Data Science & Big Data

Infraestrutura Computacional
Parte 1: Linux e Shell

DSBD

Filtros e Expressões Regulares

Filtros

Princípio Unix (e Linux): todo comando deve fazer apenas uma coisa, e pode-se facilmente concatenar resultados de diferentes comandos

Um filtro é um programa de linha de comando que recebe dados em forma de texto e transforma estes dados

- Concatenando comandos, podemos aumentar sua aplicabilidade (mais sobre isso daqui a pouco)
- Veremos alguns filtros de maneira individual

Filtros

Comando	Significado
`cat`	mostra o conteúdo
`tac`	mostra o conteúdo do fim para o início
`head, tail`	mostra as primeiras/últimas linhas
`nl`	numera linhas
`diff`	mostra diferença entre dois arquivos
`wc`	conta linhas, palavras e caracteres (*word count*)
`cut`	separa colunas
`uniq`	remove linhas adjacentes duplicadas
`sort`	ordena os dados
`sed`	busca padrões e substitui (*stream editor*)

Exemplos

```
Fred apples 20
Susy oranges 5
Susy oranges 5
Mark watermellons 12
Robert pears 4
Terry oranges 9
Lisa peaches 7
Susy oranges 12
Mark grapes 39
Mark grapes 39
Anne mangoes 7
Greg pineapples 3
Oliver rockmellons 2
Betty limes 14
```

Arquivo **`dados.txt`**

Exemplos

Comando
`cat dados.txt`
`tac dados.txt`
`head -4 dados.txt`
`nl -s '. ' -w 5 dados.txt`
`diff dados.txt dados.txt.old`
`wc -l dados.txt`
`cut -f 1 -d ' ' dados.txt`
`cut -f 1,2 -d ' ' dados.txt`
`sort dados.txt`
`sed 's/oranges/bananas/g' dados.txt`

Expressões Regulares

Que '`^.{5}$`' é isso?

É uma linguagem para descrever padrões de dados

- Utilizadas por diversos comandos e linguagens de programação
- Os caracteres podem ter significado diferente dos *wildcards*
- Exemplos com o comando **`egrep`** (**`grep -E`**)
  - Expressão regular entre aspas simples (`' '`)

Expressões Regulares

Símbolo	Significado
`.`	(ponto) - um caractere simples
`?`	o caractere anterior é casa 0 ou 1 vez
`*`	o caractere anterior casa 0 ou mais vezes
`+`	o caractere anterior casa 1 ou mais vezes
`{n}`	o caractere anterior casa exatamente **n** vezes
`{n,m}`	o caractere anterior casa pelo menos **n** e não mais que **m** vezes

Expressões Regulares

Símbolo	Significado
`[agd]`	o caractere é um dos incluídos entre colchetes
`[^agd]`	o caractere não é um dos incluídos entre colchetes
`[c-f]`	o caractere é um no intervalo entre c e f
`()`	permite agrupar diversos caracteres como se fossem um
`|`	(pipe symbol) - operação lógica OU
`^`	casa com início de linha
`$`	casa com final de linha

Exemplos com egrep

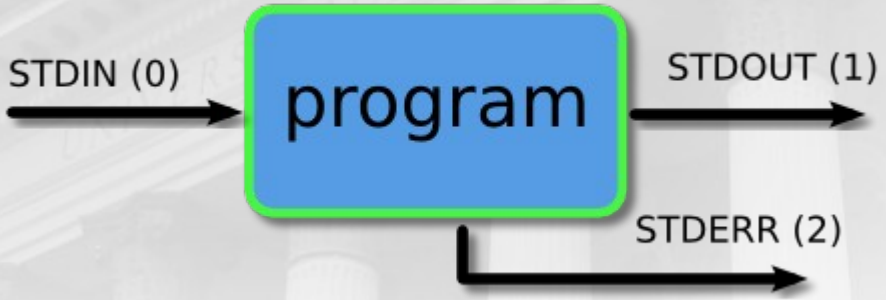
```
egrep 'mellon' dados.txt
egrep -n 'mellon' dados.txt
egrep -c 'mellon' dados.txt
egrep '[aeiou]{2,}' dados.txt
egrep '2.+' dados.txt
egrep '2$' dados.txt
egrep 'or|is|go' dados.txt
egrep '^[A-K]' dados.txt
```

Entrada/saída, Redirecionamento e *Piping*

Entrada e Saída

Todo programa em GNU/Linux possui 3 fluxos de dados (arquivos) conectados a ele

- STDIN (0) – entrada padrão (dados de entrada do programa)
- STDOUT (1) – saída padrão (dados produzidos pelo programa)
- STDERR (2) – saída de erros (mensagens de erro produzidas pelo programa)



Redirecionamento

Os fluxos podem ser redirecionados:

- Para arquivos
  - (**>**), (**>>**) - redireciona STDOUT

```
user@host: ls -l > saida.txt
user@host: ls -l $HOME >> saida.txt
user@host: cat > saida.txt
```

- De arquivos
  - (**<**) - redireciona STDIN

```
user@host: wc -l < dados.txt
user@host: wc -l < dados.txt > saida.txt
```

Redirecionamento

Redirecionamento da saída de erros

- (**2>**), (**2>>**) - redireciona STDERR

```
user@host: ls -l naoexiste 2> erros.txt

user@host: ls -l $HOME bla > saida.txt 2> erros.txt

user@host: ls -l $HOME bla > saida.txt 2>&1
```

Redirecionamento

Quatro regras para redirecionamento de saída

- Redirecionamento é feito **antes** de executar o comando
- Só é possível redirecionar saída que você pode (ou poderia) ver
- Redirecionamento vai apenas para um lugar
- Por padrão, somente a saída é redirecionada (não o erro)

Piping

Ao invés de enviar/receber dados de arquivos, é possível redirecionar dados da saída de um programa diretamente para a entrada de outro

- (**|**) - *pipe* redireciona STDOUT do comando à esquerda para a STDIN do comando à direita

```
user@host: ls -l $HOME | head -3

user@host: ls -l $HOME | head -3 | tail -1
```

- Argumentos de cada comando devem ser providos
  - DICA: construa seu *pipe* aos poucos, com um comando de cada vez

Pipe e Redirecionamento

O resultado final de um *pipe* pode ser redirecionado normalmente, como visto anteriormente

```
user@host: ls -l $HOME | head -3 | tail -1 > saida.txt

user@host: ls -l $HOME | tee saida.txt | head -3

user@host: ls -l $HOME | tail -n +8 | sort
```

Piping

Três regras para *pipes*:

- Redirecionamento por *pipe* é feito pelo shell, por primeiro, antes do redirecionamento de arquivos
- O comando à esquerda precisa produzir dados na saída padrão
- O comando à direita precisa ler da entrada padrão

Perguntas?

Controle de Processos

Processos

O que são processos?

- Uma instância em execução de um programa
- Possui uma área de memória própria, isolada de outros processos
- Diversos processos são executados simultaneamente num sistema GNU/Linux
  - Quando há mais processos ativos do que processadores, o SO dedica um pouco de tempo de processamento a cada processo, criando a ilusão de que eles são executados simultaneamente

Gerenciamento de Processos

Processos podem ser visualizados, mortos, suspensos, executados em segundo plano

Gerenciamento de processos é o trabalho de distribuir tempo de CPU entre vários programas

- Ex: Processo A é executado/interrompido, B é executado/interrompido, A volta a ser executado, …

Cada processo possui uma CPU virtual, e seus dados são carregados para o processador quando o processo está ativo

O usuário pode dar maior prioridade a determinados processos

Gerenciamento de Processos

Comando	Significado
`top`	mostra iterativamente os processos do sistema
`ps [aux]`	mostra os processos do sistema
`kill`	mata processos (sinais: -HUP, -9)
`pkill`	mata processos pelo nome
`strace`	mostra chamadas de sistema de um processo
`nice`	define a prioridade de um processo
`Ctrl+Alt+F?`	abre terminal em modo texto
`Ctrl + z`	suspende processo atual
`Ctrl + c`	mata processo atual
`bg ou &`	coloca processo em segundo plano
`fg`	coloca processo em primeiro plano

Exemplos

Comando
`ls /bin | sort | tee /tmp/lista | wc -l`
`ls -l /home/espinf | tail -n +2 | sed 's/  */ /g' | cut -d ' ' -f 3 | sort | uniq -c`
`ps aux | grep lferrari`
`pkill firefox`
`strace ls $HOME | less`
`find /usr/share/ -iname '*.html' > arq.txt 2>/dev/null`

Referências

- Anatomy of the Linux kernel
- Linux OS Tutorial
- Introduction to UNIX
- Introduction to Linux
- Ryans Linux Tutorial
- Ryans Regular Expressions