Expressões Regulares

- Definição (Aurélio Marinho Jargas)
  - Formal
    - "*É um método formal de especificar um padrão de texto*"
  - Informais
    - "*Uma maneira de procurar um texto que você não lembra exatamente como é, mas tem uma idéia das variações possíveis*"
    - "*Uma maneira de procurar um trecho em posições específicas como no começo ou no fim de uma linha, ou palavra*"
    - "*Uma maneira de um programador especificar padrões complexos que podem ser procurados e casados em uma cadeia de caracteres*"
    - "*Uma construção que utiliza pequenas ferramentas, feita para obter determinada seqüência de caracteres de um texto*"
- Recurso poderoso utilizado por muitos aplicativos do GNU/Linux
  - Exemplos
    - `grep`, `egrep`, `find`, `sed`, `awk`, `perl`, `python`, `ed`, `vi` **e** `emacs`

Expressões Regulares

- Poucos usuários dominam satisfatoriamente os recursos providos pelas expressões regulares
  - Documentação existente não é clara e completa
    - *Programming Perl* (Larry Wall, O'Reilly) possui um excelente capítulo sobre o uso e aplicação de expressões regulares
- Um pouco de história
  - 1943: as ER's surgiram a partir do estudo de dois neurologistas sobre o funcionamento dos neurônios humanos
  - Alguns anos depois um matemático definiu um modelo para representar este estudo
    - Foram definidos grupos regulares (*regular sets*) que deram origem as expressões regulares
  - Primeira implementação em computador deu-se em 1968 no editor `qed` que depois tornou-se ou `ed`
    - Comandos `g` e `p` do ed (`g/RE/p/`) deram origem ao `grep`

Expressões Regulares

- Metacaracteres
  - Correspondem as ferramentas básicas de uma expressão regular
  - São combinados para representar o padrão de busca desejado
- Metacaracteres tipo representante
  - Ponto: “.”
    - Curinga que casa com **uma única** letra, número, caracter especial (`@`,`#`,`$`,`%`,...) , TAB, o próprio ponto, ...
    - Exemplos

`.ato`	pato, rato, gato, ...
`n.o`	não, nao, nÃo, ...
`.im`	fim, Fim, ...
`13.30`	13:30, 13.30, 13 30, 13-30, ...
`c.mprido`	comprido, cumprido, ...

  - Lista: [...]
    - Mais específica que o ponto, a lista determina quais caracteres ou símbolos podem ser casados
    - Só pode ser casado um caracter por vez dentro de cada lista

Expressões Regulares

- Exemplos

`n.o`	não, nao, nÃo, n9o, nxo, n@o,...
`n[aã]o`	não, nao (Obs: não casa com `naão`)
`[pgr]ato`	pato, gato, rato
`12[:. ]45`	12:45, 12.45, 12 45
`<[BIP]>`	<B>, <I>, <P>

- Observação
  - O ponto dentro da lista NÃO é um metacaracter e sim um caracter normal

– Listas com intervalos

- Como representar uma ER para casar com qualquer letra minúscula?
  - `[abcdefghijklmnop...z]` ??? MUITO LONGA!
- Para facilitar a representação de seqüências, pode-se usar intervalos

```
[0123456789]                = [0-9]
[abcdefghijklmnop...z]      = [a-z]
[ABCDEFGHIJKLMNOP...Z]      = [A-Z]
```

- Mais de um intervalo pode ser utilizado em uma lista

```
[0-57-9]                    = [012345789]
[d-fA-C5-7]                 = [defABC567]
```

Expressões Regulares

- Observações
  - Como representar o "–" dentro de uma lista se ele é especial?
    - Deve-se sempre colocá-lo no final da lista

      `[a-f-]` = `[abcdef-]`

      `[0-9-]` = `[0123456789-]`
  - Como representar o "]" dentro de uma lista se ele é especial?
    - Deve-se sempre colocá-lo no início da lista

      `[]]` casa somente com o]

      `[]6-9]` casa com],6,7,8,9

      `[]-]` casa com] ou -

- Intervalos respeitam a ordem da tabela ASCII

  `[:-@]` casa com :, ;, <, =, >, ?, e @

- Classes de caracteres POSIX (padrão internacional)
  - Incluem letras acentuadas

  `[:upper:]` letras maiúsculas

  `[:lower:]` letras minúsculas

  `[:alpha:]` letras maiúsculas e minúsculas

  ...

Expressões Regulares

  - Representações POSIX variam conforma a configuração do sistema
    - Nos EUA, `[[:upper:]]` é igual a `[A-Z]`
    - No Brasil, `[[:upper:]]` inclui os caracteres acentuados `Á, É, Ã, . . .`

  - Listas negadas
    - Funcionam com a lógica inversa da lista normal
    - Tudo que encontra-se na lista não será casado

    `[^0-9]` casa com qualquer símbolo que NÃO seja número

    `[:;,.!?][^ ]` casa com qualquer pontunação que não esteja seguida por um espaço em branco

- Metacaracteres tipo quantificador
  - Opcional: ?
    - Indica **nenhuma ou uma** ocorrência do padrão anterior

    `[pgr]atos?` pato, gato, rato, patos, gatos, ratos

    `casa[r!]?` casa, casar, casa!

Expressões Regulares

- Asterisco: *
  - Indica **nenhuma ou muitas** ocorrências do padrão anterior

`to*c` tc, toc, tooc, toooc, tooooc, ...

`t[oc]*` t, to, tc, toc, tooc, tocc, toccoooc, ...

  - Sempre é tentado casar o maior número de vezes
    - O que casará `[ar]*a` na palavra *arara* ? (Expressões Regulares, Aurélio)

`1.a` `[ar]` zero vezes, seguido de `a`

`2.ara` `[ar]` duas vezes, seguido de `a`

`3.arara` `[ar]` quatro vezes, seguido de `a`

`4.n.d.a`

- Mais: +
  - Indica **uma ou mais** ocorrências do padrão anterior

`to+c` toc, tooc, toooc, tooooc, ...

`t[oc]+` to, tc, toc, tooc, tocc, toccoooc, ...

Expressões Regulares

- Chaves: {...}
  - Permite que seja especificada um limite mínimo e/ou máximo de repetições para o padrão anterior
  - Possibilidades

`{n,m}`	de `n` até `m`
`{n,}`	pelo menos `n`
`{n}`	exatamente `n`
`{0,}`	o mesmo que o "`*`"
`{1,}`	o mesmo que o "`+`"

  - Exemplos

`to{1,3}c`	toc, tooc, toooc
`to{2,}c`	tooc, toooc, tooooc, ...
`t[oc]{2,}`	tooc, toooc, tooooc, tco, toc, tcooocoo...

Expressões Regulares

- Metacaracteres tipo âncora
  - Circunflexo: ^
    - Indica que o padrão a seguir deve ser considerado a partir do início da linha

    `^[a-z]` linhas que começam com letras minúsculas
    `^[0-9]` linhas que começam por números
    `^[^0-9]` linhas que NÃO começam por números

  - Cifrão: $
    - Indica que o padrão anterior deve ser considerado no fim da linha

    `toc$` linhas que terminam com a palavra `toc`
    `[0-9]$` linhas que terminam com número
    `[:-@]$` linhas que terminam com os caracteres :, ;, <, =, >, ?, e @

  - Combinações úteis

    `^$` linha em branco
    `...$` últimos três caracteres da linha
    `^.{15,30}$` linhas que contenham entre 15 e 30 caracteres

Expressões Regulares

- Borda: \b
  - Indica a borda de uma palavra (início ou fim)

`ana`	ana, anamaria, analucia, mariana, luciana
`\bana`	ana, anamaria, analucia
`ana\b`	ana, mariana, luciana
`\bana\b`	ana

  - Por palavra, deve-se entender seqüências de letras, números e o caracter "_"
- Literal: \
  - Deve ser utilizado antes de qualquer metacaracter para torná-lo um caracter normal sem qualquer efeito especial para a expressão regular
- Ou alternativo: |
  - A lista funciona como um tipo de operador "OU" somente para uma letra
  - Indica um padrão com várias alternativas

`Hello | Ola`	casa ou com a palavra `Hello` ou com `Ola`
`http://|https://`	casa com `http://` ou `https://`

Expressões Regulares

- Grupo: (...)
  - Possibilita o agrupamento de caracteres que serão tratados atomicamente

`(oi!)+`	oi!, oi!oi!, oi!oi!oi!, ...
`(\.[0-9]){3}`	.3.4.5, .7.2.4, ...
`(www\.)?comunidadesol.org`	www.comunidadesol.org, comunidadesol.org
`(super|hiper)mercado`	supermercardo, hipermercado
`(su|hi)permercado`	supermercardo, hipermercado
`((su|hi)per)?mercado`	supermercardo, hipermercado, mercado
`(mini|(su|hi)per)?mercado`	supermercardo, hipermercado, mercado,minimercado

Expressões Regulares

- **`e?grep`** `[opções] PADRÃO [FILE ...]`: retorna linhas dos arquivos especificados que casem com o padrão de busca repassado

  - Comando muito popular no mundo GNU/Linux
  - O `egrep` tem a capacidade de processar expressões regulares extendidas
  - Opções
    - `-i`: torna a busca *case-insensitive*
    - `-x`: retorna somente se o texto casar exatamente com o padrão
    - `-r`: busca recursiva a partir do diretório indicado como parâmetro
    - `-f`: carrega modelos a partir do arquivo indicado, um por linha

  - Exemplos

```
$ grep ana agenda.txt
$ grep -x ana agenda.txt
$ egrep '^ana$' agenda.txt
$ egrep 't[oc]*' agenda.txt
$ egrep '(mari|luci)?ana$' agenda.txt
```