # qualquer letra, qualquer digito

- Negação de grupo (símbolo **^)** imediatamente após o parêntesis reto esquerdo. Inverte o efeito do grupo (i.e. identifica qualquer carater não presente no grupo).

  /[^0123456789]/ # tudo que não seja dígito

  /[^0-9]/ # idem

  /[^aeiouAEIOU]/ # tudo exceto vogais

  /[^\^]/ # tudo exceto o próprio ^

1.2.4. Delimitadores

Precedidas por **\b**, apenas serão encontradas as cadeias de carateres que estiverem no início de uma palavra.
Seguidas por **\b**, apenas serão encontradas as cadeias de carateres que estiverem no final de uma palavra.
Precedidas por **\B**, serão encontradas apenas as palavras que não forem iniciadas pela cadeia de carateres.
Seguidas por **\B**, serão encontradas apenas as palavras que não forem terminadas pela cadeia de carateres.

Os exemplos correspondentes são os de (32) a (35).

1.2.5. Opções

g → Global;
i → Insensível a maiúsculas ou minúsculas;
m → Interpreta carateres especiais (tipo \n);

1.2.6. Precedências

1. Precedência: **( )** (parênteses)
2. Precedência: **+ * ? {#,#}** (operadores de agrupamento)
3. Precedência: **abc ^$ \b \B** (carateres/cadeia de carateres, início ou término de linha, início ou término de palavras)
4. Precedência: **|** (alternativas)

1.2.7. Exemplos de REGEX

REGEX de pesquisa:

```
( 1) m/a/ # encontra 'a'
( 2) m/[ab]/ # encontra 'a' ou 'b'
( 3) m/[A-Z]/ # encontra todas as letras maiúsculas
( 4) m/[0-9]/ # encontra números
( 5) m/\d/ # encontra números - como em (4)
( 6) m/\D/ # encontra tudo exceto números
( 7) m/[0-9]\-/ # encontra números ou o sinal de menos
( 8) m/[\[\]]/ # encontra tudo que estiver delimitado por parênteses retos [ ]
( 9) m/[a-zA-Z0-9_]/ # encontra letras, números ou sinal de sublinhado
(10) m/[\w]/ # encontra letras, números ou sinal de sublinhado - como em (9)
(11) m/[\W]/ # encontra tudo, exceto letras, números e sinal de sublinhado
(12) m/[\r]/ # encontra o sinal de retorno (típico do DOS)
(13) m/[\n]/ # encontra o sinal para quebra de linha
(14) m/[\t]/ # encontra o sinal de tabulação (tab)
(15) m/[\f]/ # encontra o sinal para quebra de página
(16) m/[\s]/ # encontra o sinal de espaço assim como os sinais referidos de (12) a (15)
(17) m/[\S]/ # encontra tudo, exceto sinal de espaço e os de (12) a (15)
(18) m/[äöüÄÖÜ]/ # encontra todos os carateres com acentuação dupla
(19) m/[^a-zA-Z]/ # encontra tudo que não contiver letras
(20) m/[ab]/s # encontra 'a' ou 'b' também em várias linhas
(21) m/asa/ # encontra 'asa' - também 'casa' ou 'casamento'
(22) m/asa?/ # encontra 'asa', 'casa', 'casamento' e também 'as' e 'asilo'
(23) m/a./ # encontra 'as' e 'ar'
(24) m/a+/ # encontra 'a' e 'aa' e 'aaaaa' - quantos existirem
(25) m/a*/ # encontra 'a' e 'aa' e 'aaaaa' e 'b' - nenhum ou quantos 'a' existirem
(26) m/ca.a/ # encontra 'casa' e 'caça', mas não 'cansa'
(27) m/ca.+a/ # encontra 'casa', 'caça' e 'cansa'
(28) m/ca.?a/ # encontra 'casa', 'caça' e 'caso'
(29) m/x{10,20}/ # encontra sequências de 10 a 20 'x'
(30) m/x{10,}/ # encontra sequências de 10 ou mais 'x'
(31) m/x.{2}y/ # só encontra 'xxxy'
(32) m/Clara\b/ # encontra 'Clara' mas não 'Clarinha'
(33) m/\bassa/ # encontra 'assa' ou 'assado' mas não 'massa'
(34) m/\bassa\b/ # encontra 'assa' mas não 'assado' e nem 'massa'
(35) m/\bassa\B/ # encontra 'assado' mas não 'assa' e nem 'massa'
(36) m/^Julia/ # encontra 'Julia' apenas no início do contexto da pesquisa
(37) m/Helena$/ # encontra 'Helena' apenas no final do contexto da pesquisa
```

```
(38) m/^\s*$/ # encontra linhas constituídas apenas por sinais vazios ou similares
(39) m/$Nome/ # encontra o conteúdo da variável escalar $Nome
(40) m/asa/s # encontra 'asa', também em várias linhas
(41) m/a|b/ # encontra 'a' ou 'b' - idêntico a m/[ab]/
(42) m/com|sem/ # encontra 'com' e 'descompensar', como também 'sem' e 'semântica'
```

Listagem 1: Expressões regulares

REGEX de substituição:

```
#!/usr/bin/perl
use strict;
use warnings;

my $s = 'abcdefcdcd';
my $i = 'X';
my $j = 'W';
#substituição é feita a cada iteração
while ($s =~ s/CD/$i/i){
   print "1: $s\n";
   $i++;
}

#substituição é feita de uma vez só, pois é especificada a opção g
$s = 'abcdefcdcd';
while ($s =~ s/CD/$j/ig){
   print "2: $s\n";
}
```

Listagem 2: Substituições recorrendo a expressões regulares

Exercício 3

a) Elabore a função "IsNumeric3" (com base na função "IsNumeric2"), que deve avaliar, agora, todos os números reais (positivos e negativos). Faça, também, uma série de testes intensivos à função para ter a certeza que esta não tem nenhuma falha (*bug*).

Como exemplo, pode chamar a função N vezes no seu código de forma similar à indicada na listagem seguinte, em que cada string do vetor @Strings_Teste_L é um teste:

```
my @Strings_Teste_L = ("","+","+1","-1.23","5.3","5.",".8","0.6","-.5");
foreach my $Teste_S(@Strings_Teste_L){
      printf("Teste:'$Teste_S':%d\n", IsNumeric3($Teste_S));
}
```

Nota: no caso de aparecer alguma situação que esteja omissa nos testes acima indicados, decida o que a função deve devolver e implemente essa validação.

b) Repita a alínea a), mas em vez de usar o vetor **@Strings_Teste_L** faça uso de uma lista associativa (hash) denominada **%Strings_Teste_H**, em que cada chave corresponde a uma string de teste e o respetivo valor corresponde ao resultado esperado do teste.

1.2.8. Grupo e Captura

- Para especificar um grupo de valores a capturar usam-se **"( )"** (parênteses curvos). Por exemplo, se quisermos obter os valores de uma data "01-05-2015" em componentes individuais temos que conseguir separar, de forma rápida o "01", o "05" e o "2013". Isto pode ser conseguido como os **"( )"** da seguinte forma:

```
#!/usr/bin/perl -w
use strict;

my $str = "01-05-2013";

$str =~ m/^(\d\d)-(\d+)-(\d{4})$/;
print("$1 \n");   # mostra no terminal: 01
print("$2 \n");    # mostra no terminal: 05
print("$3 \n");   # mostra no terminal: 2013
```

Listagem 3: Expressões regulares

Neste exemplo são utilizadas várias expressões (complementares) para captura dos diversos grupos:

- (\d\d) – captura os primeiros dois dígitos;
- (\d+) – captura 1 ou mais dígitos até que apareça um "-", neste caso serão 2 dígitos;
- (\d{4}) – captura os últimos 4 dígitos.

Os valores capturados por estes grupos vão, respetivamente, para as seguintes variáveis especiais: **$1, $2, $3**.

Nota: não se esqueça que a variável $0 também é "especial" mas contém o nome do script que está a ser executado.

Exercício 4

Elabore a função "IsNumeric4" (com base na função "IsNumeric3"), que deve avaliar todos os números reais (positivos e negativos), discriminando, no caso de números com parte decimal, a parte inteira da parte decimal.

Exercícios

1. Crie uma script em PERL contendo o bloco de instruções que se seguem. De seguida, execute a script e interprete os resultados obtidos.

```
#!/usr/bin/perl
$\ = "\n";
$, = ",";

$s = "Isto é uma string\nEsta é uma String\nEsta é a terceira linha\n";
$s .= " Esta é a quarta linha\nEsta não é a última frase\n";
$s .= "Esta é a última linha\n";

print $s;

print "--------------\n";

$s =~ m/^Isto/ && print 'verificou-se m/^Isto/';
$s =~ m/^Esta/ || print 'não se verificou m/^Esta/';
$s =~ m/^Esta/ && print 'verificou-se m/^Esta/';

print "--------------\n";
print $s;
print "--------------\n";

@l = $s =~ m/^Esta/mg and $j = @l, print "verificou-se m/^Esta/mg $j
vezes";

print "--------------\n";

@l = $s =~ m/^[EI]st[ao]/mg and $j = @l, print "verificou-se
m/^[EI]st[ao]/mg $j vezes";

print "--------------\n";

@l = $s =~ m/^Isto|^Esta/mg and $j = @l, print "verificou-se
m/^Isto|^Esta/mg $j vezes\t", @l;

print "--------------\n";

@l = $s =~ m/^Isto|Esta/mg and $j = @l, print "verificou-se
m/^Isto|Esta/mg $j vezes\t", @l;
```

Listagem 4: Expressões regulares

2. Construa as expressões regulares que permitam validar as seguintes situações:

a. Código postal

b. Nº Telefone 91 e 96

c. endereços de email

d. Endereços IPv4

e. URL registados no domínio ".pt"

3. Recorrendo somente à linguagem PERL e às expressões regulares, elabore o script "**eitube_num_views.pl**", cujo propósito é o de reportar o número de visionamentos ("views") de cada vídeo constante do canal YouTube do curso de Eng. Informática da ESTG (http://www.youtube.com/cursoeiestg), sendo a informação precedida da data corrente. Por exemplo, quando executado o script produzirá a seguinte saída:

```
20150515_15h25:9
30 views
120 views
96 views
105 views
1599 views
201 views
682 views
746 views
5432 views
```

Nota: Para testar poderá efetuar o *download* da página inicial do site da ESTG, através do comando:

$ wget http://www.youtube.com/cursoeiestg **-O** eitube.html

4. Recorrendo à linguagem PERL e às expressões regulares, elabore o script **"encontra_imagens.pl",** que encontre e efetue a contagem de quantas imagens existem numa página HTML. O script recebe por parâmetro o ficheiro onde se deve procurar as imagens:

$ ./encontra_imagens.pl index.html

As imagens no HTML são identificadas pela TAG <*img*>.

Nota: Para testar, poderá efetuar o *download* da página inicial do site da ESTG através do comando:

$ wget http://www.estg.ipleiria.pt -O index.html

5. Recorrendo à linguagem PERL e às expressões regulares, elabore a script **"extensoes.pl"** que dada uma diretoria de procura, efetue a listagem **apenas dos ficheiros** existentes nessa diretoria, especificando a extensão de cada um deles. A diretoria será indicada por parâmetro, na chamada do script:

$ ./extensoes.pl $HOME txt

Note que, um ficheiro com o nome **a.bb.cc.zip**, deverá ser separado com o nome **a.bb.cc,** e com extensão **zip**.

O resultado final terá um aspeto similar a este:

```
Ficheiro          ->   Nome         -> Extensão
exercicio1.pl~    ->   exercicio1   -> pl~
exercicio2.pl~    ->   exercicio2   -> pl~
extensoes.pl      ->   extensoes    -> pl
extensoes.pl~     ->   extensoes    -> pl~
index.html        ->   index        -> html
```

6. Recorrendo à linguagem PERL e às expressões regulares, elabore a script **"devolveip.pl"** que deve devolver a lista de IPs encontrada no comando "`netstat -na`", e contar quantas vezes cada um se repete.

7. Elabore, com recurso à linguagem PERL, a função "GetExtension($)", que deve devolver uma string contendo a extensão de um nome de ficheiro que lhe seja passado como argumento. Por extensão, entende-se o texto que surge após o último ponto. Por exemplo, no caso do nome "*a.txt*", deverá ser devolvido "*txt*", ao passo que para o nome "*a.b.c.zip*" a função deverá devolver "*zip*".

8. Elabore a função PERL SplitPath($) que recebendo um caminho de ficheiro (seja ele absoluto ou relativo) deve devolver duas strings: uma referente aos diretórios e a segunda referente ao nome do ficheiro (caso exista). Por exemplo, dado o caminho "`/root/test/fich.txt`", a função deve devolver como *parte diretório* "`/root/test/`" e como parte ficheiro "*`fich.txt`*".

9. Elabore o script "`get_filename_parts.pl`" onde devem ser apropriadamente testadas as funções desenvolvidas nas alíneas anteriores.

10. No Linux, o ficheiro `/proc/cpuinfo` apresenta a informação resumida do(s) CPU(s) da máquina local. Uma vista parcial do conteúdo do referido `/proc/cpuinfo` está mostrado na listagem abaixo:

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 8
```

Elabore o script "parse_cpuinfo.pl", cujo propósito é o de decompor o conteúdo do ficheiro `/proc/cpuinfo` numa lista associativa (vulgo *hash*), sendo as chaves da lista associativa formadas pelos campos da coluna da esquerda do ficheiro. O script deve mostrar na saída padrão o conteúdo da lista associativa assim formada, de forma similar a abaixo mostrada:

```
'bogomips':  1862.20
'cache size':  256 KB
'coma_bug':  no
'cpu MHz':  930.357
'cpu family':  6
'cpuid level':  2
'f00f_bug':  no
'fdiv_bug':  no
```

11. O Self-Monitoring, Analysis, and Reporting Technology (SMART) é um conjunto de parâmetros disponíveis nos discos duros com o objetivo de indicar a vitalidade do disco duro. O SMART contém informação tão diversa como o "Load/Unload Cycle Count" (número de "parques" dinâmicos da cabeça do disco), o "Power On Hours Count" (número de horas de funcionamento do disco) e o "temperature" (temperatura, usualmente em graus Farenheit). Para mais informações a respeito do SMART pode consultar o Wikipedia (http://en.wikipedia.org/wiki/S.M.A.R.T.).

No Windows, existem vários utilitários para consulta dos parâmetros de SMART do(s) disco(s) das máquinas locais. Para efeitos de automação via scripting em PERL, estamos interessados na versão da BeyondLogic (consultar http://www.beyondlogic.org/solutions/smart/smart.htm) dado esse utilitário correr em modo consola. Uma alternativa ao "smart.exe" é o "smartmontools", que beneficia ainda do facto de ser disponibilizado sob licença Código Aberto e para plataforma Linux e Windows (http://smartmontools.sourceforge.net/).

O resultado da execução do "smart.exe" no disco de uma máquina devolveu a seguinte informação:

```
ID   Attribute                 Type  Threshold Value Worst        Raw Status
---- ------------------------- ----- --------- ----- ----- -------- --------
[01] Raw Read Error Rate        Prefailure   62   100   100          0 OK
[02] Throughput Performance     Prefailure   40   100   100          0 OK
[03] Spin Up Time               Prefailure   33   230   230          1 OK
[04] Start/Stop Count           Advisory      0   100   100        383 OK
[05] Reallocated Setor Count   Prefailure    5   100   100          0 OK
[07] Seek Error Rate            Prefailure   67   100   100          0 OK
[09] Power On Hours Count       Advisory      0    97    97       1650 OK
[0A] Spin Retry Count           Prefailure   60   100   100          0 OK
[0C] Power Cycle Count          Advisory      0   100   100        381 OK
[BF] Unknown SMART Attribute    Advisory      0   100   100          0 OK
[C0] Power-Off Park Count       Advisory      0   100   100          7 OK
[C1] Load/Unload Cycle Count    Advisory      0    89    89     119596 OK
[C2] Drive Temperature          Advisory      0   127   127    1048619 OK
[C4] Re-Allocated Data Count    Advisory      0   100   100          0 OK
[C5] Pending Setor Count       Advisory      0   100   100          0 OK
```

```
[C7] CRC Error Count            Advisory      0   200   253          0 OK
[DF] Unknown SMART Attribute    Advisory      0   100   100          0 OK
```

Tenha em atenção que outros discos (os parâmetros SMART variam entre fabricantes e também entre modelos de discos) poderão devolver mais ou menos campos.

a) Elabore, em ambiente WINDOWS, e recorrendo à linguagem PERL, o script **parseSmart_V1.pl**, cujo propósito é o de criar uma hash contendo todos os dados relevantes de SMART devolvido pelo utilitário smart.exe. A chave dos campos da *hash* deve ser os código hexadecimais que surgem entre parênteses retos. O valor a guardar na *hash* será o elemento da coluna "**Raw**". No final, o script deve apresentar na saída padrão o conteúdo (chave e valores da *hash*), ordenado pela chave.

b) Elabore **o script parseSmart_V2.pl**, cujo propósito é o de criar uma *hash* contendo todos os dados relevantes de SMART devolvidos pelo utilitário smart.exe. Contudo, os campos a empregar como chave da *hash* deverão ser as *strings* da coluna "Attribute". No final, o script deve apresentar na saída padrão o conteúdo (chave e valores da *hash*), ordenado pela chave.

c) Com base no script "**parseSmart_V1.pl**", elabore o script "**saveSmart.pl**" cujo objetivo é o de gravar cumulativamente no ficheiro "smart.dat", uma linha descrevendo os seguintes parâmetros do SMART:

```
[04] Start/Stop Count
[09] Power On Hours Count
[0C] Power Cycle Count
[C0] Power-Off Park Count
[C2] Drive Temperature
```

O formato do ficheiro deve ser o seguinte:

```
DATA:DATA_STR[04]:[09]:[0C]:[C0]:[C2]
```

Sendo a DATA expressa no número de segundos desde o Unix Epochs da data/hora corrente e DATA_STR uma representação em *string* dessa mesma data/hora. O formato da *string* deverá ser "YYYYMMDD_horaHMin:s" (e.g. "20071121_15h54:13"). O exemplo de uma linha a criar será:

```
# DATA:DATA_STR:[04]:[09]:[0C]:[C0]:[C2]
1195660505:"20071121_15h55:07":383:1650:7:1048619
```

NOTA: caso o "smart.exe" não funcione, instale o "smartmontools" (http://smartmontools.sourceforge.net/) no Windows.

2. Documentação PERL

Existe documentação PERL acessível na web. Como sítio de referência, sugere-se www.perl.org (e em particular perldoc.perl.org).

Em termos de bibliografia, recomenda-se o livro "*Learning PERL*" da O´Reilly (pesquise "learning perl" num motor de busca web), o mais avançado "*Programming PERL*" também da editora O'Reilly e por fim, o livro para especialistas em REGEX, o "*Mastering Regular Expressions*" de Jeffrey Friedl.

3. Bibliografia

- "Beginning PERL"
  http://www.perl.org/books/beginning-perl/ (maio 2013)

- "Mastering Regular Expressions", 3rd edition, Jeffrey Friedl, agosto 2006
  http://oreilly.com/catalog/9780596528126/

- "Perl 5 Tutorial"
  http://www.cbkihong.com/download/perltut.pdf (maio 2013)

- Advanced PERL quick reference guide
  http://refcards.com/docs/trusketti/perl-regexp/perl-regexp-refcard-a4.pdf