PROGRAMAÇÃO

SHELL

Autoras:
Cristiana Munhoz Eugênio
Lilliam Cássia Ciani Palermo

Universidade Estadual de Campinas
Centro de Computação
Versão: 6 - Fevereiro 2000

Colaboradores - Agradecimentos Especiais:
Cristina Maria Zanini
Rubens Queiroz de Almeida

O que é Shell ?

- programa que conecta e interpreta os comandos
- linguagem de programação completa - interpretada
  - possui variáveis;
  - construções condicionais e interativas;
  - ambiente adaptável ao usuário;
- é uma das linguagens originais de quarta geração (4GL).

Simplicidade do Shell

- Pipeline
- Sistemas de arquivos
- Sistema operacional UNIX

Notas:

- O programa *shell* interpreta os comandos que você digita quando trabalha com o sistema operacional e 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 portável entre Unix'es de diferentes plataformas e fornecedores.

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:

- um procedimento complexo usando muitas linhas de comando;
- um procedimento do qual todos os usuários poderão beneficiar-se;
- um comando simples usado inúmeras vezes;
- uma tarefa numa data planejada;
- integrar informações de vários sistemas existentes;

Notas:

- 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 começa a usar o *Shell* interativamente e descobre que alguns processos exigem muita digitação, é hora de aprender sobre a programação *Shell*. Deve-se colocar esses comandos interativos em um arquivo executável. 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 sem qualquer esforço ?
- A composição de sistemas a partir de programas do Shell realizará quase todas as tarefas necessárias. Em vez de esperar meses ou anos para a “solução perfeita”, aplicações protótipo do *Shell* podem ser desenvolvidas, melhoradas e implementadas. Quando necessário, elas poderão ser utilizadas como especificações de requisitos para o desenvolvimento de um sistema “real”.

Produtividade

- Linguagem interpretada - não compilada

- Um programador médio pode duplicar ou triplicar sua produtividade com o uso do Shell

- Comparação de Bruce Cox (pai do Objetive C)

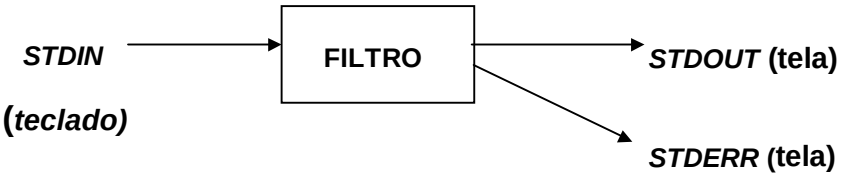
shell	1 linha de código
linguagem orientada a objeto	10 linhas de código
linguagem C	100 linhas de código

Notas:

- 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 derivar desses benefícios. Mas, é preciso apenas alguma inventividade para tornar-se mais eficiente e produtivo.

Filtros



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

- **cat**

- **cat** arquivo

- **sed -e** "s/shell/Shell/g" capit1

  O comando **sed** abrirá o arquivo "capit1" como *stdin* e passará o arquivo para *stdout* (terminal) enquanto muda todas as ocorrências de "shell" para "Shell".

Notas:

- Uma chave para transformar dados brutos em informação útil é filtrar os dados estranhos.
- Você pode pensar na maior parte dos comandos do Shell como filtros.
- Cada comando também possui duas saídas:
  - saída-padrão --> ***stdout***
  - saída-de-erro --> ***stderr***
- Cada comando filtra dados da entrada-padrão ou transforma-os de algum modo, passando-os à saída-padrão. Quaisquer erros encontrados são passados à saída-de-erro.
- Alguns filtros extraem apenas os dados que você deseja ver, outros incluem ou alteram os dados conforme suas instruções.

Redirecionamento

- Criar ou anexar arquivos;
- Usar arquivos existentes como entrada para o Shell;
- Reunir dois fluxos de saída;
- Usar parte do comando shell como entrada.

Operadores:

< *arquivo*	direciona um arquivo para *stdin*
> *arquivo*	direciona *stdout* para uma nova saída
>> *arquivo*	anexa stdout a um arquivo existente
>>&	anexa *stderr* a *stdout*

1> arq_ok **2>** arq_erro
direciona resultado do shell p/ arq_ok e msg de erros para arq_erro

<< delimitador direciona *stdin* do shell até o delimitador

Notas:

- Você pode usar o redirecionamento de E/S para mudar a direção de ***stdin/stdout e stderr.***
  Exemplo:
  sed -e "s/shell/Shell/g" < capit1 > novocapit1
  - O *stderr* ainda estaria direcionado para o terminal

- Redirecionar *stderr* para um arquivo pode ser ocasionalmente útil para depurar um comando Shell. Para combinar *stderr* com *stdout*, use :

Exemplo:
sed -e "s/shell/Shell/g" capit1 1> novocapit1 2> erro

Sintaxe: ***comando argumentos >>& arqsaida***
- O arquivo de saída terá todos os dados da saída-padrão e da saída-de-erro criados pelo comando do shell.
- Se o arquivo **novocapit1** não existir, então o Shell o criará.
- Se existir, o Shell anexará o texto no arquivo.

- mail lilliam < /sistemas/acad/jobs/MA31C

- **<<** : 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!
user anonymous password maria@unicamp.br
cd /tmp
get apostila.ps
EOF!
```

Conduto

- Conecta *stdout* de um comando à *stdin* de outro
- O resultado da execução de um comando pode ser utilizado como parâmetro de entrada na execução de outro comando
- Meio de condução para transportar dados de um comando para o outro
- Exemplos:

  grep matricula /sistemas/acad/jobs/* **| wc -l | sort**

  ls -la | grep -i Jan **"**é diferente de**" ls -la**

Notas:

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

Construção de um Shell Script

- Edite um arquivo ➔ vi meu-programa
- Esse arquivo deve conter na primeira linha:
  - **#! /bin/ksh**
  - # ➔ utilize para comentar uma linha
- **/bin/rm** arquivo
  É recomendável passar o "caminho" (path) dos comandos
- **\rm** arquivo
  Garante encontrar a primeira ocorrência no *path*.
- As shells devem estar sempre num diretório comum, por exemplo: /usr/sys/nome-sistema
- Arquivos de controle que são gerados pela shell devem estar num diretório temporário, separado das shells.

Notas:

- #! /bin/ksh ➔É utilizado para que o Unix saiba que sua shell é korn *shell.*
- É recomendável passar o "caminho" do 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 *scape* antes do comando, por exemplo **\rm.** Garante que você estará utilizando a 1ª. ocorrência encontrada no path definido no seu **.kshrc**

Caracteres especiais do Shell

- **“ “** (aspas duplas)
  Mostra a cadeia de caracteres citada entre ", substituindo o nome da variável pelo seu conteúdo.

  Ex: dia_da_semana**=**Segunda-feira
  echo “ Hoje é $dia_da_semana **”**
  resultado : Hoje é Segunda-feira

- **‘ ‘** (aspas simples)
  Mostra a cadeia de caracteres citada entre ', **sem** substituir o nome da variável pelo seu conteúdo.

  Ex: dia_da_semana**=**Segunda-feira
  echo ' Hoje é $dia_da_semana **'**
  resultado : Hoje é $dia_da_semana

- **` `** (crase)
  O resultado do comando é usado em output.

  Ex: data**= `date +%D `**

- **** (barra invertida)
  Transcreve um caracter especial

Notas:

- A utilização de caracteres especiais dentro de uma citação dará problemas. Para isso utilize o recurso da “barra invertida”.

Caracteres especiais do SHELL

- **;** (vírgula)
  Separador de comando sequencial.

  Ex: date; ksh meu_shell**; date >** /tmp/result **&**

- **?** (ponto de interrogação)
  Combina com qualquer caracter isolado .

- ***** (asterisco)
  Combina com qualquer cadeia de caracter.

- **[a-z]***
  Combina com algo que consista de caracteres alfabéticos minúsculos .

- **^**string
  Procura a string somente no começo da linha

- string**$**
  Procura a string somente no fim da linha

Notas:

Exemplos:

Suponha que no diretório corrente exista os seguintes arquivos:

capit1 , capit2, capit3, capit4 e capitulo

```
ls capit*       (aparecerá os 5 arquivos)
ls capit?       (capit1, capit2, capit3, capit4)
ls capit[1-3]   (capit1, capit2, capit3)
```

- os caracteres "?", "*" e "[a-z]" podem ser usados no meio de uma string

  ls cap?t

Tipo de Variáveis

- **Global:**

  Seta variáveis de ambiente ou variáveis globais. Elas existem enquanto a sessão estiver aberta.

  Sintaxe: **export** nomevar=valor

  Exemplos:

  export TERM=vt100 (variável de ambiente)
  export DBDATE=Y4MD- (variável de ambiente)
  export matricula**=**20444 (variável global)

- **Local:**

  Seta variáveis locais. Elas existem somente durante a execução da Shell.

  Sintaxe: nomevar=valor

PARA SABER QUAIS VARIÁVEIS ESTÃO SETADAS E QUAIS SÃO SEUS VALORES, UTILIZE O COMANDO SET.

Notas:

- ***EXPORT:***
  - Seta variáveis de ambiente ou globais. Existem enquanto a sessão estiver aberta.
  - export PS1="`hostname`{`whoami`}$ "
    magda{maria}$

- **NÃO** existe espaço em branco entre o nome da variável e seu conteúdo
  - nome=Rogério

- Nomes de Variáveis **não** pode conter **hifen.**
  - nota-aluno ➔ dará erro **X**

- É permitido o uso do **underline** para nome de variáveis**:**
  - minha_var

Definição de variáveis

- **String**

  - Atribuindo valores:

    Sintaxe : **nome=conteúdo**

    Exemplos:
    msg_ftp**=“**221 Quit command received.**”**
    dir_log=~liliam/tmp
    meus_arqs**=`ls -la`**

  - Concatenando strings:
    Exemplos:
    nome=Li
    nome1=lliam
    conc=$nome$nome1
    echo $conc → Lilliam

NOMES DE VARIÁVEIS NÃO PODEM CONTER *HÍFEN.*

Notas:

Definição de variáveis

- **String**

  - Utilizando string:
    Utilize "$" para substituir o nome da variável pelo seu conteúdo.

    Sintaxe : **$variável**

    Exemplos:
    cabec=“ Relatório do ano $ano”
    dir_log=/tmp ; ls $dir_log
    aluno=${nomes[$cont]}
    parm1=$1
    ls arq$var
    dia=Segunda-feira ; echo “ Hoje é dia $dia “

Notas:

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

Definição de variáveis

- **Inteiro**

  - Atribuindo valores do tipo ***INTEIRO***
    Exemplos:
    let cont=1 ou cont=1
    let soma=$cont+2
    let x=' 1 + 4 ' ou **let** x=1+4
    let x=$x+1

NÃO DEIXAR ESPAÇO ENTRE O NOME DA VARIÁVEL E O SINAL DE IGUAL

NÃO É NECESSÁRIO INICIALIZAR UMA VARIÁVEL COM O COMANDO LET PARA ELA SER DO TIPO INTEIRA, UTILIZE O "LET" APENAS QUANDO VOCÊ QUISER MANIPULÁ-LA .

Notas:

Definição de variáveis

- **Array**

  - Criando arrays:
    Exemplos:
    set -A var a b c
    set -A nomes ana jose maria

  - Incrementando arrays:
    Exemplos:
    nomes**[3]=**joão
    aluno=${nomes[[3]}
    echo $aluno → joão

  - Manipulando arrays:

    - **${var[*]}**
      Mostra o conteúdo do array inteiro

    - **${#var[*]}**
      Mostra o número de elementos (argumentos) de um array.

    - **${variável[n]}**
      Elementos posicionais de um array.

A 1ª OCORRÊNCIA DO ARRAY COMEÇA COM 0

Notas:

- ***set -A var val1 val2 val3 val4***

  - set -A semana Segunda Terça Quarta Quinta Sexta → seta um array chamado *semana* contendo *os dias da semana*

- Exemplos de array :

  set -A nome maria lucia joao
  echo ${nome[*]} → Resp = maria lucia joao
  echo ${#nome[*]} → Resp = 3

- Se o indexador do array ultrapassar o número de elementos existentes, será retornado branco

Edição de string

- **typeset**

  Especifica o tipo do valor da variável

  Sintaxe:

  typeset [opções] v=$var

  Opções:

  - -Ln : ajusta à esquerda. Se "n" é especificado, é preenchido com branco ou truncado à direita no valor de n
  - -Rn: ajusta à direita. Se "n" é especificado, é preenchido com branco ou truncado à esquerda no valor de n
  - -Zn: mesmo que o acima, adiciona 0 ao em vez de branco
  - -l converte para minúsculo
  - -u converte para maiúsculo

  Exemplos:

  alpha=" aBcDeFgHiJkLmNoPqRsTuVwXyZ "

  typeset -L v=$alpha" → "aBcDeFgHiJkLmNoPqRsTuVwXyZ "

  typset -uR5 v=$alpha → "VWXYZ"

  typeset -Z8 v="123.50" → "00123.50"

Notas:

Tratamento de Variáveis

- **testa se a variável tem conteúdo**

-n str: retorna verdadeiro se variável NÃO FOR NULA

-z str: retorna verdadeiro se variável for NULA

```
if [[ -z $matricula ]] then
    echo " variável matricula NÃO inicializada"
fi

matricula=23456
if [[ -n $matricula ]] then
    echo " variável matricula inicializada"
fi
```

Notas:

Tratamento de erro

- **$? - testa se o comando foi executado com sucesso.**

0 executado com sucesso
1 executado sem sucesso

Exemplos:
var=5
echo $? **(= 0)**

echo $var4
echo $? **(= 1)**

FAÇA UM TESTE ANTES POIS DEPENDENDO DO SISTEMA PODE SER O CONTRÁRIO !!!!!!

Notas:

Variáveis de ambiente

- **$USER**
  Conteúdo do comando whoami (quem é o usuário da conta)
- **$PATH**
  Lista de diretórios a serem procurados pelo meus executáveis
- **$HOME**
  Mostra qual é o diretório home
- **$PS1**
  Mostra o que está setado como prompt
- **$SHELL**
  Mostra a shell que está sendo utilizada
- **$TERM**
  Mostra qual o tipo de terminal que está sendo utilizado
- **$$**
  Número do processo do comando corrente

Notas:

- O valor das variáveis ao lado podem ser encontrados ao digitarmos o comando **set.**
- **$$** ➔ Muito utilizado pelos utilitários de mail que gravam o arquivo em /tmp/arq_número_do_processo

Execução de um Shell Script

- Fora do editor vi, na linha de comando:

  $ksh -opções_de_teste meu-shell parâmetros

- Gravar a execução de uma shell script

  $script

  $ksh -opções_de_teste meu-shell

  $ CTRL D

  $ more typescript

- Dentro do editor vi:
  - **:w** ➔ grava o arquivo que está na memória
  - **:!ksh -opções de teste %**

    ! ➔ diz ao vi para executar o comando shell seguinte
    % ➔ diz ao vi para preencher o nome do arquivo corrente

Opções de Teste:

- **x** ➔ gera saída dos comandos executados

Notas:

- Você pode executar o seu programa a medida que for sendo desenvolvido, facilitando também a detecção e correção de erros.
- Entrando no *vi* e criando as primeiras linhas do programa, grave o arquivo (:w) e execute-o sem sair do editor: (:!ksh -opção %)

Tratamento de Parâmetros

- **$#**
  No. de parâmetros

- **$0**
  Parâmetro que contém o nome do programa

- **$1, $2, $3, ...**
  Parâmetros fornecidos

- **$***
  Lista o conteúdo de todos os parâmetros

Notas:

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

```
ksh  meuscript  parm1 parm2
#! /bin/ksh
    if  (( $#  ==  2))   then
      echo " numero de parametros correto "
    fi
```

- ksh meuprogram parm1 parm2
  $0 **$1** **$2**

Principais Operadores

Operação	Numérico	String
igual	==	=
diferente	!=	!=
menor, maior	<, >	<, >
... ou igual	>=, <=	>=, <=
and , or	&&, ||	
soma	+	
subtração	-	
multiplicação	*	
divisão	/	

Notas:

Operadores de Teste

Condição Shell	Verdadeira se ...
-d file	se é diretório
-f file	se é file
-r arquivo	existir *arq*. legível
-w arquivo	existir *arq.* gravável
-x arquivo	existir *arq.* executável
-a arquivo	se arquivo existe
-s arquivo	se arquivo existe e não está vazio
-O arquivo	se o username que está logado é o owner
-G arquivo	se o gid do username que está logado é o mesmo do arquivo

Notas:

- No núcleo de cada estrutura de controle existe um teste condicional.
- O comando ***test*** pode determinar se um determinado nome é um arquivo ou um diretório; se ele pode ser lido, ou se é executável; e se uma cadeia ou inteiro é maior, menor ou igual a outro.
- ***Test***, como qualquer outro comando do Shell, sempre retorna um valor verdadeiro **(0)** ou falso **(1)** na variável *status* do shell.

```
test -r nomearq
echo $?
```

```
if [[ -d arq_com ]]      then
   echo " arq_com é um diretório "
fi
```

Comandos condicionais

- **if - then - else**

- **if ((** expressão aritmética **)) then**
  processa 1
  else
  processa 2
  fi

- **if [[** testes condicionais **]] then**
  processa 1
  else
  processa 2
  fi

PARA COMPARAÇÕES DE TESTES CONDICIONAIS E COMPARAÇÕES DE STRINGS É NECESSÁRIOS TER ESPAÇOS EM BRANCO ENTRE OS COLCHETES E OS OPERADORES

Exemplo**:**

```
if [[   $a    =    $b    ]] then
   echo "variáveis iguais "
fi
```

Notas:

- Exemplos:

Comparação numérica

```
a=5 ;    b=7
if (( $a  ==  $b ))   then
   echo "valores iguais "
else
   echo " valores diferentes"
fi
```

Comparação de testes condicionais

```
if [[  -z $parametros  ]] then
   echo " ERRO - não foi passado parâmetros !!! "
   exit
fi
```

Comparação de string

```
a=inteiro ;    b=metade
if [[ $a  =  $b ]] then
   echo "variáveis iguais "
else
   echo " variáveis diferentes"
fi
```

Comandos condicionais:

- **case**

```
case expressão in
        opção1 )  processa a
                  echo opcao a ;;
        opção2 )  processa b;;
        opção3 )  processa c;;
                * ) echo " Opção inválida " ;;
esac
```

Exemplo:

```
 $ cria_imp -p pcca -ip 143.106.30.9  -n pcca

while (( $# ))
do
     case $1 in

     -ip)  ip_da_impressora=$2  ; shift ;;

     -p)   nome_da_impressora=$2  ; shift ;;

     -n)   nome_da_imp_rede=$2  ; shift ;;
      esac
     shift
done
```

Notas:

- Exemplos:

```
data=`date | cut -c9-10`
texto="Relatorio do mes de"

case $data  in
        01)   echo "$texto Janeiro"      ;;
        02)   echo "$texto Fevereiro"   ;;
    9 | 09)   echo "$texto Setembro"   ;;
        12) ;;
         * )   echo " ERRO !!!!!"   ;;
esac
```

Comandos condicionais:

- **for**

for var **in** expressão
do
comandos que usam $var
done

Exemplos:

```
for  file in *.c
  do
    mv $file `echo $file | tr ´A-Z´ ´a-z´  `
    echo $file
  done
```

Onde *expressão* pode ser:
- resultado de um comando
- valores
- ocorrências de um array

- As instruções dentro do comando for serão executadas tantas vezes quantas forem o nº de elementos pertences à expressão

Notas:

- Exemplos:

Utilizando array:

```
set  -A tabela lilliam  kikinha
for  pessoa in  ${tabela[*]}
     do
       echo  $pessoa
    done
end
```

Utilizando o resultado de um comando*:*

```
for  arquivo in   `grep -l "matr_aluno" *.sqb`
do
   ex +%s/"old_string"/"new_string"/g +wq $arquivo > /dev/null
done
```

Comandos condicionais:

- **while**

```
dir_relatorio=~kikinha/prog
while ((1))
   do
   clear
   echo " \n*  Centro de Computacao* \n"
   echo "Selecione a opcao desejada"
   echo " 1- Tabela de Orgaos "
   echo " 2 - Notice "
   echo " 3 - Sai "
   print -n " → "
   read  RESP
      case $RESP  in
      1)  $dir_relatorio/jsi03e_carga;;
      2)  /usr/local/notice/manager  ;;
      3)  exit  ;;
      *)  print -n " Opcao Invalida - Tecle <enter> "
          read ENTER  ;;
      esac
done
```

Notas:

- Sintaxe:
  while ((expressão))
  do
  done

- Comando **exit:**

  Serve para sair 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

Notas:

- 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*:**

Exemplo:

man nome_comando

- Comandos :

1.	próxima linha	ENTER
2.	próxima página	space bar
3.	página anterior	CTRL B
4.	procura string	/\caracter especial ou /string
5.	continuar a busca	n
6.	cancelar pesquisa	q ou CTRL C

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

TIPO	COMANDO
Diretório	cd ls pwd
	mkdir rmdir
Arquivo	cat cp csplit
	ln mv rm split
Seleção	awk
	cut
	grep
	head
	line
	sed
	tail
	uniq
	wc
Junção	cat
	join
	paste
Ordenação	sort
Transformação	sed
	tr
	dd
Impressão	cat
	echo
Segurança	chmod
Leitura	$<
	touch
	sleep
	exit

Notas:

Os comandos de Diretórios e Arquivos já foram abordados anteriormente. Eles podem ser utilizados sem restrições numa shell.

Exemplos:

```
\rm  arquivo_old >& /dev/null
\ls -la
/bin/cat arquivo
/bin/mv arqvelho arqnovo
/bin/pwd
/bin/mkdir meudir
```

Alguns lembretes:

cd	altera o diretório corrente
ls	lista o conteúdo do diretório
pwd	mostra qual é o diretório corrente
mkdir	cria diretório
mkdir -p	cria árvore completa de diretórios
rmdir	deleta diretório
cp	copia arquivo
cp -p	copia mantendo as características originais
mv	renomeia arquivo
rm	deleta arquivo
basename	mostra o nome do arquivo tirando o path
dirname	mostra o nome do arquivo colocando o path
diff	compara conteúdo de 2 arquivos < 1° arquivo > 2° arquivo
cmp	compara dois arquivos -s (echo $? →traz 1 se diferente) (echo $? →traz 0 se igual)

Comandos de Impressão

- **echo ou print**

  Escreve argumentos em stdout.

  Sintaxe:

  echo [opções] string

  Opções:

  - -n : não adiciona uma nova linha no stdout antes de imprimir a string, deixando o cursor na linha da mensagem escrita. **FUNCIONA APENAS COM PRINT**

Notas:

- ***echo***

  Exemplos:

  1. print -n "Entre com a opção desejada => "
     Entre com a opção desejada ➔__
  2. echo "Hoje é dia $dia"
  3. echo **"**Bom dia**"**
  4. echo "Os arquivos são: " *
  5. echo "Essa pessoa $nome eh muito legal" > arq
  6. echo * → o mesmo que o comando ls *

Comandos de Impressão

- **cat**

Concatena e imprime. O comando *cat* não altera dados, ele toma a stdin e a reproduz no stdout. Possui também a opção de varrer o arquivo mas não pára a cada tela do terminal.

Sintaxe:

cat [opções] stdin

Opções:

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

file:

Notas:

- ***cat***

Exemplos:

1. ***cat*** capt1 capit2 > livro
   livro irá conter capit1,2

2. ***cat*** capt3 >> livro
   anexe capit3 no livro

3. ***cat*** capit1
   lista capit1 no terminal

4. ***cat*** -n capit1
   lista capit1 numerando as linhas

5. ***cat*** > arq1
   direciona o stdin para o terminal e stdout para arq1, assim tudo que for digitado será gravado no arq1. Para encerrar tecle CTRL D.

Comando de Segurança

- **chmod**

  Altera a permissão do arquivo.

  Sintaxe:

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

  Opções:

  - quem: **u**(usuário), **g**(grupo), **o**(outros), **a** (todos)
  - operação: **+(**atribuir), **-(**remover), **=**
  - acesso: r(read), w(write), x(execução)
  - **-R** : altera a permissão dos arquivos e diretórios descendetemente.

Para tornar um shell script executável, é necessário alterar a permissão do arquivo.

Comando de Leitura

- **read**

  Direciona o terminal como stdin, ou seja , captura o que foi digitado no teclado.

  Sintaxe:

  read variável

Notas:

- ***chmod***

  Se a sua shell for executar o comando FTP você deverá alterar a permissão do arquivo que contém o usuário e a senha para somente o dono ler, gravar e executar.

  Exemplos:

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

- Exemplo:

```
echo  -n  “Digite a opcao:   “
read opcao
if  (($opcao  >  9))  then
   echo  “Opcao Invalida”
fi
```

Comando de Ordenação

- **sort**

  Ordena e combina múltiplos arquivos.

  Sintaxe:

  sort [opções] stdin .....

  Opções:

  - **-b**: ignora espaços em branco para encontrar a 1ª ou última coluna do campo
  - **-d**: sorteia usando ordem alfabética. Somente letras, dígitos e espaços são considerados na comparação.
  - **-f**: altera todas as letras minúsculas para maiúsculas antes da comparação.
  - **-m**: junta todos arquivos de entrada num só. Os arquivos de entrada devem estar sorteados.
  - **-n**: ordena o arquivo numericamente
  - **-r** : ordena o arquivo na forma decrescente
  - **-u**: deleta linhas duplicadas
  - **-t**: define delimitador
  - **-k [cpo ini.col ini] [opção] [, cpo fim.col fim] [opção]** : define chave do sort
  - **-o:** especifica que o stdout vai ser o mesmo arquivo do stdin

Notas:

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

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

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 :

1. sort -nru arquivo
2. sort -k2,2f -k3 arquivo
3. sort -k1.1 -k2.4 arquivo
4. sort -rt' ' -k3b -k4 arquivo
5. sort -m arq1 arq2
6. sort -d arq1 -o arq1_sort

Comandos de Seleção

- **line**

  Copia uma linha do stdin e grava no stdout.
  Sintaxe:

  line stdin > stdout

  stdin - arquivo
  stdout - arquivo ou terminal

- **head**

  Seleciona as 10 primeiras linhas de um arquivo.
  Sintaxe:

  head [opção] stdin > stdout

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

  head -c3 /etc/passwd

- **tail**

  Seleciona as 10 linhas finais de um arquivo
  Sintaxe:

  tail [opções] stdin > stdout

  Opções:
  - -n : seleciona as n últimas linhas do stdin
  - -f : mostra as linhas que estão sendo adicionadas no arquivo.

  tail -f /var/adm/messages

Notas:

Comandos de Seleção

- **sed**

  Edita e altera dados.

  Sintaxe:

  sed [opções] stdin > stdout

  Opções:

  - -e "script" : edita e altera dados

    Onde script pode ser:

    "s/string antiga/string nova/g"
    "/string pertencente à linha a ser deletada/d"
    "s/string a ser deletada//g"
    "s/ *$//g" → deleta espaço em branco até o fim da linha
    1d arquivo → deleta a 1ª linha do arquivo

  - -f arquivo : contém os scripts que devem ser executados. Não colocá-los entre aspas duplas.

  - -n x,zp arquivo : Seleciona da linha "x" até a linha "z" do arquivo especificado.

  .

- Não usar stdin como stdout
- **Pode ser alterado o delimitador do sed de "/ " para qualquer outro caracter "+", "(", "=", etc**

Notas:

- 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:

  1. ***sed -e*** **"**s/café/chá/g**"** entrada > entrada_alterada
     Altera a palavra **café** para **chá** em todas as ocorrências (g)
  2. ***sed -e*** **"**/João/d**"** entrada > saída
     Deleta todas as linhas do stdin que contém a palavra **João**
  3. ***sed -e*** **"**s/devedor//g**"** entrada
     Deleta a palavra
  4. ***sed -f*** arqcom entrada > arqsaida
     Executa os comandos que estão no arquivo arqcom.
     arqcom: s/shell/Shell/g
     s/Terça/terça/ → altera 1 ocorrência por linha

     OBS: NÃO USAR ASPAS DENTRO DO ARQUIVO

  5. ***sed -e*** "s/cha/café" ***-e*** "s/limao/mate/" arq_sala > arqsaida
  6. **sed -n 5,10p arquivo**
     imprime da linha 5 até a linha 10 do arquivo.
  7. nlin=`cat arquivo | wc -l `
     lin=1
     while (($nlin <= $lin))
     do
     sed -n "$lin"p arquivo
     let lin=lin+1
     done
     Percorre linha a linha do arquivo

Comandos de 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

- **find**

  Seleciona arquivos, a partir do diretório especificado. É recursivo.

  Sintaxe:

  find dir opções

  Opções:
  - **-type**: especifica o tipo de arquivo (f, d ou l)
  - **-name** : informa o nome do arquivo
  - **-print**: lista os arquivos (incluindo o path) que satisfazem o critério.

Notas:

Exemplos:

- find . -name "*.HTML" -print → lista todos os arquivos que terminam com .HTML
- find . -type f -print → lista todos os arquivos do tipo file a paritir do diretório corrente (.)
- find /home -type d -print → lista todos os arquivos do tipo diretórioa partir do diretório /home
- find . -name "*.sqb" -print | xargs grep "matr_aluno" | wc -l

Comandos de Seleção

- **grep**

Seleciona linhas usando um único critério de seleção.

Sintaxe:

grep [opções] string stdin

Opções:

- **-c** : conta o nº de vezes que a string aparece.
- **-i** : ignora uppercase
- **-l** : lista somente o nome dos arquivos que contém a string procurada.
- **-n** : numera linhas que contém a string procurada.
- **-v** : lista todas as linhas do arquivo exceto às que contem a string.
- **^letra** : lista todas as linhas que iniciem com a letra ou string.
- **-w :** lista somente as linhas que tem exatamente a string passada

String pode ser:

- $variável
- "palavra composta"
- palavra
- "metacaracteres" (* ? [a-z])
- "stringmetacaracter"

Notas:

- 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:

1. grep "Mar*" arquivo
2. grep Maria arq*
3. grep -l Maria arquiv*
4. grep -n Maria arquivo
5. grep $variável arquivo
6. grep -v "Maria Amélia" arquivo
7. grep "[a-z]" arquivo
8. grep ^R arquivo
9. grep -c Chico arquivo
10. grep -i maria arquivo
11. ps -ef | grep maria | grep \d → não aparece a linha do grep anterior

Comandos de Seleção

- **cut**

  Seleciona colunas.

  Sintaxe:

  cut [opções] stdin

  Opções:

  - -cx-y : define coluna inicial(x) e coluna final(y)
  - -dCaracter : define delimitador
  - -fx,y : quais campos serão selecionados
  - -s : suprimi linhas que não contém delimitador.

  Somente pode ser usado junto com a opção -d

OBS: O comando *cut* opera em colunas

- **wc**

  Conta linhas, palavras e caracteres de arquivos.

  Sintaxe:

  wc [opções] stdin

  Opções:

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

Notas:

- 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:

1. cut -f1,5 -d: arquivo
2. cut -c1-72 arquivo
3. cut -d: -s -f3,5 arquivo

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

- wc

Exemplos:

1. grep -l Maria arqu* | wc -l
2. cat arq l wc -l

Comandos de Seleção

- **split**

  Divide o stdin em arquivos menores contendo um número específico de linhas .

  Sintaxe:

  split [opções] stdin [prefixo]

  Opções:

  - **-** qtd. linhas: especifica o nº de linhas de cada arquivo de saída. Default 1000 linhas por arquivo.
  - prefixo: prefixo dos stdouts criados. O sufixo é aa,ab até zz. Default prefixo é x.

Notas:

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

  Exemplo:

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

  2. split -5 arquivo capit (gera arquivos capitaa, capitab,...)

Comandos de Seleção

- **csplit**

Separa os arquivos sempre que encontrar uma combinação entre seus argumentos e o conteúdo do arquivo. **Não funciona no FreeBSD**

Sintaxe:

csplit [opções] stdin [argumentos]

Opções:

- -f prefixo: prefixo dos stdouts criados. O sufixo é 00,01 até 99. Default xx
- argumentos:
- "/argn/" : cria um arquivo por argumento (inclusive). O arquivo **prefixo**00 contém as linhas anteriores ao 1º argumento.
- "%arg%" : cria um arquivo contendo os dados a partir do argumento.
- -k: não desfaz a criação dos stdouts caso o no. de stdouts especificado seja maior que o qtd de stdin

Notas:

- ***csplit***

Exemplos:

- O arquivo livro contém:
  Introdução,
  Capitulo1,
  Capítulo 2,
  Resumo.

1. csplit -f capit livro "/Capitulo /" "/Capitulo /"

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

2. csplit -f capit livro "%Capitulo 1%"
   resultado: capit00 (contém Capítulo 1 Capítulo 2 e Resumo)
3. csplit livro "/Capitulo 1/"
   resultado: capit00 (contém Introdução)
   capit01 (contém Capítulo 1, 2 e Resumo)

4. csplit -k livro "/Capitulo /" (9)
   resultado: será criado até 10 stdout

Comandos de Transformação

- **tr**

  Transforma caracter por caracter.

  Sintaxe:
  tr ‘caracter velha‘ ‘caracter nova‘ stdin > stdout

o comando TR é melhor que o DD, quando for converter strings.

- **dd**

  Converte e copia arquivo.

  Sintaxe:
  dd if=stdin of=stdout conv=[opções conv]

  Opções do conv:
  - lcase: converte arquivo para minúsculo
  - ucase: converte arquivo para maiúsculo
  - ascii: converte ebcdic para ascii
  - ebcdic: converte ascii para ebcdic
  - ibm: converte ascii to versão ebcdic da IBM

  Não aceita passar path

- **sed**

  Visto anteriormente.

Notas:

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

  Exemplos:

  1. tr ‘(A-Z)’ ‘(a-z)’ arq1 > arq1min

  2. tr ‘(:)’ ‘(\t)’ cad1 > cad1t

  3. for file in `ls`
     do
     mv $file `echo $file | tr ‘(A-Z)’ ‘(a-z)’ `
     echo $file
     done

Comandos de Junção

- **join**

  Junta dois arquivos, combinando linha a linha. Os arquivos devem estar ordenados (comando sort).

  Sintaxe:

  join [opções] arquivo1 arquivo2

  Opções:

  - **-t**Caracter: define delimitador
  - **-a nº arquivo**: produz uma linha de saída para cada linha do arquivo especificado pelo **nº arquivo**, mesmo para os campos do join que não casam com nenhuma linha do outro arquivo de entrada.
  - **-o nºarquivo.campo** : As linhas de saída são compostas pela lista dos campos especificados na variável **nºarquivo.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).
  - **-e string:** altera campos em branco do output pela string especificada

Notas:

- 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***).

  Exemplos:

  arquivo1: Aldo:HC
  Bruno:IMECC
  João:CCUEC

  arquivo2: Aldo:enfermeiro:7234
  João:digitador:7111
  Ricardo:médico:7323

  1. join -t: arquivo1 arquivo2

  2. join -a1-t: arquivo1 arquivo2 > arqsaida

  arqsaida: Aldo:HC:enfermeiro:7234
  Bruno:IMECC:_____
  João:CCUEC:digitador:7111

  3. join -o 1.1 1.2 2.3 -t":" arquivo1 arquivo2

  arqsaída: Aldo:HC:7234
  João:CCUEC:7111

  4. join -1 2 -2 2 arquivo1 arquivo2

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

Comandos de Junção

- **paste**

  Anexa múltiplos arquivos coluna por coluna.

  Sintaxe:

  paste [opções] stdin1 stdin2

  Opções:

  - **-d**Caracter: define delimitador
  - **-s** : junta linhas subsequentes do arquivo

Notas:

Exemplos**:**

1. **paste -d "/" arq1 arq2**

   Aldo:HC/Aldo:enfermeiro
   Bruno:Imecc/Ivo:digitador

2. **paste -s -d "/" arq1 arq2**

   Aldo:HC/Bruno:Imecc
   Aldo:Enfermeiro/João:digitador

Comandos de Junção

- **awk**

  Linguagem de processamento e procura de padrões.

  Encontra linha(s) no(s) arquivo(s) de entrada que combinam com um padrão especificado e então executa as ações definidas.

  Sintaxe:

  awk [opções] '/padrão/' {ação} stdin

  Opções:

  - -Fcaracter: define delimitador

  A ação pode ser:
  -f pgm: arquivo que contém as ações a serem executadas pelo comando awk.
  'pgm': são as ações a serem executadas pelo awk.

Notas:

Exemplos:

1. awk -F: '{print $5 $1}' arq_fun
2. awk -F: '{print $2 " " $1}' arq_fun
3. awk -F: {'$1 ~/e|a/'} arq_fun
4. awk -F: '{print NR,NF,$0}' arq_fun
5. awk -F: '{print $1 > "matr"; print $2 > "nome"}' arq_fun
6. awk -F: '{printf "%15s %10s\n", $1, $3}' arq1
7. awk -F: '/smith+ern/' arq1
8. awk -F: '/amel??/' arq_fun
9. awk -F: '{printf ("meu nome eh %s e meu ramal %5d\n " , $2,$4)} arq_fun
10. awk -F: '$3 ~/Oct/ {print $2 $3} ' arq1
11. awk 'NR == '$cont' {print $0} ' arq → le linha por linha de um arquivo
12. awk ($1 == " '$var' ") [print $0}' arq

Comandos de Junção

- **AWK** (Cont.)

  Tipos de padrões de pesquisa:

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

Notas:

Comandos de Junção

- **AWK** (Cont.)
  - **‘$1 ~/char1/’** : grava em stdout todos os registros que contém no 1º campo o caracter char1 ou uma string .
  - **‘$1 ~/^char1/’** : grava em stdout todos os registros que **não** contém no 1º campo o caracter char1 ou uma 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 à 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 a string1 e o 2º é igual a string2.”

Notas:

Comandos de Junção

- **AWK** (Cont.)

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 da string **ra** e a 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.

sequencia	significado
\n	nova linha
\b	backspace
\t	tab

Notas:

Comandos de Junção

- **AWK** (Cont.)

Concatenação de campos:
'{ print $1 = $1 $2}' : o campo 1 é a concatenação do campo 1 com campo 2

Imprimir linha em branco:
'{print " "} '

Impressão formatada de campos variáveis ou constantes :
'{printf "%15s\n" , $1}' : imprime o campo 1 por linha formatando-o utilizando 15 posições alinhado à direita.

'{printf "%20d %-5s\n", $3+$2 , $1}' : imprime o resultado da soma do campo2 e do campo3 e o campo 1.

'{printf "%20s %4d\n" , "NOME" , RAMAL"}' imprime as constantes NOME e RAMAL no formato definido pelo printf.

Notas:

- Tabela de caracteres de controle para o printf

caracter	expressão
c	caracter ASC II
d	decimal inteiro
e	[-] d.dddddd [+-] dd
f	[-] ddd.dddddd
s	string
%	imprime %

- Formatos válidos para o printf:

Fmt	$1	printf '{fmt, $1}'
%c	97	a
%d	97.5	97
%5d	97.5	bbb97
%e	97.5	9.750000e+01
%f	97.5	97.500000
%7.2f	97.5	97.50
%06d	97	000097
|%s|	Janeiro	|Janeiro|
|%10s|	Janeiro	| Janeiro|
|%-10s|	Janeiro	|Janeiro |
|%.3s|	Janeiro	|Jan|
|%10.3s|	Janeiro	| Jan|
|%-10.3s|	Janeiro	|Jan |
%%	Janeiro	%

Comandos de Junção

- **AWK** (Cont.)

* **Begin**

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

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

* **End**

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

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

Notas:

Comandos de Junção

- **AWK** (Cont.)

∗ **Instruções condicionais:**

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

```
BEGIN { FS = ":" }
if ($1 != prev {
      print " "
      prev = $1
      }
```

∗ **Variáveis especiais:**

NR: número da linha corrente
NF: quantidade de campos da linha
FS = "caracter": define delimitador

Notas:

Date

Algumas opções de date:

- `date +%a` => Tue ; `date +%A` => Tuesday
- `date +%b` => Sep ; `date +%B` => September
- `date +%c` => Tue Mar 31 15:58:44 2000
- `date +%d` => 31 (com 0 ex: 01)
- `date +%e` => 31 (sem 0 ex: 1)
- `date +%D` => 09/31/00(mês/dia/ano)
- `date +%j` => 254 (dia do ano)
- `date +%m` => 03
- `date +%p` => PM
- `date +%U` => 36 (no. da semana)
- `date +%y` => 00 (ano)
- data= `date +%d-%m-%Y` => 22-02-2000
- hora=`date +%H%M%S`

Notas:

Dicas de programação

1. Se a sua shell for fazer FTP, alterar a permissão do arquivo que contém o usário e a senha para “700”. **Este é um ítem obrigatório por medidas de segurança.**

2. Cada sistema tem um usário para acessar o IBM (mainframe) via FTP. Se o objetivo deste ftp é acessar alguma visão DB2, deverá ser solicitado ao DBA autorização para acesso às visões.

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

Exemplo: grep $1 arq1 | sort

4. Utilize condutos ao invés de arquivos temporários para melhorar a eficiência da sua shell

Com arquivo temporário:

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

Com conduto: cut -f1,f5 arq1 | sort

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

Notas:

Dicas de programação

6. Sempre utilize delimitador de campos nos arquivos. O delimitador deve ser único e simples e deve ser utilizado entre colunas.

7. Utilize delimitador único e diferente para subcampos, como por exemplo data (dia/mês/ano).

8. Organize os dados para maior eficiência da máquina: primeiro chaves primárias, depois chaves secundárias e depois os itens de dados.

Notas:

Executar comandos vi em modo batch

- **Utilizando o editor vi:**

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

Comandos_vi:

/FIM
dd
^[
:x
```

Notas:

- vi
  - Se o arquivo existir, touch atualiza a data do arquivo, se não existir touch cria o arquivo
  - echo, escreve no stdout, no caso no arquivo criado por touch
  - ^[- representa a tecla ESC (digitar CTRL v CTRL [ )
  - :x - sai do editor vi e salva o arquivo
  - 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

- ed
  - editor ed executa os comandos definidos abaixo dele até encontrar o delimitador, neste exemplo, EOF!
  - s - substitui
  - g - global
  - w - salva as alterações no arquivo aberto por ed
  - q - sai do editor ed

Como utilizar ISQL num SHELL

- /usr/informix/bin/isql -s bgql01 -qr delvtab53 > \
  logerror_delete

- /usr/informix/bin/isql
  Path do isql

- -s bgql01
  Banco de dados que está a visão/relação que será acessada.

- -qr
  Opções do isql que serão executadas. No caso, q (query) e r(run)

- delvtab53
  Arquivo que contém os comandos isql

- \
  Continuação da linha comando

- logerror_delete
  Se ocorrer algum erro, será gravado neste arquivo

Notas:

R E F E R Ê N C I A S

Comandos básicos do vi:

INICIANDO O VI	
vi filename	abre ou cria um arquivo
vi +18 filename	abre o arquivo na linha 18
vi +/"teste" filename	abre o arquivo na 1ª. ocorrência de "teste"
view filename	abre o arquivo somente para leitura

COMANDOS DE CURSOR	
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
:	vai para a 1ª linha do arquivo
:$	vai para a última linha do arquivo

INSERINDO CARACTER E LINHAS	
a	insere caracter à direita do cursor
A	insere caracter à direita do cursor e sinaliza fim de inha
I	insere caracter à esquerda do cursor
I	insere caracter à esquerda do cursor e sinaliza fim de inha
o	insere linha abaixo do cursor
O	insere 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 a string onde 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

DELETANDO TEXTO	
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

COPIANDO OU MOVENDO TEXTO

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

SETANDO LINHA NUMERADA

:set nu	mostra as linhas numeradas
:set nonu	inibe a numeração das linhas

PROCURANDO UMA LINHA

G	vai para a última linha do arquivo
21G	vai para a linha 21
?	busca de traz para diante

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

:%s/string_velha/string_nova/g
troca string_velha pela string_nova

:g/....$/s///	deleta as 4 últimas posições
:g/^$/d	deleta linha em branco

:g/^Capit [0-9]$/g .w >> stdout
grava no arquivo todas as linhas que contenham a string "Capit" seguida de espaço em branco e 1 algarismo variando de 1 a 9

:g/^Capit [0-9]$/ . copy $
a mesma pesquisa, porem serão copiadas para o final do arquivo

:g/string/d	procura a string e deleta
:%s/string velha/string nova/g	
:%s/^h.t$/host	substitui palavra que começa com h e termina com t

LIMPANDO A TELA

Ctrl L	limpa a tela

INSERINDO UM ARQUIVO NUM ARQUIVO

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

SALVANDO E CANCELANDO

:w ou :x	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

arq .exrc
ab vc - Vocabulary

Í N D I C E