

Tópicos Especiais em Fundamentos da Computação Shell Scripting

Expressões Regulares

Andrei Rimsa Álvares
andrei@cefetmg.br



Sumário

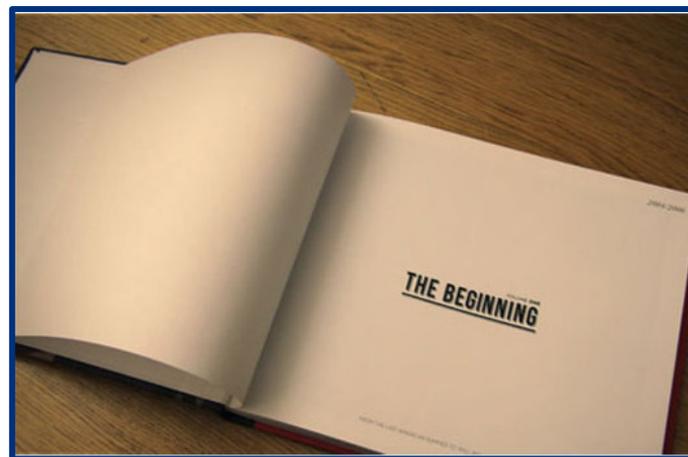
- Introdução
- `grep`
- Expressões regulares
- Regra
- Substituição de texto
- Expressões regulares estendidas



CEFET-MG

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

INTRODUÇÃO



Shell Scripting



Introdução

- De forma simples, expressões regulares são notações usadas para identificar padrões em textos
 - De certa forma, parecem com o sistema de caracteres coringas de expansão de arquivos, mas em uma escala bem maior
- Expressões regulares definem um conjunto de uma ou mais strings
 - Uma string simples é uma expressão regular que define uma string: ela mesma
 - Uma expressão regular complexa usa letras, números e caracteres especiais para definir muitas diferentes strings

Uma expressão regular casa com qualquer string que ela define



Onde Usar Expressões Regulares?

- Expressões regulares são suportadas por vários programas em linha de comando e por muitas linguagens de programação para facilitar a resolução de problemas de manipulação de textos



sed



vim



ed



awk



grep



perl

Cuidado: nem todas expressões regulares funcionam da mesma forma, podem variar de ferramentas e LPs



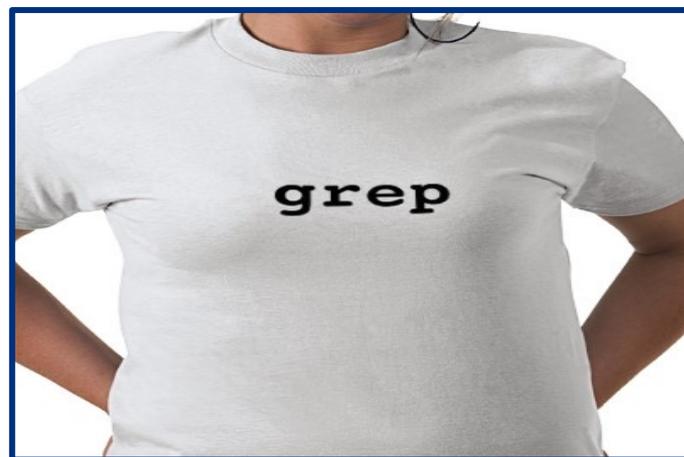
emacs



CEFET-MG

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

GREP



Shell Scripting



grep

- O programa principal que será usado para demonstrar o uso das expressões regulares será o `grep`♠
 - Em essência, esse programa procura em arquivos de texto pela ocorrência de expressões regulares especificadas e imprime qualquer linha que casa na saída padrão
- Até agora, `grep` foi usado somente para buscar *strings* fixas

```
$ ls /usr/bin | grep zip
bunzip2
bzip2
gunzip
gzip
unzip
...
```

♠: *global regular expression print*



grep

- `grep` possui a seguinte sintaxe

```
grep [options] regex [file...]
```

, onde *regex* é uma expressão regular

- A tabela a seguir mostra algumas opções comumente utilizadas

| Opção | Descrição |
|-------|--------------------------------------------------------------------------|
| -i | Ignora caixa, não distingue entre caracteres maiúsculos e minúsculos |
| -v | Inverter o casamento, mostra todas as linhas que não contém o casamento |
| -c | Mostra o número de casamentos (ou não-casamentos no caso de -v) |
| -l | Mostra os nomes do arquivos que contém o casamento, ao invés das linhas |
| -L | Similar a opção -l, mas que mostra os arquivos que não contém casamentos |
| -n | Prefixa cada linha casada com o número da linha |
| -h | Para pesquisa com múltiplos arquivos, suprime o nome do arquivo na saída |



Exemplos

- Antes de mostrar exemplos, criar vários arquivos com dados

```
$ ls /bin > dirlist-bin.txt
$ ls /usr/bin > dirlist-usr-bin.txt
$ ls /sbin > dirlist-sbin.txt
$ ls /usr/sbin > dirlist-usr-sbin.txt
$ ls dirlist*.txt
dirlist-bin.txt          dirlist-usr-bin.txt
dirlist-sbin.txt        dirlist-usr-sbin.txt
```

- Exemplos

```
$ grep bzip dirlist*.txt
dirlist-usr-bin.txt:bzip2
dirlist-usr-bin.txt:bzip2recover
```

```
$ grep -l bzip dirlist*.txt
dirlist-usr-bin.txt
```

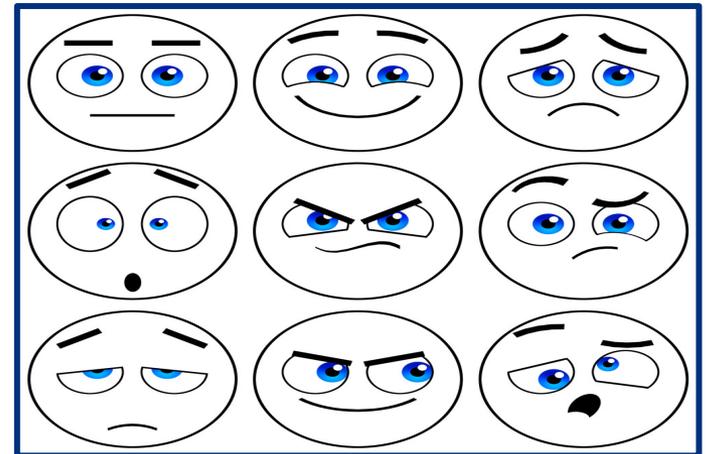
```
$ grep -L bzip dirlist*.txt
dirlist-bin.txt
dirlist-sbin.txt
dirlist-usr-sbin.txt
```



CEFET-MG

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

EXPRESSÕES REGULARES



Shell Scripting



Caracteres

- Um caractere é considerado qualquer um que não seja nova linha ($\backslash n$)
 - **Caracteres literais:** caracteres que se representam (casam eles próprios)
 - **Caracteres especiais (*metacaracteres*):** caracteres que são usados para representar casamentos mais complexos; são eles
 - Expressão regular básica: \wedge $\$$ \cdot $[$ $]$ $*$ \backslash
 - Expressão regular estendida: $?$ $+$ $($ $)$ $\{$ $\}$ $|$

Dica: se precisar usar um caractere especial para representar a si próprio, deve-se escapá-lo



Delimitadores

- Um caractere chamado delimitador usualmente marca o início e o fim de uma expressão regular
- O delimitador é sempre um caractere especial para a expressão regular que ele delimita (não representa a si próprio)
- Algumas ferramentas (como vim) permitem o uso de outros caracteres como delimitadores, enquanto `grep` não usa delimitador nenhum
 - Aqui será usado a barra normal (/) como delimitador
- Em alguns casos não ambíguos, o segundo delimitador não é necessário; pode ser omitido se for imediatamente seguido por RETURN



Strings Simples

- A expressão regular mais básica é uma *string* simples que não contém nenhum caractere especial, a não ser os delimitadores
 - Uma *string* simples casa somente ela mesmo

| Expressão Regular | Casamento | Exemplos |
|-------------------|-----------------|-----------------------------------------------------------------|
| /ring/ | ring | <u>ring</u> , <u>spring</u> , <u>ringing</u> , <u>stringing</u> |
| /Thursday/ | Thursday | <u>Thursday</u> , <u>Thursday's</u> |
| /or not/ | or not | <u>or not</u> , <u>poor nothing</u> |



Caracteres Especiais

- Os caracteres especiais são
 - **Ponto (.)**: o ponto casa qualquer (um) caractere
 - **Colchetes ([])**: os colchetes definem uma classe de caracteres, que casa qualquer caractere único dessa classe; **^** casa qualquer caractere não especificado na classe, enquanto **-** pode definir uma faixa de caracteres
 - **Asterisco (*)**: o asterisco representa zero ou mais ocorrências de um casamento de expressão regular
 - **Âncoras (^ e \$)**: uma expressão regular que começa com **^** casa apenas strings no começo da linha, enquanto **\$** casa no final
 - **Escapes (\)**: pode-se escapar qualquer caractere especial (mas não parênteses ou dígitos) colocando uma barra invertida antes



Ponto (.)

- Um ponto casa qualquer (um) caractere

| Expressão Regular | Casamento | Exemplos |
|-------------------|-----------------------------------|----------------------------------------------------------|
| / .alk/ | espaço + qualquer caractere + alk | will <u>talk</u> , may <u>balk</u> |
| / .ing/ | qualquer caractere + ing | <u>sing</u> song, <u>ping</u> , before <u>ing</u> lenook |

- Exemplo

```
$ grep -h '.zip' dirlist*.txt
bunzip2
bzip2
bzip2recover
funzip
gunzip
gzip
unzip
unzipsfx
```



Colchetes ([])

- Define uma classe de caracteres, que casa qualquer caractere único dessa classe
 - Circunflexo (^) casa qualquer caractere não especificado na classe
 - Hífen (–) pode definir uma faixa de caracteres

| Expressão Regular | Casamento | Exemplos |
|-------------------|------------------------------------------------|-------------------------------------------------------------------------------------------------|
| /[bB]ill/ | b ou B + ill | <u>bill</u> , <u>Bill</u> , <u>billed</u> |
| /t[aeiou].k/ | t + qualquer vogal + qualquer caractere + k | <u>talkative</u> , <u>stink</u> , <u>teak</u> , <u>tanker</u> |
| /# [6-9]/ | # seguido de espaço + dígitos entre 6 e 9 | <u># 6 0</u> , <u># 8</u> , get <u># 9</u> |
| /[^a-zA-Z]/ | Qualquer caractere que não é uma letra | <u>1</u> , <u>7</u> , <u>@</u> , <u>,</u> , <u>}</u> , Stop! |

Cuidado: barra invertida (\) e asterisco (*)
perdem seu significado especial entre colchetes



Colchetes ([])

- Exemplos

- b ou g seguidos de zip

```
$ grep -h '[bg]zip' dirlist*.txt  
bzip2  
bzip2recover  
gzip
```

- Qualquer caractere diferente de b ou g seguidos de zip

```
$ grep -h '[^bg]zip' dirlist*.txt  
bunzip2  
funzip  
gunzip  
unzip  
unzipsfx
```

- Contém uma letra maiúscula

```
$ grep -h '[A-Z]' dirlist*.txt  
BuildStrings  
CpMac  
DeRez  
GetFileInfo  
HsColour  
...
```

O que
[-AZ] faz?



Colchetes ([])

- Classes de caracteres pré-definidas

| Classe | Significado |
|--------------|-----------------------------------------------------------------------------------------------------|
| [:alnum:] | Caracteres alfanuméricos: letras e dígitos |
| [:alpha:] | Caracteres do alfabeto: letras |
| [:blank:] | Caracteres branco: espaço em branco e tabulação |
| [:cntrl:] | Caracteres de controle (CONTROL) |
| [:digit:] | Caracteres numéricos: dígitos |
| [:graph:] | Caracteres gráficos: [:alnum:] e [:punct:] |
| [:lower:] | Caracteres minúsculos do alfabeto: [a-z] |
| [:print:] | Caracteres imprimíveis: [:alnum:], [:punct:] e [:space:] |
| [:space:] | Caracteres de espaçamento: espaço, tabulação, nova linha, <i>form feed</i> e <i>carriage return</i> |
| [:upper:] | Caracteres maiúsculos do alfabeto: [A-Z] |
| [:xdigit:] | Dígitos hexadecimais: [0-9], [a-f] e [A-F] |



Asterisco (*)

- O asterisco representa zero ou mais ocorrências da expressão regular **precedente**
 - Asterisco depois de um caractere literal indica zero ou mais sequências desse caractere
 - Asterisco depois de ponto (.) casa qualquer sequência de caracteres
 - Asterisco depois de uma classe de caracteres casa qualquer string cujos caracteres são membros da classe



Asterisco (*)

| Expressão Regular | Casamento | Exemplos |
|---------------------------|-----------------------------------------------------------------|-----------------------------------------------------------------------|
| <code>/ab*c/</code> | a + zero ou mais b's + c | <u>ac</u> , <u>abc</u> , <u>abbc</u> , debbca <u>abbbc</u> |
| <code>/ab.*c/</code> | ab + qualquer sequência de caracteres + c | <u>abc</u> , <u>abxc</u> , <u>ab45c</u> , <u>xab 765.345 x cat</u> |
| <code>/t.*ing/</code> | t + qualquer sequência de caracteres + ing | <u>thing</u> , <u>ting</u> , I <u>thought of going</u> |
| <code>/[a-zA-Z]*/</code> | Strings compostas por letras (maiúsculas e minúsculas) e espaço | 1. <u>any string without numbers or punctuation!</u> |
| <code>/(.*)/</code> | A maior string entre (e) | Get <u>(this) and (that)</u> ; |
| <code>/([^]*)/</code> | A menor string entre (e) | <u>(this)</u> , Get <u>(this and that)</u> |

- Exemplo

```
$ cat dirlist-* | grep "uu.*e"
```

```
uudecode
```

```
uuencode
```

```
uuidgen
```

```
uuname
```

```
uusched
```



Âncoras (^ e \$)

- Uma expressão regular que começa com um circunflexo (^) casa apenas strings no começo da linha; já o sinal de dólar (\$) no final da expressão regular casa no final da linha

| Expressão Regular | Casamento | Exemplos |
|-------------------|--------------------------------------------------------------|-----------------------------------------------------|
| /^T/ | Um T no começo da linha | <u>T</u> his line, <u>T</u> hat time, In Time |
| /^+[0-9]/ | Um sinal de mais (+) seguido de um dígito no começo da linha | <u>+</u> 5 +45.72, <u>+</u> 759 Keep this... |
| /:\$/ | Um dois pontos no final da linha | ...below: <u>:</u> |

Curiosidade: ^ e \$ são chamadas de âncoras porque forçam (ancoram) um casamento no começo ou final de uma linha



Âncoras (^ e \$)

- Exemplos

- Programas que começam com zip

```
$ grep -h '^zip' dirlist*.txt
```

```
zip  
zipcloak  
zipdetails  
zipgrep  
zipinfo  
zipnote  
zipsplit
```

- Programas que terminam com zip

```
$ grep -h 'zip$' dirlist*.txt
```

```
funzip  
gunzip  
gzip  
unzip  
zip
```

- Programa com nome exato de zip

```
$ grep -h '^zip$' dirlist*.txt
```

```
zip
```



Escapes (\)

- Pode-se escapar qualquer caractere especial (mas não parênteses ou dígitos) colocando uma barra invertida antes do caractere

| Expressão Regular | Casamento | Exemplos |
|-------------------|---------------------------|-----------------------------------------------------------|
| /end\./ | end + ponto | The <u>end.</u> , <u>send.</u> , pret <u>end</u> .mail |
| \/ | Uma única barra invertida | \ |
| \/ | Um asterisco (*) | *.c, um asterisco (*) |
| \/ | [5] | it was five <u>[5]</u> |
| /and\/or/ | and/or | <u>and/or</u> |

- Exemplo

```
$ grep -h '\[' dirlist*.txt  
[
```



Agrupamento de Expressões Regulares

- Pode-se usar parênteses escapados, `\(` e `\)`, para agrupar uma expressão regular – assim, a string que essa expressão regular casa pode ser recuperada posteriormente
 - A expressão regular não tenta casar o parêntese escapado
- Uma expressão regular entre parênteses escapados casa exatamente as mesmas strings que a expressão regular sem os parênteses casa
 - `/a\(b*\)` casa o que `/ab*c/` casa
- Ainda se pode aninhar agrupamentos, como na expressão `\([a-z]\([A-Z]*\)` que possui um agrupamento dentro de outro
 - Para a string **3 t dMNORx7 l u**, a expressão regular casa **dMNORx**, onde o primeiro agrupamento casa **dMNORx** e o segundo **MNOR**

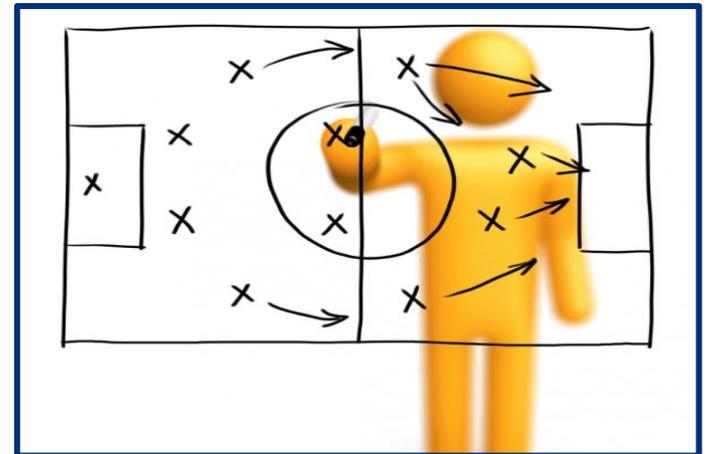
Para quê
serve isso?



CEFET-MG

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

REGRA



Shell Scripting



Casamento da Maior Sequência Possível

- **Regra:** uma expressão regular SEMPRE casa a string mais longa possível, começando no mais próximo do começo da linha
- Exemplos
 - *This (rug) is not what it once was (a long time ago), is it?*
 - /Th.*is/
This (rug) is not what it once was (a long time ago), is it?
 - /(.*)/
This (rug) is not what it once was (a long time ago), is it?
 - *singing songs, singing more and more*
 - /s.*ing/
singing songs, singing more and more
 - /s.*ing song/
singing songs, singing more and more

SUBSTITUIÇÃO DE TEXTO





Substituição de Texto

- Os editores de texto `vim` e `sed` usam expressões regulares como *strings* de busca em comandos de substituição
- Pode-se usar os caracteres especiais E comercial (&) ou dígitos escapados (`\1`, `\2`, ...) para representar as strings casadas na string de substituição correspondente



E Comercial (&)

- Em uma string de substituição, um E comercial (&) recebe o valor da string que a string de pesquisa (expressão regular) casou
- Por exemplo, a expressão a seguir envolve o nome da shell entre dois sublinhados (__SHELL__)

```
$ echo $SHELL
/bin/bash
$ echo $SHELL | grep "[^/]*sh$"
/bin/bash
$ echo $SHELL | sed "s:[^/]*sh$:__&__:"
/bin/__bash__
```



Dígito Escapado (\n)

- Dentro da *string* de substituição, um dígito escapado (\n) representa a *string* que a expressão regular agrupada (aquela entre parênteses escapados), começando com a n-ésima \n casada
- Exemplo de uma lista de nomes no formato **last-name, first-name initials** e se deseja passar para o formato **first-name initials last-name**

```
$ cat megadeth.txt
```

```
Mustaine, David S.
```

```
Ellefson, David W.
```

```
Friedman, Martin A.
```

```
Menza, Nick
```

```
$ cat megadeth.txt | sed "s/\([^\,]*\) , \(.*\) / \1 /"
```

```
David S. Mustaine
```

```
David W. Ellefson
```

```
Martin A. Friedman
```

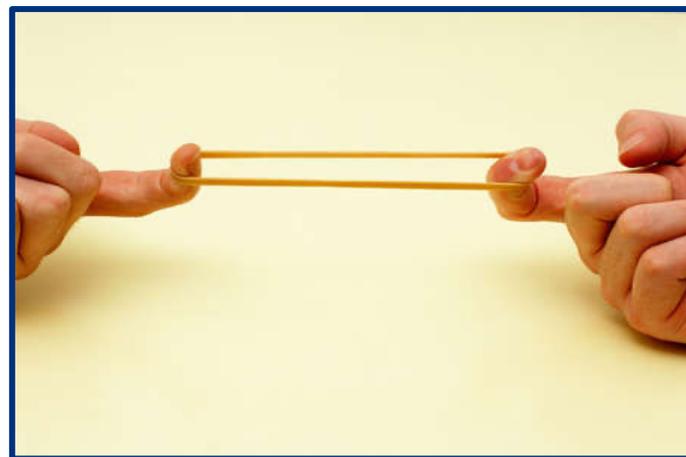
```
Nick Menza
```



CEFET-MG

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

EXPRESSÕES REGULARES ESTENDIDAS



Shell Scripting



Padronização POSIX

- A padronização POSIX separa a implementação de expressões regulares em dois tipos
 - Expressões regulares básicas (ERB)
 - Expressões regulares estendidas (ERE)
- Com ERB os metacaracteres `$ ^ . [] *` são reconhecidos, todos os outros são considerados literais; ERE adicionam os metacaracteres `() { } ? + |` com suas funções associadas
 - Interessante notar que os caracteres `() { }` são tratados como metacaracteres no ERB quando escapados, enquanto em ERE qualquer caractere escapado é considerado literal
- Para usar expressões regulares estendidas pode-se usar a ferramenta `egrep` ou `grep` com a opção `-E`



Expressão Regular Estendida

- Expressões regulares estendidas possuem duas funcionalidades
 - **Alternadores:** permitem casamento em um conjunto de expressões (ao invés de apenas uma)
 - **Quantificadores:** permitem especificar a quantidade de vezes que um elemento é casado
 - `?`: casa um elemento zero vezes ou uma vez
 - `*`: casa um elemento zero ou mais vezes
 - `+`: casa um elemento uma ou mais vezes
 - `{ }`: casa um elemento um número específico de vezes



Alternadores

- Alternadores permitem casamento em um conjunto de expressões, casando uma delas
- Exemplo
 - Expressão regular AAA|BBB permite casar AAA ou BBB

```
$ echo "AAA" | grep -E "AAA|BBB"  
AAA  
$ echo "BBB" | grep -E "AAA|BBB"  
BBB  
$ echo "CCC" | grep -E "AAA|BBB"  
$
```

- Não está limitada a apenas duas expressões

```
$ echo "AAA" | grep -E "AAA|BBB|CCC"  
AAA
```



Alternadores

- Para combinar alternadores com outras expressões regulares, pode-se usar parênteses agrupando as expressões
- Exemplo
 - Todos os arquivos que começam com bz, gz ou zip

```
$ grep -Eh '^(bz|gz|zip)' dirlist*.txt  
bzcat  
bzcmp  
bzdiff  
bzip2  
bzless  
bzmore  
gzcat  
gzexe  
gzip  
zip  
zipgrep  
zipinfo
```

O que aconteceria se os parênteses fossem retirados?



Quantificador ?: Zero ou Uma Vez

- O efeito do quantificador ? é tornar o "elemento precedente opcional"
- Exemplo: verificar se um número de telefone com DDD entre os parênteses, *(dd) dddd-dddd*, ou sem parênteses *dd dddd-dddd*

```
$ echo "(31) 1234-5678" | grep -E '^\(?[0-9][0-9]\)?  
> [0-9][0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]$'  
(31) 1234-5678  
$ echo "31 1234-5678" | grep -E '^\(?[0-9][0-9]\)?  
> [0-9][0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]$'  
31 1234-5678  
$ echo "AA 1234-5678" | grep -E '^\(?[0-9][0-9]\)?  
> [0-9][0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]$'  
$
```



Quantificador *: Zero ou Mais Vezes

- Como o metacaractere ?, o * é usado para denotar um item opcional: mas, diferentemente de ?, o item pode ocorrer qualquer número de vezes (zero ou mais)
- Exemplo: verificar se é uma sentença, começa com letra maiúscula e possui qualquer sequência de letras maiúsculas e minúsculas e termina com ponto final

```
$ echo "This works." | grep -E '[[[:upper:]][[[:upper:]][:lower:]] ]*\.'
```

This works.

```
$ echo "This Works." | grep -E '[[[:upper:]][[[:upper:]][:lower:]] ]*\.'
```

This Works.

```
$ echo "this does not" | grep -E '[[[:upper:]][[[:upper:]][:lower:]] ]*\.'
```

\$



Quantificador +: Uma ou Mais Vezes

- O metacaractere + funciona de forma bastante similar ao *, mas necessita pelo menos uma instância do elemento para um casamento
- Exemplo: apenas linhas que possuem grupos de um ou mais caracteres alfabéticos separados por um único espaço

```
$ echo "This that" | grep -E '^([[alpha:]]+ ?)+$'  
This that  
$ echo "a b c" | grep -E '^([[alpha:]]+ ?)+$'  
a b c  
$ echo "a b 9" | grep -E '^([[alpha:]]+ ?)+$'  
$ echo "abc d" | grep -E '^([[alpha:]]+ ?)+$'  
$
```



Quantificador {}: Número Específico de Vezes

- Os metacaracteres { e } são usados para expressar o número mínimo e máximo de casamentos necessários

| Especificador | Descrição |
|---------------|-----------------------------------------------------------------------------------|
| {n} | Casa o elemento precedente se ocorrer exatamente n vezes |
| {n,m} | Casa o elemento precedente se ocorrer no mínimo n vezes, mas não mais que m vezes |
| {n,} | Casa o elemento precedente se ocorrer no mínimo n ou mais vezes |
| {,m} | Casa o elemento precedente se ocorrer não mais que m vezes |

- Exemplo: do formato do telefone

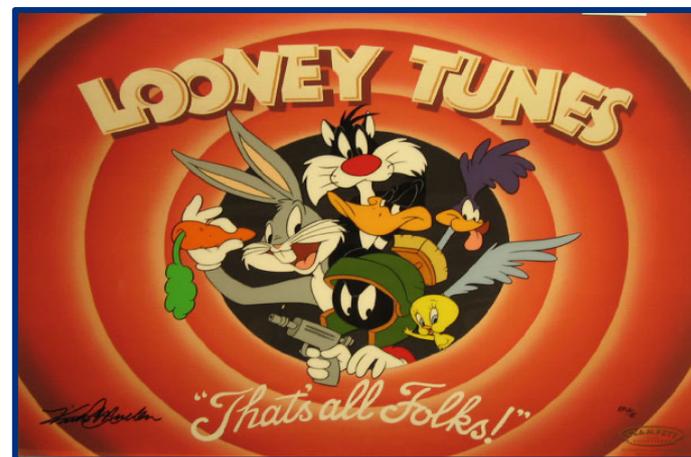
```
$ echo "(31) 1234-5678" | grep -E '^\(?[0-9]{2}\)? [0-9]{4}-[0-9]{4}$'
(31) 1234-5678
$ echo "31 1234-5678" | grep -E '^\(?[0-9]{2}\)? [0-9]{4}-[0-9]{4}$'
31 1234-5678
$ echo "031 1234-5678" | grep -E '^\(?[0-9]{2}\)? [0-9]{4}-[0-9]{4}$'
$
```



CEFET-MG

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

ISSO É TUDO, PESSOAL!



Shell Scripting