



– *UNIX- Avançado*

– *Programação C-Shell*

– *autoras:*

– *Cristiana Munhoz Eugênio*

– *Lilliam Palermo*

•





O que é Shell ?

- ◇ **Programa que conecta e interpreta comandos**
- ◇ **Linguagem de programação completa - interpretada**
 - possui variáveis
 - construções condicionais e interativas
 - ambiente adaptável ao usuário
- ◇ **Uma das linguagens originais de quarta geração (4GL)**
- ◇ **Simplicidade do Shell**
 - pipeline
 - sistemas de arquivos
 - sistema operacional UNIX

Unix Avançado
Programação C-Shell

3

- O programa *shell* interpreta os comandos que você digita e os traduz para comandos que o *kernel* compreende.
- *Shell*: é uma linguagem de programação completa, possuindo variáveis, construções condicionais, interativas e ambiente adaptável ao usuário. O Shell do Unix é a ferramenta original de protótipo rápido que ensina conceitos-chaves como modularidade, reutilização e desenvolvimento.
- Os comandos do Shell se comunicam entre si por meio de uma interface simples e coerente chamada *conduto (pipeline)*.
- O Shell permite que o usuário realize suas atividades sem afetar qualquer outro processo que não lhe pertence. Quando um usuário conecta-se a um sistema Unix, o sistema operacional inicia automaticamente uma cópia do Shell, sob a qual o usuário poderá realizar qualquer função disponível.
- O shell utiliza o sistema de arquivos do UNIX que permite organizar arquivos em pastas (diretórios). Esta hierarquia de diretórios e arquivos gera uma visão simples e clara de toda a informação no sistema.
- O UNIX é transportável; ele roda em quase todo hardware de computadores fabricados atualmente. Seu investimento não será desperdiçado, pois qualquer programa escrito é largamente portátil entre Unix's de diferentes plataformas e fornecedores.



Produtividade

- ◇ **Linguagem interpretada - não compilada**
- ◇ **Um programador médio pode duplicar ou triplicar sua produtividade**
- ◇ **Comparação de Bruce Cox (pai do Objective C)**
 - Shell: 1 linha de código
 - Linguagem O.O: 10 linhas de código
 - Linguagem C: 100 linhas de código

Unix Avançado
Programação C-Shell

4

- Não é de se espantar que o Shell possa duplicar ou triplicar a produtividade pois ele pode automatizar a maior parte das tarefas recursivas, que compreendem de 50% a 80% das atividades intensamente humanas. A simplicidade dos arquivos do UNIX e do projeto de sistema de arquivo possibilitam isso.

- É necessário alguns conhecimentos internos do Shell e do seu uso para se beneficiados seus recursos. Mas, é preciso alguma inventividade para torná-lo mais eficiente e produtivo.



Quando usar

- ◇ **Interativamente**
 - realizar uma operação simples, uma única vez
 - interagir com o comando
- ◇ **O shell script deverá ser utilizado sempre que for necessário realizar:**
 - procedimentos complexos usando muitas linhas de comando
 - procedimento do qual todos os usuários poderão beneficiar-se
 - uma tarefa numa data planejada
 - integrar informações de vários sistemas existentes

Unix Avançado
Programação C-Shell

5

• A qualquer momento que você executa algum comando (ls, cat), você está usando o *Shell* interativamente: *stdin*, *stdout* e *stderr* são todos direcionados para o terminal.

- ***Stdin* => standart input**
- ***Stdout* => standart output**
- ***Stderr* => standart error**

• Quando você começar a usar o *Shell* interativamente e descobrir que algumas tarefas exigem muita digitação, está na hora de você programar em *Shell* colocando esses comandos interativos em um arquivo. Você poderá reutilizar sempre que precisar, melhorando ainda mais a sua produtividade.

• O *Shell* pode extrair e manipular grandes quantidades de informações. Por que pesquisar um relatório inteiro quando o *Shell* pode verificar e recuperar informações importantes para você com mínimo esforço ?



Características do C-Shell

- ◇ Escrita por Bill Joy na Universidade da Califórnia - Berkeley
 - BSD 4.3 e System V
- ◇ Similar à linguagem C
- ◇ Controle sobre as tarefas em primeiro e segundo plano
- ◇ History e Aliasing
- ◇ Operadores condicionais (==, >, etc.)
- ◇ Estrutura de controle (if, while, etc.)
- ◇ Variáveis globais e locais (set, setenv)

Unix Avançado
Programação C-Shell

6

- O **history** armazena comandos à medida que são executados permitindo serem reutilizados sem a necessidade de redigitá-lo.

```
!!    => reexecuta o último comando
!n    => reexecuta o comando número "n" do histórico
!ftp  => reexecuta o último comando ftp
!echo:p => lista o último comando echo executado.
```

- O **Shell C** oferece aliasing, permitindo ao usuário criar nomes simbólicos para nomes de comando.

Exemplos: alias dir 'ls -la | more' ou alias rm 'rm -i'

Evite criar mais de 20 alias pois compromete a performance!!

- No Shell C, você poderá mover tarefas em primeiro plano para o segundo, conforme a necessidade.

- Sempre que você precisar de respostas imediatas, execute os comandos em primeiro plano.

Quando o comando for ocupar muito tempo de terminal, ou seja, levar muito tempo para voltar a resposta, você poderá submetê-lo em segundo plano.

```
Exemplos: find / -name *.ps -print&    (& início do comando em 2º. plano)
          jobs                        (lista os comandos que estão sendo executados)
          fg 1                        (traz o comando para o 1º. plano)
```

Unix Avançado
 Programação C-Shell

Filtros

```

      graph LR
      A[STDIN  
(teclado)] --> B[FILTRO]
      B --> C[STDOUT  
(tela)]
      B --> D[STDERR  
(tela)]
    
```

◇ **grep -i "Arthur" capit1**
 Autor: Lowel Arthur
 Rei Arthur e os Cavaleiros da
 Távola

◇ **sed -e "s/shell/Shell/g" capit1**

7

- O Shell tem comandos que permitem extrair dados relevantes de um arquivo.
- Você pode pensar na maior parte dos comandos do Shell como filtros.
- Cada comando também possui duas saídas:
 - saída-padrão --> **stdout (default tela)**
 - saída-de-erro --> **stderr (default tela)**
- Cada comando filtra dados da entrada-padrão ou transforma-os direcionando-os à saída-padrão. Quaisquer erros encontrados são direcionados à saída-de-erro.
- Filtros podem extrair, incluir ou alterar dados dependendo dos comandos utilizados.
- O comando **sed** (do exemplo acima) abrirá o arquivo "capit1" como *stdin* e passará o arquivo para *stdout* (terminal) enquanto muda todas as ocorrências de "shell" para "Shell".
- **OBS:** Se você **não** redirecionar o resultado da execução dos comandos para um arquivo, utilizando os operadores de redirecionamento, o resultado somente será exibido na tela (**default do stdout**)



Redirecionamento

- ◇ ***Criar arquivos***
- ◇ ***Anexar arquivos***
- ◇ ***Usar arquivos existentes como entrada para o shell***
- ◇ ***Reunir dois fluxos de saída***
- ◇ ***Usar parte do comando shell como entrada***



Unix Avançado
Programação C-Shell

Operadores

- ◇ **< arquivo** direciona um arq p/ *stdin*
- ◇ **> arquivo** direciona stdout p/ uma nova saída
- ◇ **>> arquivo** anexa stdout a um arq. existente
- ◇ **>>&** anexa stderr a stdout
- ◇ **<< delimitador** direciona stdin do shell até o delimitador
- ◇ **>! arquivo** grava sobre noclobber/ cria novo arq.
- ◇ **>>! arquivo** grava sobre noclobber/ anexa novo arq.
- ◇ **|&** reúne stderr e stdout e canaliza resultados

9

- Você pode usar o redirecionamento de E/S para alterar o ***stdin/stdout e stderr***.

Exemplo: **sed -e "s/shell/Shell/g" capit1 > novocapit1**

- O *stderr* ainda estaria direcionado para o terminal, mas o resultado da execução deste comando será gravado em novocapit1.

- Redirecionar *stderr* para um arquivo pode ser ocasionalmente útil para depurar um comando Shell. Para combinar *stderr* com *stdout*, inclua um **&** após o sinal de redirecionamento:

Exemplo: **sed -e "s/shell/Shell/g" capit1 >>& novocapit1**

Se o arquivo **novocapit1** existir, o Shell anexará o texto ao arquivo

- Senão, O Shell vai criá-lo. (comando >>)
- O arquivo de saída terá todos os dados da saída-padrão e da saída-de-erro criados pelo comando do shell. (comando &)

- **noclobber**: variável que pode ser setada para evitar a destruição acidental de arquivos existentes: (setenv noclobber)

Para modificar esta proteção, use o ponto de exclamação (!).

Exemplo: **cat >! novoarq**

- **<<** delimitador: este dispositivo de redirecionamento, usa linhas de dados *dentro* do comando do Shell como entrada. Em vez de ter um arquivo separado como entrada para o comando, você poderá incluí-lo diretamente com o comando do Shell

Exemplo: **ftp -niv ftp.unicamp.br << EOF!**



Conduto

◇ *Meio de condução para transportar dados de um comando para outro*

◇ *O resultado da execução de um comando pode ser utilizado como parâmetro de entrada na execução de outro comando*

◇ *Conecta stdout de um comando à stdin de outro*

◇ *Exemplos:*

```
cat meuarq | grep palavra | wc -l
```

```
ls -al | grep -i "*.txt"
```

- Os comandos do *Shell* podem ser reutilizados, acoplados e moldados para lidar com as mais difíceis aplicações de informação.
- Além de eliminar arquivos temporários, o conduto permite que dois comandos operem ao mesmo tempo.



Construir um shell script

- ◇ **Edite um arquivo:**
vi meu programa
- ◇ **Primeira linha deve conter:**
#!/bin/csh
- ◇ **Para comentar uma linha:**
#
- ◇ **Passar o "caminho" dos comandos:**
/bin/rm arquivo ou \rm arquivo

Unix Avançado
Programação C-Shell

11

- `#!/bin/csh` → É utilizado para que o Unix saiba que seu shell é *cshell*.
- É recomendável especificar o "caminho" dos comandos, pois cada usuário pode criar na sua área um alias para qualquer comando.
- Os comandos ficam geralmente no diretório /bin
- Outra opção seria colocar *escape* (`\`) antes do comando, por exemplo `\rm`. Garante que você estará utilizando a 1ª ocorrência encontrada no path definido no seu `.cshrc`.



Dicas de programação

- ◇ **Os shell scripts devem estar sempre em um diretório comum.**
 - Exemplo: /usr/sys/nome-sistema
- ◇ **Arquivos de controle que são gerados pelo shell devem estar em um diretório temporário, separados dos shell scripts**



Unix Avançado
Programação C-Shell

Dicas de programação

- ◇ *Utilize filtros de redução*
- ◇ *Utilize condutos ao invés de arquivos temporários*
- ◇ *Se seu shell for fazer FTP lembre-se de alterar o `chmod` do arquivo*
- ◇ *Sempre utilize delimitador nos arquivos*
- ◇ *Utilize delimitador único e diferente para subcampos*
- ◇ *Organize os dados para maior eficiência da máquina*

13

- Se o seu shell for fazer FTP, alterar a permissão do arquivo que contém o username e a senha para "700". **Este é um item obrigatório por medida de segurança.**

- Utilize filtros de redução, ***grep***, ***cut*** e ***awk*** em primeiro lugar em um conduto, reduzindo a quantidade de dados que deverão ser transferidos pelo Shell.

Exemplo: `grep $1 arq1 | sort`

- Utilize condutos ao invés de arquivos temporários para melhorar a eficiência do seu shell

Com arquivo temporário:

```
cut -f1,5 arq1 > /tmp/arq1tmp
sort /tmp/arq1tmp
rm /tmp/arq1tmp
```

Com conduto:

```
cut -f1,f5 arq1 | sort
```

- Qualquer programa Shell com mais de duas páginas é muito complexo.



Para executar um shell script

◇ **Fora do editor “vi”, na linha de comando:**

`csh -opção meu-programa`

◇ **Dentro do editor “vi”:**

`:w` (grava o arquivo)

`!:csh -opção %`

– ! (executa o comando shell seguinte)

– % (diz ao “vi” para preencher o nome do arquivo corrente)

- Você pode executar o seu programa à medida que for sendo desenvolvido, facilitando também a detecção e correção de erros. Entre no *vi* e crie as primeiras linhas do programa, grave o arquivo (`:w`) e execute-o sem sair do editor:

Comando :

`!:csh -opção %`



Opções de teste

- ◇ **-e**
 - sai imediatamente se houver código de retorno falso
- ◇ **-v**
 - verbose (maiores informações)
- ◇ **-x**
 - gera saída dos comandos executados

Unix Avançado
Programação C-Shell

15

- Se você quiser gravar num arquivo a execução de um shell:
 - Execute o comando **script**, (será criado um arquivo chamado typescript)
 - Execute a shell
 - Ao final da execução do shell tecler CTRL D (Pronto, o arquivo typescript conterá a execução da shell.)
- Para ler o arquivo typescript use o comando **more**



Caracteres especiais

- ◇ **Aspas duplas (")**
"string"
– mostra o string e substitui variáveis (se existir)
- ◇ **Aspas simples (')**
'string'
– mostra o string e **NÃO** substitui variáveis (se existir)
- ◇ **Crase (`)**
`comando(s)`
– o resultado do comando é usado em output
- ◇ **Barra invertida (\)**: transforma caracter especial em caracter normal

- A utilização de caracteres especiais dentro de um string dará problemas. Para isso utilize o recurso da "barra invertida".

- Exemplos:

```
set dia = 19
echo " Hoje é dia $dia"
resultado = Hoje é dia 19
```

```
set dia = 19
echo ' Hoje é dia $dia'
resultado = Hoje é dia $dia
```

```
set maquina = `hostname`
echo $maquina
Resultado: sc13.ccuec.unicamp.br
```



Metacaracter

- ◇ ; *Separador de comando sequencial*
- ◇ ? *Combina com qualquer caracter isolado*
- ◇ * *Combina com qualquer cadeia de caracter*
- ◇ [a-z*] *Combina com algo que consista de caracteres alfabéticos minúsculos*

Unix Avançado
Programação C-Shell

17

- Suponha que no seu diretório contenhas os arquivos:
 - capit1
 - capit2
 - capit3
 - capit4

- ls capit(*) - aparecerá os 5 arquivos

- ls capit? - aparecerá os arquivos (capit1, capi2, capit3, capit4)

- ls capit[123] aparecerá os arquivos (capit1, capi2, capit3)



Variáveis no C-Shell

◇ *Globals*

- SETENV
- Exemplos:

```
setenv TERM vt100
setenv DBDATE Y4MD-
setenv DISPLAY máquina:0.0
```

Unix Avançado
Programação C-Shell

18

- Nomes de Variáveis (locais ou globais) **não** pode conter hífen:
nota-aluno (dará erro) **X**
- É permitido o uso do underscore:
minha_var
nota_aluno
- **SETENV**: Configura variáveis de ambiente ou globais. Estas variáveis serão válidas enquanto a sessão estiver aberta.



Locais

◇ *Atribuindo valores do tipo STRING:*

`set var = expressão`

Exemplos:

`set msg = "Ola"`

`set ano = 1998`

`set dir_log = ~/exercicios`

`set meudir = `ls -a``

`set name = "Curso de C-Shell"`

Array:

`set var = (a b c)`

◇ *Não usar hífen no nome de variáveis*

Set var = expressão

- Seta variáveis locais, ou seja, elas atuam somente durante a execução do Shell
- A variável recebe um valor string
- `set var = (a b c)` é seta um array chamado *var* contendo *a*, *b*, *c*, respectivamente na primeira, segunda e terceira posições.

Obs: Para saber quais variáveis estão setadas e quais são seus valores, utilize o comando **set**.



Locais

◇ *Atribuindo valores do tipo INTEIRO:*

@ var = expressão

Exemplos:

- @ ano = 1998**
- @ soma = \$valor1 + \$valor2**
(separados por branco)
- @ i = 1**
- @ j++** *(soma 1 na variável)*
- @ p--** *(subtrai 1 na variável)*
- @ ano += 2** *(soma 2 na variável ano)*
- @ 1 - var** *(subtrai 1 na variável)*

20

Unix Avançado
Programação C-Shell

- Entre o @ e o nome da variável é necessário um espaço em branco.
- Já para incrementar ou decrementar, não pode existir espaço em branco entre a variável e o sinal correspondente.
Ex: @ h++
@ p--
- Para trabalhar com o conteúdo de uma variável devemos utilizar o sinal "\$".
 - set ano = 1998
 - echo \$ano
 - echo "O valor de soma e: \$soma"



Manipulando variáveis

◇ *\$variável*

substitui o nome da variável pelo seu conteúdo

Exemplos:

```
ls $dir_log  
set letra = $var[1]  
set k = $var[$i]  
@ k = ($i + 3)  
set dia = 19  
set parm1 = $1  
echo "Hoje é dia $dia"
```

- Não deixe espaço em branco entre o cifrão (\$) e o nome da variável.



\$?variável

◇ ***Testa a existência da variável (se ela tem valor)***

= 1 se a variável está setada

= 0 se a variável não está setada

Exemplos:

```
set var = 5
```

```
echo $?var
```

```
Resultado: 1
```

```
unset var
```

```
echo $?var; echo $var
```



\$status

◇ ***Testa se o comando foi executado com sucesso***

= 0 sucesso

= 1 sem sucesso

Exemplos:

```
set var = 5
```

```
echo $status
```

```
(0)
```

```
unset var
```

```
echo $status
```

```
(0)
```

```
echo $var4; echo $status
```

```
(1)
```



\$#variável

◇ *Retorna o número de elementos de um array*

Exemplos:

```
set name = (maria lucia joao)
```

```
echo $#name
```

```
- Resultado: 3
```

- ***\$#variável*** → volta sempre um número inteiro que pode ser usado para alguma finalidade



\$variável[n]

◇ ***\$variável[n]*** ou ***\$variável[n-y]*** ou ***\$variável[n-]***
– elementos posicionais

Exemplos:

```
set nome = (maria jose joao)
$nome[1]
$nome[1-3]
$nome[2-]
```

25

Unix Avançado
Programação C-Shell

- Se você ultrapassar o número de subscript, ou seja, tentar usar um elemento que ultrapasse o número existente no array, o Cshell emitirá uma mensagem
→ “Subscript out of range “
- Exemplos: \$nome[1] (Maria)
\$nome[1-3] (Maria José João)
\$nome[2-] (José João)



Passagem de parâmetros

- ◇ **`$#argv`**
número de argumentos passados como parâmetro
- ◇ **`$1, $2, $3`**
argumentos posicionais
- ◇ **`$0`**
nome do programa
- ◇ **`$*`**
todos os argumentos
- ◇ **`$$`**
no. do processo do comando corrente
- ◇ **`$argv[n]`**
conteúdo do parametro n passado

26

```

• csh          meuprogram      parm1      parm2
                $0                $1          $2
                        $argv[1]    $argv[2]

```

- **`$$`** → Muito utilizado pelos utilitários de mail que gravam o arquivo em `/tmp/arq_número_do_processo`

- Se você tem um shell script que recebe parâmetros de entrada, você pode usar **`$#argv`** no começo do programa para obter o número de argumentos (parâmetros) recebidos, testando se esse era o número esperado.

Exemplo: **`csh meuscript parm1 parm2`**

```

#!/bin/csh
if ($#argv != 2) then
    echo "Número de parâmetros inválido "
    exit
else
    set nome = $1 ou set nome = $argv[1]
    set ramal = $2 ou set nome = $argv[2]

endif

```



Variáveis conhecidas

- ◇ *\$user*
- ◇ *\$cwd*
- ◇ *\$path*
- ◇ *\$home*
- ◇ *\$prompt*
- ◇ *\$shell*
- ◇ *\$term*
- ◇ *\$hostname*

Unix Avançado
Programação C-Shell

27

- *\$user* : Conteúdo do resultado do comando whoami (quem é o usuário da conta)
- *\$cwd* : Conteúdo do resultado do comando pwd (diretório corrente)
- *\$path* : Lista de diretórios a serem procurados pelo meus executáveis
- *\$home* : Mostra qual é o diretório home
- *\$prompt* : Mostra o que está setado como prompt
 - set prompt = `pwd`
 - set prompt = \$cwd
- *\$shell* : Mostra a shell que está sendo utilizada
- *\$term* : Mostra qual o tipo de terminal que está sendo utilizado
- *\$hostname*: Nome do computador ao qual estou logado
- O valor das variáveis acima podem ser encontrados ao digitarmos o comando **set**.



Expressões e Operadores

◇ *Comparar variáveis numéricas ou strings*

== (igual)

!=(diferente)

◇ *Comparar somente strings*

=~ (igual)

!~(diferente)

◇ *Operador lógico*

and, or: &&, ||

◇ *Operador aritmético*

+, -, *, /

◇ *maior, menor, igual: >, <, >=, <=*

- ```
if ($a == $b) then
 echo "As variáveis são iguais"
else
 echo "As variáveis são diferentes"
end
```
- ```
if ($nome != "Curso Shell ") then
    echo " Você esta no curso errado !!!!!"
else
    echo " Bem vindo ao Curso de Shell"
end
```
- ```
while (" $1" != "")
end
```
- ```
if ($i <= $k) then
```
- ```
if ("$1" == "Maria" || "$2" == "João") then
```



## *if - then - else*

◇ *if (expressão) then*  
    *processa1*  
    *else*  
    *processa2*  
    *endif*

◇ *if (\$a == \$b) then*  
    *echo "variáveis iguais"*  
    *else*  
    *echo "variáveis diferentes"*  
    *endif*

◇ *if (-e arq\_entrada) then*  
    *echo "O arquivo existe"*  
    *end*

- O **then**, obrigatoriamente, tem que estar na mesma linha de comando do **if** e não pode ter outro comando na linha.
- Na linha do **else** não pode ter comandos



Unix Avançado  
 Programação C-Shell

## Operadores de teste

| <u>Condição</u>   | <u>Verdadeira se ...</u>                        |
|-------------------|-------------------------------------------------|
| -e <i>arquivo</i> | se <i>arquivo</i> existe                        |
| -r <i>arquivo</i> | existir <i>arq.</i> Legível                     |
| -w <i>arquivo</i> | existir <i>arq.</i> Gravável                    |
| -x <i>arquivo</i> | existir <i>arq.</i><br>Executável               |
| -d <i>dir</i>     | existir o diretório                             |
| -z <i>arquivo</i> | <i>arq.</i> existir e não<br>tiver              |
| -f <i>arquivo</i> | dados (tamanho = 0)<br>existir (tendo ou<br>não |
|                   | dados) 30                                       |

```

if (-e arq_entrada) then
 echo " O arquivo existe"
else
 echo " O arquivo NÃO existe"
end

```

```

if (-d Mail) then
 echo "Isto é um diretório"
else
 echo "Isto é um arquivo"
end

```

```

if (! -d Mail) then
 echo "Isto é um arquivo"
else
 echo "Isto é um diretório"
end

```



## Switch

◇ **switch (string)**  
    **case valor1:**  
        **ação 1**  
        **breaksw (obrigatório)**  
    **case valor3:**  
        **ação 2**  
        **breaksw**  
    **default:**  
        **ação default**  
        **breaksw**  
**endsw**

Unix Avançado  
Programação C-Shell

31

Exemplo de um script utilizando o comando switch

```
set data = `date | cut -c9-10`
```

```
set texto = "Relatorio do mes de"
```

```
switch ($data)
```

```
case 01:
```

```
 echo "$texto Janeiro"
```

```
 breaksw
```

```
case 09:
```

```
 echo "$texto Setembro"
```

```
 breaksw
```

```
default:
```

```
 echo "Relatório Anual"
```

```
 breaksw
```

```
endsw
```

- Obs: É obrigatório a variável case ser string.  
    É obrigatório ter um *breaksw* para cada comando *case*.



## Foreach

◇ *foreach* *variável* (*expressão*)  
    *comando* *\$variável*  
    *end*

**Expressão:**

- resultado de um comando
- valores ou ocorrências de um array

*As instruções dentro do foreach serão executadas tantas vezes quantos forem o número de elementos presentes entre parênteses*

32

Exemplos:

- Utilizando array:

```
set tabela = (maria joão)
foreach pessoa ($tabela[*])
 echo $pessoa
end
```

- Utilizando o resultado de um comando:

```
foreach arquivo (`ls -l | awk '{print $3, $4, $8}'`)
 echo $arquivo
end
```

- **Não** há comando “**FOR**” para C Shell



## **While**

◇ *while* (*expressão*)  
    *ações*  
    *end*

◇ *while* (*\$a <= 10*)  
    *\$a++*  
    *echo "Valor = \$a"*  
    *end*



## While

```

◇ while (1)
 echo "Centro de Computação"
 echo "Selecione a opção
desejada"
 echo "1 - Notice"
 echo "3 - Sair"
 echo -n "--> "
 set RESP = (<)
 switch ($RESP)

 endsw
end

```

Unix Avançado  
 Programação C-Shell

34

```

while (1)
 echo " 1- Tabela de Orgaos "
 echo " 2 - Notice "
 echo " X - Sai "
 echo -n "Selecione a opcao desejada --> "
 set RESP = (<)
 switch ($RESP)
 case 1:
 source jsi03e_carga; breaksw
 case 2:
 /usr/local/notice/manager ; breaksw
 case X:
 exit
 default:
 echo -n " Opcao Invalida - Tecl e <enter> "
 set CONT = (<)
 breaksw
 endsw
end

```

- comando exit: sai do while sem ser pela condição definida na expressão



## Comandos do Shell

◇ *Os comandos shell serão mostrados divididos em várias categorias:*

- *Arquivos e diretórios*
- *Seleção*
- *Combinação e ordenação*
- *Impressão*
- *Segurança*

Unix Avançado  
Programação C-Shell

35

- Para cada comando existem várias opções. Neste curso tentaremos mostrar as opções mais utilizadas.
- Para conhecer outras opções ou maiores detalhes utilize o comando **man**:

**man** nome\_comando

- teclas para utilizar no comando man :

|                    |                               |
|--------------------|-------------------------------|
| .próxima linha     | ENTER                         |
| .próxima página    | space bar                     |
| .página anterior   | CTRL B                        |
| .procuo string     | ^caracter especial ou /string |
| .continuar a busca | n                             |
| .cancelar pesquisa | q ou CTRL C                   |

**OBS:** Nem todos os comandos usam as opções da mesma maneira.





## Impressão

### ◇ *ECHO*

escreve *argumentos em stdout*

Sintaxe:

`echo [opções] string`

Opções:

**-n**: não adiciona uma nova linha no *stdout* antes de imprimir o *string*, deixando o cursor na linha da mensagem escrita

Exemplos:

```
echo -n "Entre com a opção desejada => "
 Entre com a opção desejada → __
set dia = `date +%A`
echo "Hoje é $dia"
 Hoje é Monday
echo "Bom dia"
echo "Os arquivos são: " *
echo "Essa pessoa $nome é muito legal" > arq
```



## Impressão

### ◇ *CAT*

- concatena e imprime
- não altera dados. Toma a *stdin* e a reproduz no *stdout*
- varre o arquivo e não pára a cada tela do terminal

#### Sintaxe:

`cat [opções] [- file1 file2 ...]`

#### Opções:

`-n`: lista as linhas do arquivo numerando-as

### • Exemplos

- |                                          |                                                                                                                                     |
|------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <code>cat capt1 capit2 &gt; livro</code> | livro irá conter capit1,2                                                                                                           |
| <code>cat capt3 &gt;&gt; livro</code>    | anexa capit3 no livro                                                                                                               |
| <code>cat capit1</code>                  | lista capit1 no terminal                                                                                                            |
| <code>cat -n capit1</code>               | lista capit1 numerando as linhas                                                                                                    |
| <code>cat &gt; arq1</code>               | direciona o stdin para o terminal e stdout para arq1, assim tudo que for digitado será gravado no arq1. Para encerrar tecle CTRL D. |



## Segurança

◇ **CHMOD**

- para tornar o arquivo do shell executável é necessário alterar a permissão do arquivo

**Sintaxe:**  
chmod [quem] operação acesso arquivo

**Opções:**  
quem: u (usuário), g (grupo), o (outros), a (all)  
operação: + (atribuir), - (remover), =  
acesso: r (read), w (write), x (execução)

39

- **ATENÇÃO:** Se o seu shell for executar o comando FTP você deverá alterar a permissão do arquivo que contém o username e a senha para que somente o dono possa ler, gravar e executar.

Exemplos:

```
chmod u=rwx,go= arquivo
ls -la
-rwx----- 1 lilliam supsof 110 Sep 25 16:33 arquivo
```



## Leitura

◇ \$<  
– direciona o terminal como *stdin*

Exemplo:  
set opção = \$<

Unix Avançado  
Programação C-Shell

40

Exemplos:

```
echo -n "Digite a opcao: "
set opcao = $<
if ($opcao > 9) then
 echo "Opcao Invalida. Tecele enter para continuar"
 $<
endif
```

- Existem duas maneiras de trabalhar com este comando
- set opção = \$<  
( neste caso, eu estou guardando o que o usuário digitou
- \$<  
( Está é uma maneira de deixar a msg na tela até o usuário digitar alguma tecla.



## Seleção

### ◇ *LINE*

- copia uma linha do *stdin* e grava no *stdout*

Sintaxe:

```
line < stdin > stdout
```



## Seleção

### ◇ HEAD

- seleciona as 10 primeiras linhas de um arquivo

#### Sintaxe:

```
head [opções] stdin > stdout
```

#### Opções:

- c *n*: seleciona as *n* primeiras colunas do *stdin*
- n: seleciona as *n* primeiras linhas do *stdin*

42

- Exemplos

```
head -5 arquivo (lista as 5 primeiras linhas do arquivo)
```



## Seleção

◇ *TAIL*

- seleciona as 10 linhas finais de um arquivo

Sintaxe:  
`tail [opções] stdin > stdout`

Opções:

- n*: seleciona as *n* últimas linhas do *stdin*
  
- f*: *mostar as linhas que são adicionados ao arquivo*

43

- Exemplos

`tail -5 arquivo` (lista as 5 últimas linhas do arquivo)

`tail -f /var/messages`

Unix Avançado  
 Programação C-Shell



## Seleção

◇ **SED**  
 – edita e altera dados

**Sintaxe:**  
**sed [opções] stdin > stdout**

**Opções:**  
**-e "script":**  
     **onde script pode ser:**  
     *"s/string\_antiga/string\_nova/g"*  
     *"/string que pertence à linha que deverá ser deletada/d"*  
     *"s/string a ser deletada//g"*

**-f arquivo: contém os scripts a serem executados. Não usar aspas dentro do arquivo**  
**Não usar stdin como stdout**

44

- O comando **sed** muda a informação dentro de um arquivo ou tabela. Este comando também deleta linhas de arquivos. Portanto pode ser utilizado como um programa que atualiza campos ou deleta linhas.

- Exemplos:

- **sed -e "s/caf /ch /g"** entrada  
 Altera a palavra **caf ** para **ch ** em todas as ocorr ncias (g) no stdin
- **sed -e "/Jo o/d"** entrada  
 Deleta todas as linhas do stdin que cont m a palavra **Jo o**
- **sed -e "s/devedor//g"** entrada  
 Altera a palavra **devedor** por nada (/), ou seja, sed deletou a palavra
- **sed -e "s/cha/caf " -e "s/limao/mate/"** arq\_sala > arqsaida
- **sed -f** arqcom entrada > arqsaida  
 Executa os comandos que est o no arquivo **arqcom**.  
     arqcom: s/shell/Shell/g -> altera todas as ocorr ncias por (g)  
           s/Ter a/ter a/ -> altera 1 ocorr ncia por linha

**OBS:**Na op o **-f** n o usar aspas dentro do arquivo



## Seleção

### ◇ **UNIQ**

- *seleciona apenas uma ocorrência de cada linha do stdin. Obrigatoriamente stdin deve estar ordenado (comando sort)*

#### **Sintaxe:**

*uniq [opções] stdin*

#### **Opções:**

- u: seleciona linhas únicas*
- c: conta o número de vezes que cada linha se repete*
- d: seleciona apenas as linhas com mais de uma ocorrência*



## Seleção

◇ **GREP**  
– *seleciona linhas usando um único critério de seleção*

**Sintaxe:**  
*grep [opções] string stdin*

**Opções:**  
-c: *conta o número de vezes que o string aparece*  
-i: *ignora uppercase*  
-l: *lista somente o nome dos arquivos que possuem o string procurada*  
-n: *numera cada linha que contém o string procurada*

46

- O comando *grep* é a ferramenta principal para extrair informações de campos de dados, procurar cadeia de caracteres e escrever a informação em stdout.
- Exemplos:
  - grep "Mar\*" arq\_user - listará todas as linhas do arq\_user que contém palavras que contenham a string "Mar"
  - grep Maria arq\* - listará todas as linhas de arquivos que comecem com arq que contenham a string Maria
  - grep -l Maria arquiv\* - listará os nomes dos arquivos que contenham a string Maria
  - grep -n Maria arq\_user - listará todas as linhas do arq\_user que contenham a string Maria, numerando-as
  - grep \$variável arq1 - listará todas as linhas do arq que contenham a variável
  - grep -c Chico arq1 - conta o no. de vezes que a string Chico aparece no arquivo
  - grep -i maria arq1 - listará todas as linhas que contém a string maria (maiúscula ou minúscula)



## Seleção

◇ **GREP**

**Opções:**

- v: lista todas as linhas do arquivo exceto às que contém a string**
- ^letra: lista todas as linhas que iniciem com a letra ou string**

**String:**

- \$variável, "string composta", string,**
- "metacaracteres" ( \* ? [a-z] )**

◇ **grep, head, line, sed, tail e uniq operam em linhas**

47

- Exemplos:

grep -v "Amélia" arq1 - lista todas as linhas do arq1 exceto às que contém a string "Amélia"

grep ^R arq1 - lista todas as linhas do arq1 que começam com R

grep "[a-z]" arq1 -



Unix Avançado  
Programação C-Shell

## Seleção

◇ **CUT**  
– *seleciona colunas*

**Sintaxe:**  
*cut [opções] stdin*

**Opções:**  
*-cx-y: define coluna inicial (x) e  
coluna final (y)*  
*-dChar: define delimitador*  
*-fx,y: define campo inicial (x) e  
campo final (y)*  
*-s: suprime linhas que não  
contém delimitador.*  
*Obrigatoriamente deve ser usado  
com a opção -d*

**OBS: O comando cut trabalha com  
colunas**

48

- O comando **cut** corta arquivos em colunas. Também pode operar em cada caractere ou campo, ou alguma combinação de ambos. Quando o arquivo tem algum separador entre campos você pode selecioná-los através de **cut**.

- Exemplos:

`cut -f1,5 -d: arquivo` - mostra campos de 1 a 5 (inclusive)

`cut -c1-72 arquivo` - mostra coluna de 1 a 72 ( inclusive)

`cut -d: -s -f3,5 arquivo` - mostra campos de 3 a 5 exceto as linhas que não contém delimitador

OBS: Não é permitido o uso da opção **-c** junto com a opção **-d**.



## Seleção

◇ **WC**

- conta linhas, palavras e caracteres de arquivos

**Sintaxe:**

```
wc [opções] stdin
```

**Opções:**

- c: conta caracteres
- l: conta linhas
- w: conta palavras

Unix Avançado  
Programação C-Shell

49

• Exemplos:

```
grep -l Maria arqu* | wc -l - lista a qtd de linhas que contém a variável
Maria
```

```
cat arq | wc -l - lista a qtd de linhas do arquivo arq
```



## Seleção

◇ **SPLIT**

- *corta o stdin em arquivos menores contendo um número específico de linhas*

**Sintaxe:**

*split [opções] stdin [prefixo]*

**Opções:**

- n*: *especifica o número de linhas de cada arquivo de saída. Default 1000 linhas por arquivo*
- prefixo*: *prefixo dos stdouts criados. O sufixo será aa, ab até zz. O default do prefixo é x*

50

- O comando **split** cria o nome do 1º. arquivo do output combinando o prefixo mais o sufixo aa, ab (p/ o 2o.) até zz.

- Exemplos:

split -10 arquivo (gera arquivos xaa, xab, ...) cada um contendo 10 linhas

split -5 arquivo capit (gera arquivos capitaa, capitab,...) cada um contendo  
5 linhas

O split quebra por quantidade de linhas e não por quantidade de arquivos.



## Seleção

### ◇ CSPLIT

- *separa os arquivos sempre que encontrar uma combinação entre seus argumentos e o conteúdo do arquivo*

#### Sintaxe:

*csplit [opções] string  
[argumentos]*

#### Opções:

*-f prefixo: prefixo dos stdouts criados. O sufixo varia de 00 à 99.  
Default xx*

- Exemplos

O arquivo livro contém:

Introdução,  
Capítulo 1,  
Capítulo 2,  
Resumo.

```
csplit -f capit livro "/Capítulo /" "/Capítulo /"
```

resultado: capit00 (contém Introdução)  
capit01 (contém Capítulo 1)  
capit02 (contém Capítulo2 e Resumo)

```
csplit -f capit livro "%Capítulo 1%"
```

resultado: capit00 (contém Capítulo 1 Capítulo 2 e Resumo)



## Seleção

### ◇ CSPLIT

#### Opções:

*"/argn/": cria um arquivo por argumento (inclusive). O arquivo prefixo00 contém as linhas anteriores ao primeiro argumento*

*"%arg%": cria um arquivo contendo os dados a partir do argumento*

Não funciona no Free BSD



## Ordenação

### ◇ *SORT*

– *ordena e combina múltiplos arquivos*

*Sintaxe:*

*sort [opções] arquivos ...*

*Opções:*

*-b: ignora espaços em branco para encontrar a primeira ou última coluna do campo*

*-d: classifica usando ordem alfabética. Somente letras, dígitos e espaços são considerados na comparação*

*-f: altera todas as letras minúsculas para maiúsculas antes da comparação*

53

- O comando **sort** opera com caracter, campo ou em uma combinação dos dois. Caso o arquivo classificado possua linhas duplicadas estas podem ser retiradas através do comando **uniq**. Para tornar mais eficiente esse comando deve ser utilizado após os dados terem sido selecionados (**grep** ou **cut**).

- Exemplos :

sort -nru arquivo - ordena o arquivo em ordem numérica decrescente e deleta as linhas duplicadas.

sort -m arq1 arq2 - junta arq1 e arq2 e ordena

sort -f arq1 -o arq1\_sort - ignora uppercase e grava o resultado em arq1\_sort.



## Ordenação

◇ *SORT*

**Opções:**

- m: junta todos os arquivos de entrada num só. Os arquivos de entrada devem estar classificados*
- n: ordena o arquivo numericamente*
- r: ordena o arquivo em ordem decrescente*
- u: deleta linhas duplicadas*
- t: define delimitador*
- k [campo\_inicial.coluna\_inicial] [opção] [, campo\_final.coluna\_final] [opção]: define a chave do sort*

54

Unix Avançado  
Programação C-Shell

- As opções válidas para a **opção -k** são: b,d,f,n,r
- Os valores default para a opção -k:

|         |                        |
|---------|------------------------|
| cpo ini | começo da linha        |
| col ini | 1ª. coluna do campo    |
| cpo fim | final da linha         |
| col fim | última coluna do campo |
- Exemplos:
  - sort -k2,2f -k3 arquivo
  - sort -k1.1 -k2.4 arquivo
  - sort -rt ' ' -k3b -k4 arquivo



## Transformação

### ◇ *TR*

– *transforma caracter por caracter*

#### **Sintaxe:**

```
tr 'string_antiga' 'string_nova'
<stdin> stdout
```

- O comando *tr* traduz os dados de entrada caractere por caractere, com base em tabelas de conversão (string) especificadas pelo usuário.
- Exemplos:

```
tr '(A-Z)' '(a-z)' < arq1 > arq1min
tr '(:)' '(t)' < cad1 > cad1t
```



## Transformação

### ◇ *DD*

– converte e copia arquivo

Sintaxe:

```
dd if=stdin of=stdout
conv=[opções]
```

Opções:

*lcase*: converte o conteúdo do arquivo para minúsculo

*ucase*: converte o conteúdo do arquivo para maiúsculo

*ascii*: converte ebcdic para ascii

*ibm*: converte ascii para a versão ebcdic da IBM



## Junção

### ◇ JOIN

– *junta dois arquivos combinando linha a linha. Os arquivos devem estar ordenados*

#### Sintaxe:

*join [opções] arquivo1 arquivo2*

#### Opções:

*-tChar: define delimitador*

*-a no. arquivo: produz uma linha de saída para cada linha do arquivo especificado pelo n arquivo, mesmo para os campos do join que não casam com nenhuma linha do outro arquivo de entrada*

57

- Quando o **join** encontra registros que se combinam nos 2 arquivos de entrada ele cria um único registro contendo algum ou todos os campos dos dois registros. Para utilizá-lo, os dois arquivos devem estar ordenados (utilize **sort**).

**OBS:** Obrigatoriamente a opção **-a** tem que vir antes da opção **-t**.



## Junção

◇ **JOIN**

**Opções:**

- o *no arquivo.campo*: as linhas de saída são compostas pela lista dos campos especificadas na variável
- n*arquivo.campo*: para definir mais campos separe-os por branco ou vírgula
- 1 *campo* ou -2 *campo*: faz o join de 2 arquivos usando o campo especificado na variável *campo* do arquivo 1 (-1) ou arquivo 2 (-2)

58

• Exemplos:

|                          |                                       |
|--------------------------|---------------------------------------|
| <b>arq_ent1:</b> Aldo:HC | <b>arq_ent2:</b> Aldo:enfermeiro:7234 |
| Bruno:IMECC              | João:digitador:7111                   |
| João:CCUEC               | Ricardo:médico:7323                   |

```
join -t: arq_ent1 arq_ent2 > arq_saida
arqsaida: vazio
```

```
join -a1-t: arq_ent1 arq_ent2 > arqsaida
arqsaida: Aldo:HC:enfermeiro:7234
 Bruno:IMECC:_____
 João:CCUEC:digitador:7111
```

```
join -o 1.1 1.2 2.3 -t"." arq_ent1 arq_ent2 > arqsaida
arqsaida: Aldo:HC:7234
 João:CCUEC:7111
```



## Junção

◇ **PASTE**  
– anexa múltiplos arquivos coluna por coluna

**Sintaxe:**  
*paste [opções] arquivo1  
arquivo2*

**Opções:**  
-dChar: define delimitador  
-s: junta linhas subsequentes do arquivo

Unix Avançado  
Programação C-Shell

59

- Exemplos :

```
arq_ent1: Aldo:HC arq_ent2: Aldo:enfermeiro:7234
 Bruno:IMECC João:digitador:7111
```

```
paste -d "/" arq_ent1 arq_ent2 > arq_saida
arq_saida: Aldo:HC/Aldo:enfermeiro:7234
 Bruno:Imecc/João:digitador:7111
```

```
paste -s -d "/" arq_ent1 arq_ent2 >
arq_saida: Aldo:HC/Bruno:Imecc
 Aldo:Enfermeiro:7234/João:digitador:7111
```



## Seleção e Impressão

### ◇ AWK

– *linguagem de processamento e procura de padrões. Encontra linhas no(s) arquivo(s) de entrada que combinam com um padrão especificado e então executa as ações definidas*

#### Sintaxe:

***awk [opções] 'padrão' {-f pgm 'pgm'} stdin***

#### Opções:

***-Fchar: define delimitador***

***-f pgm: arquivo que contém as ações a serem executadas pelo comando awk***

***'pgm': são as ações a serem executadas pelo awk***

60

```
awk -F: '{print $5 $1}' arq_fun)
```

```
awk -F: '{print $2 " " $1}' arq_fun
```

```
awk -F: {'$1 ~/e|a/'} arq_fun
```

```
awk -F: '{print NR,NF,$0}' arq_fun
```

```
awk -F: '{print $1 > "matr"; print $2 > "nome"}' arq_fun
```

```
awk -F: '{printf "%15s %10s\n", $1, $3}' arq1
```

```
awk -F: '/smith+ern/' arq1
```

```
awk -F: '/amel??/' arq_fun
```

```
awk -F: '{printf ("nome eh %s ramal %5d\n ", $2,$4)} arq_fun
```

```
awk -F: '$3 ~/Oct/ {print $2 $3}' arq1
```



## Seleção e Impressão

◇ *Tipos de padrões*

- '/string/*
- '/string1+string2/*
- '/string1?/*
- '/string1|string2/*
- '/string1[char1-char2]/*
- '/string1[^\^char1-char2]/*

Unix Avançado  
Programação C-Shell

61

Tipos de padrões:

- **'/string/** : grava em stdout todas as linhas do stdin que contém pelo menos uma ocorrência do string.
- **'/string1+string2/** : grava em stdout as linhas que contém o string1 seguido por um ou mais caracteres que precede o sinal de + e termina com o string2.
- **'/string1?/** : grava em stdout as linhas que contém o string1 seguido por nenhum ou um caracter que precede o sinal ?
- **'/string1|string2/** : grava em stdout as linhas que contém o string1 e/ou o string2.
- **'/string1[char1-char2]/** : grava em stdout as linhas que contém o string1 seguido por qualquer caracter que esteja entre char1 e char2 inclusive (char1 e char2 devem estar em ordem)
- **'/string1[^\^char1-char2]/** : grava em stdout as linhas que contém o string1 seguido por qualquer caracter que **não** esteja entre char1 e char2 inclusive (char1 e char2 devem estar em ordem).



## Seleção e Impressão

◇ *Tipos de padrões*

- '\$1 ~/char1/'
- '\$1 ~/^char1/'
- '\$2 ~/char1\$/'
- '\$1 == "string1"'
- '\$1 >= "char1"'
- '\$1 == "string1" && \$2 == "string2"'

Unix Avançado  
Programação C-Shell

62

Tipos de padrões:

- '\$1 ~/char1/' : grava em stdout todos os registros que contém no 1o. campo o caracter char1 ou um string .
- '\$1 ~/^char1/' : grava em stdout todos os registros que **não** contém no 1o. campo o caracter char1 ou um string.
- '\$2 ~/char1\$/' : grava em stdout todos os registros que contém o último caracter igual a char1 ou string.
- '\$1 == "string1" ' : grava em stdout todos os registros que contém o 1o. campo igual ao string1
- '\$1 >= "char1" ' : grava em stdout todos os registros que o 1o. campo é maior ou igual a char1
- '\$1 == "string1" && \$2 == "string2" ' : grava em stdout todos os registros que o 1o. campo é igual ao string1 e o 2o. é igual o string2."



## Seleção e Impressão

◊ *Tipos de ações ( OUTPUT)*

- *Impressão de campos, variáveis e constantes:*  
`{print $1 " " $2}`
- *Redirecionamento:*  
`{print $1 > "arq_output"}`
- *Instruções aritméticas:*  
`{print $1 *= $2}`  
`{print "ra", $1 "média:", $2 / $3}`
- *Variáveis especiais:*  
`{print NR, $0}`

63

Tipos de ações (Output) :

Impressão de campos, variáveis e constantes:

`{ print $1 " " $2 }` : grava em stdout o campo 1 e campo 2 separados por um espaço em branco.

Redirecionamento:

`{ print $1 > "arqoutput" }` : redireciona o campo 1 para o stdout arqoutput .

Instruções aritméticas:

`{ print $1 *= $2 }` : o campo 1 é o resultado da multiplicação do campo1 com o campo2.

`{print "ra", $1 "média :", $2 / $3 }` : grava em stdout o campo 1 precedido do string **ra** e o string **média** é o resultado da divisão do campo \$2 pelo campo \$3 .

Variáveis Especiais:

`{ print NR, $0 }` : grava em stdout os registros de stdin numerado.



## Seleção e Impressão

### ◇ Tipos de ações

- Concatenação de campos:  
`{print $1 == $1 $2}`
- Imprimir linha em branco:  
`{print " "}`
- Impressão formatada de campos variáveis ou constantes:  
`{printf "%15s\n", $1}`  
`{printf "%20d %-5s\n", $3+$2, $1}`  
`{printf "%20s %4d\n", "NOME", "RAMAL"}`

| <u>seqüência</u> | <u>significado</u> |
|------------------|--------------------|
| <code>\n</code>  | nova linha         |
| <code>\b</code>  | backspace          |
| <code>\t</code>  | tab                |



## Seleção e Impressão

### ◇ Tipos de ações

**BEGIN**

```
BEGIN { FS = ":"
{printf ("%20s %5s\n", "NOME",
"RAMAL")}
{printf ("%20s %5s\n", $2, $1)}
```

**END**

```
END {printf ("%20s", "fim de
relatório")}
END {print "fim de relatório"}
```

### **\*Begin**

Ações que devem ser executadas antes do stdin ser lido. Na execução de mais de uma ação ela deve estar entre colchetes.

### **\*End**

Ações que devem ser executadas após o stdin ser lido. Se for executar mais de uma ação ela deve estar entre colchetes.



## Seleção e Impressão

### ◇ Tipos de ações

#### – Instruções condicionais:

```
if (expressão) {ações} [else
{ações}]
```

```
if ($1 != prev) {
 print " "
 prev = $1}
```

#### – Variáveis especiais:

**NR:** número da linha  
corrente

**NF:** quantidade de campos  
da linha

**FS = "caracter":** define  
delimitador



## Exemplos de formatação *printf*

| <u>fmt</u> | <u>\$1</u> | <u>printf '{ fmt,\$1}'</u>      |
|------------|------------|---------------------------------|
| %c         | 97         | a (character ASCII)             |
| %d         | 97.5       | 97 (decimal inteiro)            |
| %5d        | 97.5       | bbb97                           |
| %06d       | 97         | 000097                          |
| %e         | 97.5       | 9.750000e+01<br>(exponenciação) |
| %f         | 97.5       | 97.500000                       |
| %7.2f      | 97.5       | 97.50<br>(ponto flutuante)      |



## Exemplos de formatação *printf*

| <u>fmt</u> | <u>\$1</u> | <u>printf '{ fmt,\$1}'</u> |
|------------|------------|----------------------------|
| %s         | Janeiro    | Janeiro (string)           |
| %10s       | Janeiro    | Janeiro                    |
| %-10s      | Janeiro    | Janeiro                    |
| %.3s       | Janeiro    | Jan                        |
| %10.3s     | Janeiro    | Jan                        |
| %-10.3s    | Janeiro    | Jan                        |
| %%         | Janeiro    | %                          |



## Comando vi em modo batch

◇ *Utilizando o editor vi:*

```
vi arquivo < comandos_vi >&!
/dev/null
```

◇ *arquivo comandos\_vi:*

```
/FIM
dd
^ [
:x
```

Unix Avançado  
Programação C-Shell

69

- Comandos vi:

vi abre o arquivo e executa os comandos gravados no arquivo comandos\_vi  
>&! /dev/null - se a execução do vi der algum erro ele joga fora as mensagens de erro

^ [ - representa a tecla ESC (digitar CTRL v CTRL [ )

:x - sai do editor vi e salva o arquivo



## Formato do comando date

```
`date +%a` => Tue
`date +%A` => Tuesday
`date +%b` => Sep
`date +%B` => September
`date +%c` => Tue Sep 10 15:58:44 1998
`date +%d` => 10 (dia)
`date +%D` => 09/10/98 (mês/dia/ano)
`date +%j` => 254
`date +%m` => 09 (mês)
`date +%p` => PM
`date +%U` => 36
`date +%y` => 98 (ano)
```

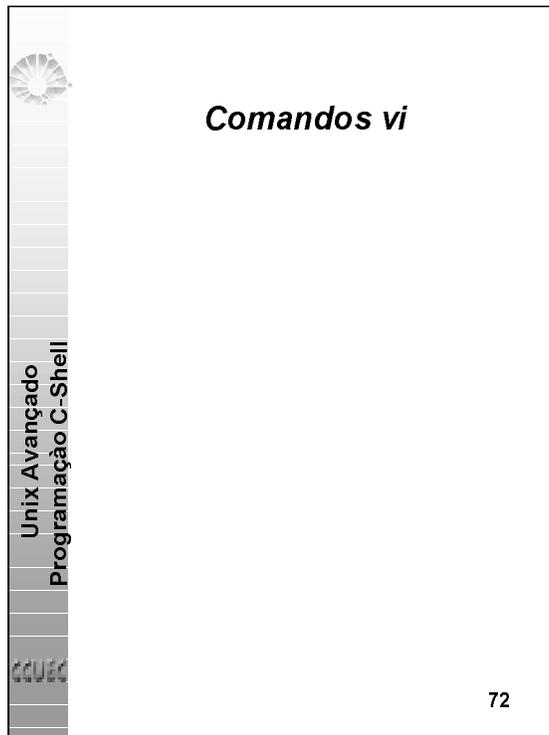
O default do comando date é :

```
%a %b %d %T %Z %Y
```

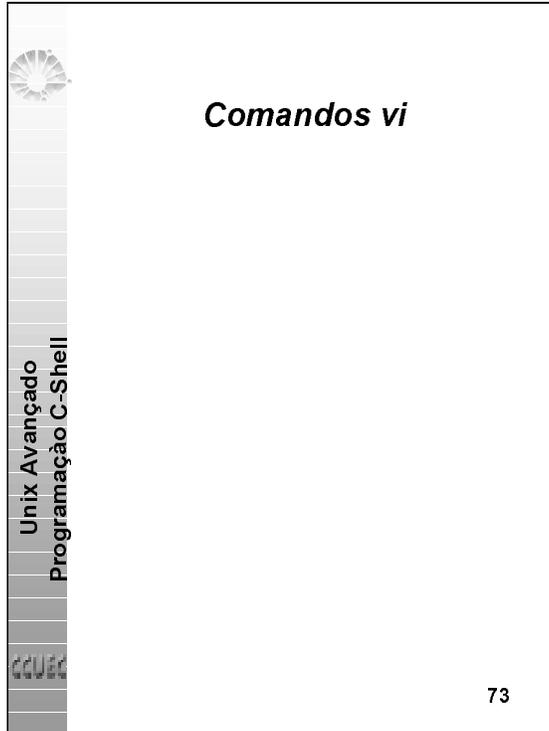


## **Referências**

- ◇ ***UNIX Programação Shell***  
– Lowell Jay Arthur



| <u>INICIANDO O VI</u>       |                                                   |
|-----------------------------|---------------------------------------------------|
| vi filename                 | abre ou cria um arquivo                           |
| vi +18 filename             | abre arquivo na linha 18                          |
| vi +/"frigorífico" filename | abre o arquivo na 1ª. ocorrência de "frigorífico" |
| view filename               | abre o arquivo somente para leitura               |
| <u>COMANDOS DE CURSOR</u>   |                                                   |
| h                           | move à esquerda                                   |
| j                           | move para baixo                                   |
| k                           | move para cima                                    |
| l                           | move à direita                                    |
| w                           | move uma palavra à direita                        |
| W                           | move uma palavra à direita (além da pontuação)    |
| b                           | move uma palavra à esquerda                       |
| B                           | move uma palavra à esquerda (além da pontuação)   |
| Return                      | move uma linha abaixo                             |
| Back Space                  | move um caracter à esquerda                       |
| Space Bar                   | move um caracter à direita                        |
| H                           | move para o início da tela                        |
| M                           | move para o meio da tela                          |
| L                           | move para o fim da tela                           |
| Ctrl-F                      | avança uma tela                                   |
| Ctrl-D                      | avança meia tela                                  |
| Ctrl-B                      | retorna uma tela                                  |
| Ctrl-U                      | retorna meia tela                                 |

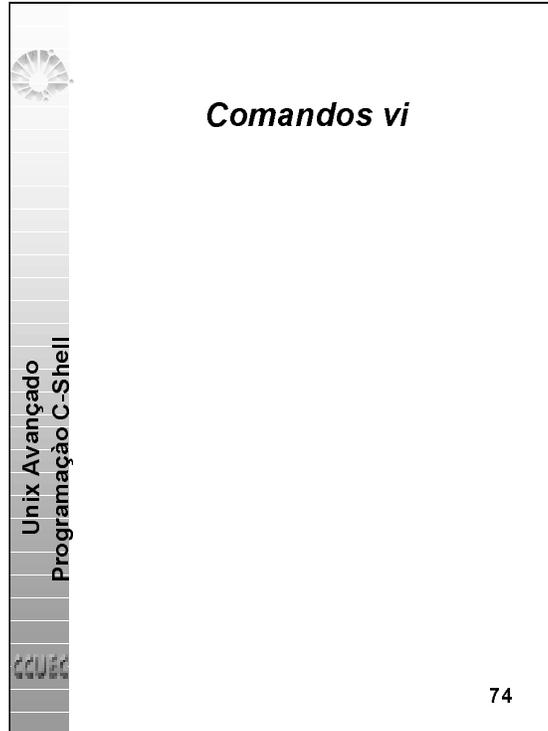


### **INSERINDO CARACTER E LINHAS**

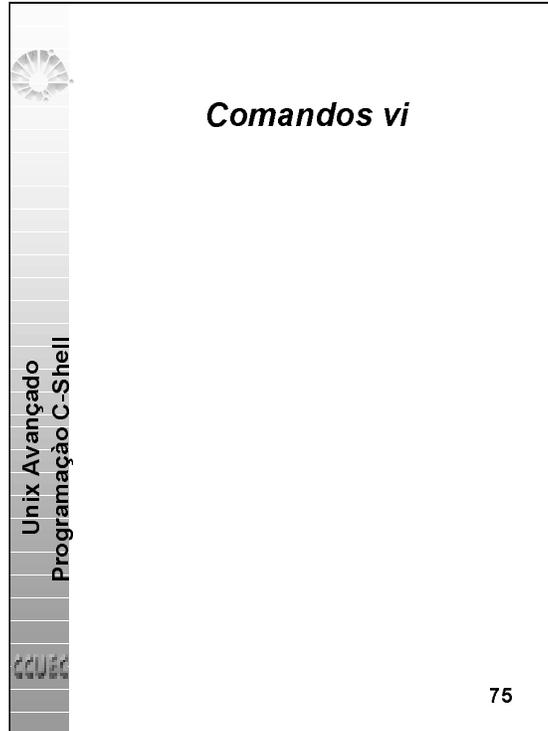
|   |                                                               |
|---|---------------------------------------------------------------|
| a | inserir caracter à direita do cursor                          |
| A | inserir caracter à direita do cursor e sinaliza fim de linha  |
| I | inserir caracter à esquerda do cursor                         |
| l | inserir caracter à esquerda do cursor e sinaliza fim de linha |
| o | inserir linha abaixo do cursor                                |
| O | inserir linha acima do cursor                                 |

### **ALTERANDO O TEXTO**

|          |                                                                        |
|----------|------------------------------------------------------------------------|
| cw       | altera palavra ( ou parte da palavra à direita do cursor               |
| cc       | altera linha                                                           |
| C        | altera parte da linha à direita do cursor                              |
| s        | substitui string que o cursor está posicionado                         |
| r        | repete string que o cursor está com um outro caracter                  |
| r-Return | para linha                                                             |
| J        | junta a linha corrente com a linha acima                               |
| xp       | muda o caracter que o cursor está posicionado com o caracter à direita |
| -        | altera letra (upper ou lower)                                          |
| u        | desfaz o comando anterior                                              |
| U        | desfaz todas as alterações da linha                                    |
| :u       | desfaz o comando anterior da última linha                              |



|           | <b><u>DELETANDO TEXTO</u></b>                            |
|-----------|----------------------------------------------------------|
| x         | deleta caracter                                          |
| dw        | deleta palavra (ou parte da palavra à direita do cursor) |
| dd        | deleta linha                                             |
| D         | deleta parte da linha à direita do cursor                |
| :5,10 d   | deleta da linha 5 à linha 10                             |
|           | <b><u>COPIANDO OU MOVENDO TEXTO</u></b>                  |
| yy ou Y   | marca linha a ser copiada                                |
| p         | copia a linha marcada abaixo da linha corrente           |
| P         | copia a linha marcada acima da linha corrente            |
| dd        | deleta linha (em vez de mover)                           |
| :1,2 co 3 | copia as linhas 1-2 e coloca-as depois da linha 3        |
| :4,5 m 10 | move as linhas 4-5 e coloca-as depois da linha 10        |
|           | <b><u>SETANDO LINHA NUMERADA</u></b>                     |
| :set nu   | mostra as linhas numeradas                               |
| :set nonu | inibe a numeração das linhas                             |
|           | <b><u>PROCURANDO UMA LINHA</u></b>                       |
| G         | vai para a última linha do arquivo                       |
| 21G       | vai para a linha 21                                      |



### **PROCURANDO E ALTERANDO**

|                                       |                                                        |
|---------------------------------------|--------------------------------------------------------|
| /string/                              | procura a string                                       |
| ?string?                              | procura a string no texto acima                        |
| n                                     | procura a próxima ocorrência da string                 |
| :g/search-string/s//replace-string/gc | procura e altera, consultando antes de cada ocorrência |

### **LIMPANDO A TELA**

Ctrl L      limpa a tela

### **INSERINDO UM ARQUIVO NUM ARQUIVO**

:r filename      inseri o arquivo depois do cursor  
:34 r filename      inseri o arquivo após a linha 34

### **SALVANDO E CANCELANDO**

:w      salva as alterações (no buffer)  
:w filename      grava o buffer no arquivo  
:wq ou :zz      salva as alterações e sai do vi  
:q!      sai do vi sem salvar as alterações