[MAC0211] Laboratório de Programação I
Aula 19
Expressões Regulares e AWK

Kelly Rosa Braghetto

DCC–IME–USP

16 de maio de 2013

Na aula passada...

- GNU Autoconf
- Expressões regulares (introdução)

[Aula passada, sobre expressões regulares]
Tipos no UNIX e seus derivados – I

- **Expressões *Glob*** – limitam-se às expressões envolvendo metacaracteres usadas nos shells antigos para o casamento com nomes de arquivos.
  Metacaracteres existentes:
  - `*` casa com qualquer sequência de caracteres.
    Exemplo: carn* casa com *carne*, *carneiro*, *carnudo*
  - `?` casa com exatamente 1 caracter simples.
    Exemplo: ?at casa com *Bat* ou *cat*, mas não com *at* ou *habitat*
  - `[...]` casa com uma classe de caracteres.
    Exemplo: [CB]at casa com *Cat* ou *Bat*, mas não com *cat*, *bat* ou *at*

[Aula passada, sobre expressões regulares] Tipos no UNIX e seus derivados – II

- **Expressões regulares básicas** – notação aceita pelos utilitários `grep`, `sed` (stream editor) e `ed` (line editor)
- **Expressões regulares estendidas** – notação aceita pela versão estendida do `grep`, o `egrep`. Expressões regulares no *Lex* e no *Emacs* são muito parecidas com essas
- **Expressões regulares Perl** – notação aceita para as funções de expressões regulares de Perl e Python. Essas expressões são bem mais poderosas que as anteriores

[Aula passada] Metacaracteres das expressões regulares básicas

`.`	um caracter qualquer
`[abcd]`	um caracter no conjunto dado
`[a-zA-Z]`	um caracter nos intervalos dados
`[^abcd]`	um caracter que não está no conjunto dado
`^`	início da cadeia de caracteres
`$`	fim da cadeia de caracteres. Em ferramentas que se baseiam no processamento de linhas, casa com o final da linha
`*`	elemento precedente zero ou mais vezes

[Aula passada] Exemplos de expressões regulares básicas

'x.y'	x seguido por qualquer caracter seguido por y
'x\.y'	x seguido por um ponto seguido por y
'xz*y'	x seguido por qualquer número de instâncias de z, seguido por y. Ou seja, "xy" ou "xzy" ou "xzzzy", mas não "xz" nem "xdy"
"s[xyz]t"	s seguido por qualquer um dos caracteres x, y ou z, seguido por t. Ou seja, "sxt", "syt" ou "szt", mas não "st" nem "sat"
's[^x0-9]t'	s seguido por qualquer caracter que não seja x nem algum outro no intervalo 0–9, seguido por t. Ou seja, "slt" ou "smt", mas não "sxt", "s0t" nem "s4t"
'^x'	x no início de uma cadeia de caracteres. Ou seja, "xzy" ou "xzzy", mas não "yzy" nem "yxy"
'x$'	x no fim de um cadeia de caracteres. Então, "yzx" ou "yx", mas não "yxz" or "zxy"

Metacaracteres das expressões regulares estendidas

?	elemento precedente uma ou nenhuma vez
+	elemento precedente uma ou mais vezes
(n1|n2|n3)	qualquer uma das opões listadas
{m,n}	elemento precedente pelo menos m e não mais que n vezes

Exemplos

"xz?y"	x seguido por no máximo um z seguido por y. Ou seja, "xy" ou "xzy", mas não "xz" nem "xdy"
"xz+y"	x seguido por uma ou mais instâncias de z, seguido por y. Ou seja, "xzy" ou "xzzy", mas não "xy" nem "xz"e nem "xdy"
"xz{2,5}y"	x seguido por 2, 3, 4 ou 5 instâncias de z, seguido por y. Ou seja, "xzzy", "xzzzy", "xzzzzy" ou "xzzzzzy"

Algumas classes de caracteres POSIX

POSIX	ASCII	Descrição
[:alnum:]	A-Z a-z 0-9	caracteres alfa-numéricos
[:alpha:]	A-Z a-z	caracteres do alfabeto
[:blank:]	\t	espaço e tabulação
[:cntrl:]	\x00-\x1F \x7F	caracteres de controle
[:digit:]	0-9	dígitos
[:lower:]	a-z	letras minúsculas
[:space:]	\t \r \n \v \f	caracteres brancos
[:upper:]	A-Z	letras maiúsculas
[:xdigit:]	A-F a-f 0-9	dígitos hexadecimais

Exemplos no *bash*

- Lista arquivos cujo nome começa com uma letra entre 'a' e 'h':
  ls -l [a-h]*
- Mostra as linhas de `arq.txt` que começam com a sigla de uma disciplina do MAC:
  grep -E '^MAC[[:digit:]]{3,4}' arq.txt
- Verifica se a sequência de caracteres digitada é um *username* válido (ou seja, que começa com letra, mas na sequência pode conter letras ou números, e possui tamanho mínimo de 2 caracteres e máximo de 7):
  grep -E '^[A-z][A-z0-9]{2,7}'

Obs.: A opção '-E' no `grep` habilita o uso de expressões regulares estendidas.

Awk

- O *Awk* é uma ferramenta para tratamento de textos baseada em expressões regulares
- Com ela, podemos fazer operações tais como:
  - Processamento de arquivos texto e criação de relatórios a partir dos resultados
  - Tradução de arquivos de um formato para outro
  - Criação de pequenos bancos de dados
  - Realização de operações matemáticas em arquivos de dados numéricos
- O *Awk* pode ser usado para realizar tarefas simples de processamento de texto via linha de comando, ou como uma linguagem de programação para a criação de scripts para processamentos mais complexos
- *Awk* vem do nome dos seus criadores: Aho, Weinberger & Kernighan

Estrutura de um script Awk

- Formato mais simples:

```
awk '<padrão de busca> <ações do programa>' arq_entrada
```

- Formato mais geral:

```
awk [-F<sc>] 'prog' | -f <arq_prog> [<vars>] [-|<arq_entrada>]
```

sc	caracter separador de campo
prog	programa Awk de linha de comando
arq_prog	arquivo contendo um programa Awk
vars	inicialização de variáveis
-	lê da entrada padrão
arq_entrada	arquivo texto de entrada

onde arq_prog é um script que possui a seguinte estrutura

```
BEGIN                {<inicializações>}
<padrão de busca> {<ações do programa>}
<padrão de busca> {<ações do programa>}
...
END                  {<ações finais>}
```

Awk – entradas

- O programa `awk` lê um script e o aplica na sua entrada
- O script pode ser passado como o primeiro parâmetro do programa (entre ‘ ’) ou então através da opção `-f nome_do_script`
- A entrada é quebrada em registros (normalmente, linhas)
- Os registros são quebrados em campos (normalmente, nos espaços em branco) que podem ser referenciados por meio das variáveis posicionais $1, $2, ..., $n
- $0 se refere ao registro inteiro
- `NR` é uma variável “embutida” que contém o número de registros já processados
- `NF` é a variável que contém o número de campos do registro atual
- `length` é a variável que contém o número de caracteres da linha atual
- Para definir um outro separador de campos, basta usar a opção `-F` no `awk` para determinar o separador.

Expressões regulares do Awk

Semelhantes às do UNIX:

^	casa com o começo da string (p.ex., começo da linha)
$	casa com o fim da string (p.ex., fim da linha)
\	é o metacaracter de *escape*, que pode ser usado para remover o significado especial de um metacaracter
.	casa com qualquer caractere, inclusive o newline
[xyz]	casa com 1 caractere do conjunto xyz
[^xyz]	casa com 1 caractere qualquer que não esteja no conjunto xyz
|	para indicar alternativas (ou)
*	casa com a expressão anterior repetida 0 ou mais vezes
+	casa com a expressão anterior repetida 1 ou mais vezes
BEGIN	casa com o início do arquivo
END	casa com o final do arquivo

Awk – exemplos simples

- Move todos os arquivos cujo nome inicia com “junk” para o diretório “lixo”, renomeando-os com uma extensão “”.lix’
  ls junk* | awk ‘{print “mv “$0” ../lixo/“$0”.lix”}’ | bash

- Calcula a soma e a média dos números armazenado em um arquivo (um número por linha)

```
BEGIN { s = 0 }
      { s += $1 }
END   { print "Soma: ", s, "Media: ", s/NR  }
```

- Calcula o tamanho médio das linhas de um arquivo

```
    { s += length }
END { print "Tamanho medio das linhas: ", s/NR }
```

Outras características importantes do Awk

- As variáveis não são declaradas. Elas passam a existir na primeira vez em que são usadas
- Não existe tipo de variável; uma mesma variável pode conter num dado momento uma string e, num outro, um número
- Não é preciso inicializar variáveis; o valor inicial é sempre string vazia (o que é convertido em 0 numa operação aritmética)
- É possível a utilização de vetores. Os índices de vetores podem ser tanto números quanto strings (ver o exemplo anterior)

Awk – exemplos envolvendo vetores

- Contador de ocorrências de palavras

```
    {
        for (i = 1; i <= NF; i++)
            freq[$i]++
    }
END {
        for (palavra in freq)
            printf "%s\t%d\n", palavra, freq[palavra]
    }
```

- Ordena um arquivo considerando como chave os valores numéricos armazenados em seu primeiro campo

```
    {
       if ($1 > max)
         max = $1
       vet[$1] = $0
    }
END {
       for (x = 1; x <= max; x++)
         if (x in vet)
           print vet[x]
    }
```

Awk – exemplos envolvendo busca de padrões

Na próxima aula...

Bibliografia e materiais recomendados

- Livro: *The Art of Unix Programming*, de Eric S. Raymond. Capítulo 8, Seção "Case Study: Regular Expressions"
  `http://www.catb.org/esr/writings/taoup/html/ch08s02.html#regexps`
- Livro: *Mastering Regular Expressions*, de Jeffrey Friedl, considerado a "Bíblia" das expressões regulares
  `http://regex.info/`
- GNU AWK – Guia do Usuário
  `http://www.gnu.org/software/gawk/manual/gawk.html`
- *An Awk primer* (tutorial de Awk bastante interessante)
  `http://www.vectorsite.net/tsawk.html`
- Notas das aulas de MAC0211 de 2010, feitas pelo Prof. Kon
  `http://www.ime.usp.br/~kon/MAC211`

Cenas dos próximos capítulos...

Na próxima aula:

- Awk (continuação)
- Geradores de analisadores léxicos
- GNU Flex