Tópicos Especiais em Fundamentos da Computação
Shell Scripting

Expansões

Andrei Rimsa Álvares
andrei@cefetmg.br

Sumário

- Expansão de arquivos
- Expansão til (~)
- Expansão aritmética
- Expansão de chaves
- Expansão de parâmetros
- Substituição de comandos
- Escapes

EXPANSÃO DE ARQUIVOS

Shell Scripting

Caracteres Coringa

- Quando se passa para a *shell* nomes de arquivos abreviados que contém caracteres especiais (metacaracteres), ela pode gerar nomes de arquivos que casam com o nome de arquivos existentes

- Esses caracteres especiais são chamados de **caracteres coringa**, já que se comportam como cartas coringa de um baralho

- Quando esses caracteres aparecem em um argumento em linha de comando, a *shell* o **expande** em uma lista de nomes de arquivos

Esse processo é chamado de expansão de caminho de arquivos ou ***globbing***

Caracteres Coringa

- Os caracteres coringa com seus respectivos significados são descritos na tabela a seguir

Classe	Significado
`?`	Qualquer caractere único
`*`	Quaisquer sequências de caracteres
`[caracteres]`	Qualquer caractere único que é membro desse conjunto de caracteres
`[!caracteres]` `[^caracteres]`	Qualquer caractere único que não é membro desse conjunto de caracteres
`[[:class:]]`	Qualquer caractere único que é membro da classe especificada

O Caractere Coringa: ?

- O caractere coringa ? (ponto de interrogação) casa **qualquer caractere único** no nome de arquivos existentes

```
$ lpr memo?
```

  - A *shell* expande o argumento `memo?` e gera uma lista de arquivos no diretório de trabalho que possui nomes compostos de `memo` seguido de qualquer caractere único
  - Se nenhum arquivo casar com o padrão a *shell* então não expande o argumento e passa a própria *string* (`memo?`) para o comando (é possível configurar para expandir vazio usando `nullglob`)

Os nomes de arquivos são expandidos como se fossem digitados um a um na linha de comando

O Caractere Coringa: ?

- O exemplo a seguir usa o comando `ls` para mostrar todos os arquivos no diretório de trabalho e depois aqueles que casam com `memo?`

```
$ ls
mem     memo12 memo9  memomax    newmemo5
memo    memo5  memoa  memos
$ ls memo?
memo5 memo9 memoa memos
```

Dica: use `ls` e `echo` para praticar a geração de nomes de arquivos

- Pode-se usar o caractere coringa `?` em qualquer posição no nome do arquivo, inclusive no meio

```
$ ls
7may4report   may4report    mayqreport    may_report
May14report   may4report.79    mayreport     may.report
$ echo may?report
may.report may4report may_report mayqreport
```

O Caractere Coringa: *

- O caractere coringa * (asterisco) realiza operação similar ao do ponto de interrogação, mas, ao invés de casar um único caractere, pode casar qualquer quantidade de caracteres (inclusive nenhum)

- Exemplo

```
$ ls
amemo  memo.0612     memosally     memsam    user.memo
mem    memoa         memosam.0620  sallymemo
memo   memorandum    memosam.keep  typescript
$ echo memo*
memo       memo.0612     memoa          memorandum
memosally memosam.0620  memosam.keep
$ echo *mo
amemo memo sallymemo user.memo
$ echo *sam*
memosam.0620 memosam.keep memsam
```

Cuidado

- Para exibir os arquivos escondidos (que começam com ponto), pode-se usar a opção `-a` no comando `ls`

```
$ ls
aaa         memo.sallysally.0612thurs
memo.0612   report          saturday
$ ls -a
.    .aaa        aaa             memo.sallysally.0612thurs
..   .profile    memo.0612       report          saturday
```

- Tanto o caractere coringa `?` quanto o caractere `*` respeitam esse comportamento de não casar arquivos escondidos

```
$ echo *
aaa memo.0612 memo.sally report sally.0612 saturday thurs
```

- Para exibir os arquivos escondidos, deve-se colocar explicitamente o caractere ponto antes do padrão

```
$ echo .*
.   ..  .aaa   .profile
```

Os Caracteres Coringa: `[]`

- Um par de colchetes envolvendo uma lista de caracteres faz com que a *shell* case nomes de arquivos contendo um dos caracteres individualmente
  - Enquanto `memo?` casa `memo` seguido de qualquer caractere, `memo[17a]` é mais restritivo, casa apenas `memo1`, `memo7` e `memoa` (caso esses arquivos existam)

- A *shell* expande o argumento que inclui a definição com colchetes, substituindo cada membro da classe de caracteres, um de cada vez

- Exemplos

```
$ echo [aeiou]*
...
```

```
$ less page[2468].txt
...
```

Os Caracteres Coringa: `[]`

- Um **hífen** dentro de colchetes define uma faixa de caracteres dentro de uma definição de classes
  - `[6-9]`: representa **`[6789]`**
  - `[a-z]`: todas as letras minúsculas (sem acentos)
  - `[a-zA-Z]`: todas as letras minúsculas e maiúsculas (sem acentos)

- Os seguintes comandos mostram três formas diferentes de imprimir os arquivos: `part0`, `part1`, `part2`, `part3` e `part5`

```
$ lpr part0 part1 part2 part3 part5
```

```
$ lpr part[01235]
```

```
$ lpr part[0-35]
```

Os Caracteres Coringa: `[]`

- Outros exemplos usando hífen
  - Imprimir 39 arquivos, indo de `part0` até `part38` (em sequência):

```
$ lpr part[0-9] part[12][0-9] part3[0-8]
```

  - Mostrar arquivos que começam com `a` até `m`

```
$ echo [a-m]*
```

  - Mostrar arquivos que terminam com `x`, `y` e `z`

```
$ echo *[x-z]
```

Os Caracteres Coringa: [ ]

- Quando um caractere de ponto de exclamação (!) ou circunflexo (^) é imediatamente seguido do colchete de abertura, significa que não casa com os caracteres especificados dentro dos colchetes
  - Exemplo: `[^tsq]*` casa qualquer nome de arquivo que não começa com `t`, `s` e `q`

- O primeiro comando a seguir mostra arquivos que não terminam com `a` ou `b`, enquanto o segundo mostra arquivos que não começam com `b`, `c` ou `d`

```
$ ls
aa ab ac ad ba bb bc bd cc dd
$ ls *[^ab]
ac ad bc bd cc dd
$ ls [^b-d]*
aa ab ac ad
```

Dica: É possível casar os caracteres – ou] colocando-os imediatamente após colchete de abertura

Os Caracteres Coringa: `[:class:]`

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 (`\t`), nova linha (`\n`), *form feed* (`\f`) e *carriage return* (`\r`)
`upper`	Caracteres maiúsculos do alfabeto: `[A-Z]`
`xdigit`	Dígitos hexadecimais: `[0-9]`, `[a-f]` e `[A-F]`

Resumo

- Exemplos resumo de uso dos caracteres coringa

Padrão	Casa com
`*`	Todos os arquivos
`g*`	Qualquer arquivo começando com `g`
`b*.txt`	Qualquer arquivo começando com `b`, seguido de quaisquer caracteres e terminando com `.txt`
`Data???`	Qualquer arquivo começando com `Data` seguido de exatamente três caracteres quaisquer
`[abc]*`	Qualquer arquivo começando com `a`, `b` ou `c`
`BACKUP[0-9][0-9][0-9]`	Qualquer arquivo começando com `BACKUP` seguido de exatamente três caracteres numerais
`[[:upper:]]*`	Qualquer arquivo começando com um caractere maiúsculo
`[![:digit:]]*`	Qualquer arquivo não começando com um número
`*[[:lower:]123]`	Qualquer arquivo terminando com um caractere minúsculo ou um dos numerais de `1` a `3`

EXPANSÃO TIL (~)

Expansão Til (~)

- O caractere til (~) tem um significado especial
  - Quando utilizado sozinho, expande no diretório pessoal (*home*) do usuário atual (*current user*)

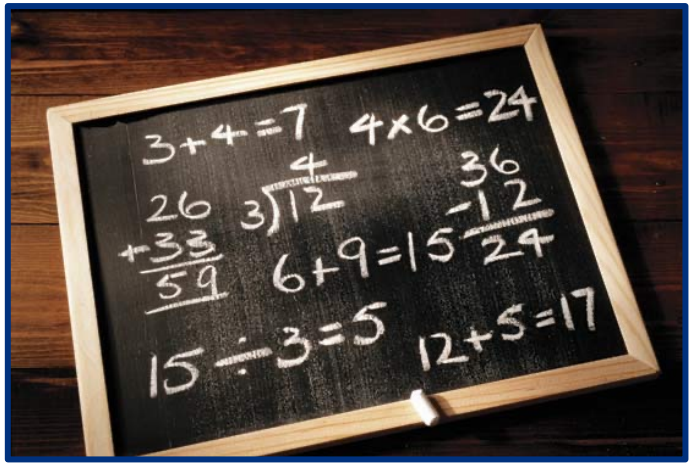
```
$ echo ~
/home/rimsa
```

  - Quanto utilizado no começo de uma palavra, expande no nome do diretório pessoal do usuário especificado, se esse existir

```
$ echo ~root
/root
```

De onde vem o conteúdo desta expansão?

EXPANSÃO ARITMÉTICA



Expansão Aritmética

- A *shell* permite expansões para realização de operações aritméticas; isso permite usar a *shell* como uma calculadora

```
$ echo $((2 + 2))
4
```

- Expressões aritméticas são da forma: `$((expressão))`, em que `expressão` consiste de valores e operadores aritméticos

Operador	Descrição
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão
**	Exponenciação

Expansões aritméticas suportam apenas **números inteiros**

Expansão Aritmética

- Espaços são ignorados em expressões aritméticas e expressões podem ser aninhadas
  - Por exemplo, multiplicar 5^2 por 3:

```
$ echo $(($((5**2)) * 3))
75
```

- Parênteses únicos também podem ser usados para agrupar múltiplas subexpressões
  - Por exemplo, a mesma operação acima reescrita de outra forma

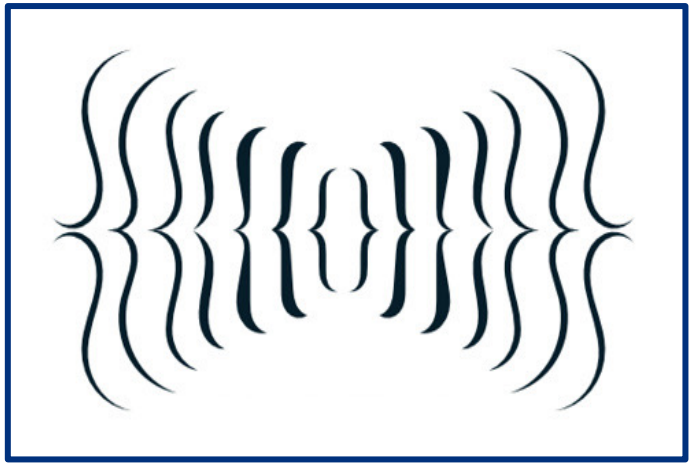
```
$ echo $(((5**2) * 3))
75
```

Expansão Aritmética

- Exemplo com operações de divisão e resto da divisão para demonstrar o efeito da divisão inteira

```
$ echo 5 divido por 2 igual a $((5/2))
5 divido por 2 igual a 2
$ echo com $((5%2)) de resto
com 1 de resto
```

EXPANSÃO DE CHAVES



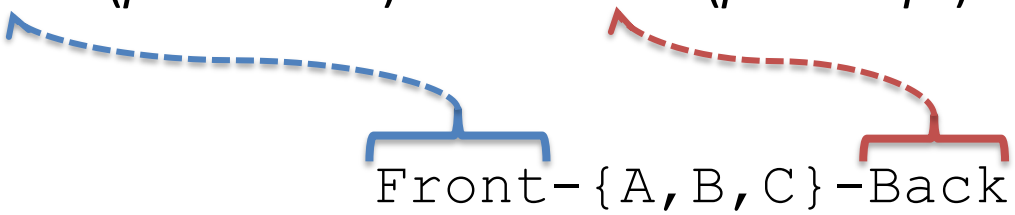
Shell Scripting

Expansão de Chaves

- Com a expansão de chaves, pode-se criar múltiplas strings a partir de um padrão contendo chaves

```
$ echo Front-{A,B,C}-Back
Front-A-Back Front-B-Back Front-C-Back
```

- Padrões com chaves podem ser expandidos para conter um **prefixo** (*preâmbulo*) e um **sufixo** (*postscript*)



Expansão de Chaves

- Além da expansão usando vírgulas, pode-se usar uma faixa de valores
  - Faixa de inteiros

```
$ echo Number_{1..5}
Number_1 Number_2 Number_3 Number_4 Number_5
```

  - Faixa de caracteres

```
$ echo {Z..A}
Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
```

- Expansões de chaves podem ser aninhadas também

```
$ echo a{A{1,2},B{3,4}}b
aA1b aA2b aB3b aB4b
```

Cuidado: o padrão não pode conter espaços em branco (use aspas para resolver)

Expansão de Chaves

- Um exemplo real

```
$ mkdir -p Pics/{{2009..2011}-0{1..9},{2009..2011}-{10..12}}
$ ls Pics/
2009-01 2009-07 2010-01 2010-07 2011-01 2011-07
2009-02 2009-08 2010-02 2010-08 2011-02 2011-08
2009-03 2009-09 2010-03 2010-09 2011-03 2011-09
2009-04 2009-10 2010-04 2010-10 2011-04 2011-10
2009-05 2009-11 2010-05 2010-11 2011-05 2011-11
2009-06 2009-12 2010-06 2010-12 2011-06 2011-12
```

É possível fazer de um jeito melhor?

EXPANSÃO DE PARÂMETROS

Expansão de Parâmetros

- A *shell* é capaz de armazenar uma quantidade de pequenos pedaços de dados e dar um nome a cada um deles (variáveis)
- Por exemplo, a variável de nome `USER` contém o nome do seu usuário; para invocar a expansão de parâmetros e exibir seu conteúdo, basta colocar o símbolo de dólar antes da variável

```
$ echo $USER
rimsa
```

Dica: para ver a lista de variáveis disponíveis, basta executar o comando `printenv` ou `set`

- Note que se escrever o padrão errado, não será impresso nada; diferente de outras expansões que imprimiam o próprio padrão

```
$ echo $SUER

```

SUBSTITUIÇÃO DE COMANDOS

Substituição de Comandos

- A substituição de comandos permite usar a saída de um comando como uma expansão em linha de comando

```
$ echo $(ls)
Desktop Documents Music Pictures
Public Templates Videos
$ ls -l $(which cp)
-rwxr-xr-x  1 root  wheel  24768 Sep  9 19:49 /bin/cp
```

Dica: existe uma sintaxe alternativa para a substituição de comandos, basta envolver um comando entre duas crases: `` `comando` ``

- A substituição de comandos não está limitada a comandos simples, pode-se usar qualquer comando processável pela *shell*

```
$ file $(ls /usr/bin/* | grep unzip)
/usr/bin/bunzip2:  Mach-O 64-bit executable x86_64
/usr/bin/funzip:   Mach-O 64-bit executable x86_64
/usr/bin/gunzip:   Mach-O 64-bit executable x86_64
/usr/bin/unzip:    Mach-O 64-bit executable x86_64
/usr/bin/unzipsfx: Mach-O 64-bit executable x86_64
```

ESCAPES

Escapes

- Agora que foram vistas quantas formas a *shell* pode fazer expansões, é preciso saber como controlá-las quando preciso
  - Nesse exemplo, a *shell* remove espaços extras (*word splitting*)

```
$ echo This is a    test
This is a test
```

  - Nesse outro, a *shell* expande o parâmetro `$1`, mas como não possui valor esse é expandido em branco

```
$ echo The total is $100.00
The total is 00.00
```

Como exibir a saída conforme pretendido?

Escape com Aspas Duplas

- Se colocar texto entre aspas duplas, todos os caracteres especiais perdem seu significado e são tratados como caracteres ordinários, exceto `$` (sinal de dolar), `\` (barra invertida) e `` ` `` (crase)
  - Isso significa que *word splitting*, expansão de arquivos, expansão de til e expansão de chaves são suprimidas, mas expansão de parâmetros, expansão aritmética e substituição de comandos não

- Exemplo, suponha um arquivo chamado `two words.txt`

```
$ ls two words.txt
ls: two: No such file or directory
ls: words.txt: No such file or directory
$ ls "two words.txt"
two words.txt
```

Escape com Aspas Duplas

- Exemplo em que a expansão de parâmetros, expansão aritmética e substituição de comandos ainda são executados entre aspas duplas

```
$ echo "$USER $((2+2)) $(cal)"
rimsa 4     November 2014
Su Mo Tu We Th Fr Sa
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30
```

Escape com Aspas Duplas

- Por padrão, *word splitting* procura pela presença de espaços, tabulações e novas linhas e os trata como delimitadores de palavras; ou seja, não são considerados como parte do texto
- O fato que novas linhas são consideradas delimitadores pelo mecanismo de *word splitting* traz efeitos interessantes na substituição de comandos

```
$ echo $(cal)
November 2014 Su Mo Tu We Th Fr Sa 1 2 3 4 5 6 7 8 9 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
$ echo "$(cal)"
   November 2014
Su Mo Tu We Th Fr Sa
                  1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30
```

O primeiro comando possui 39 parâmetros, enquanto o segundo somente um parâmetro

Escape com Aspas Simples

- Se quiser suprimir todas as expansões e substituições de comandos, basta usar aspas simples

- Exemplo de um mesmo comando sem aspas, com aspas duplas e com aspas simples

```
$ echo text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER
text /home/rimsa/test.txt a b foo 4 me
$ echo "text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER"
text ~/*.txt {a,b} foo 4 me
$ echo 'text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER'
text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER
```

Escape de Caracteres

- As vezes é interessante escapar apenas um único caractere; para isso basta colocar uma \ (barra invertida) antes dele

```
$ echo "O saldo para o usuário $USER é: \$5.00"
O saldo para o usuário rimsa é: $5.00
```

- É comum usar caracteres de escape para eliminar o tratamento especial de um caractere em um nome de arquivo
  - Isso inclui caracteres como $, !, &, espaço, entre outros

```
$ mv bad\&filename good_filename
```

Dica: use duas barras invertidas para mostrar uma única barra

Cuidado: usar a barra invertida entre aspas simples perde seu significado especial

Escape de Caracteres

- A barra invertida é usada também para representar alguns caracteres especiais chamados de **códigos de controle**
  - Alguns exemplos famosos: *tab*, *backspace*, *line feed*, *carriage return*
  - Outros nem tanto: *null*, *end of transmission*, *acknowledge*

- A tabela a seguir mostra algumas sequências de escape famosas

Sequência de escape	Significado
`\a`	Alerta (sino): faz o computador emitir um sinal de bipe
`\b`	*Backspace*
`\n`	Nova linha (*line feed*)
`\r`	Retorno de carro (*carriage return*)
`\t`	Tabulação

Ideia originada na linguagem C que inspirou sua aplicação em outros lugares, como na shell

Escape de Caracteres

- Adicionar a opção `-e` para o comando `echo` irá habilitar a interpretação de sequência de escape

```
$ sleep 10; echo -e "Time's up\a"
```

- Pode-se também colocar a sequência de escape dentro de `$''`

```
$ sleep 10; echo "Time's up" $'\a'
```

ISSO É TUDO, PESSOAL!