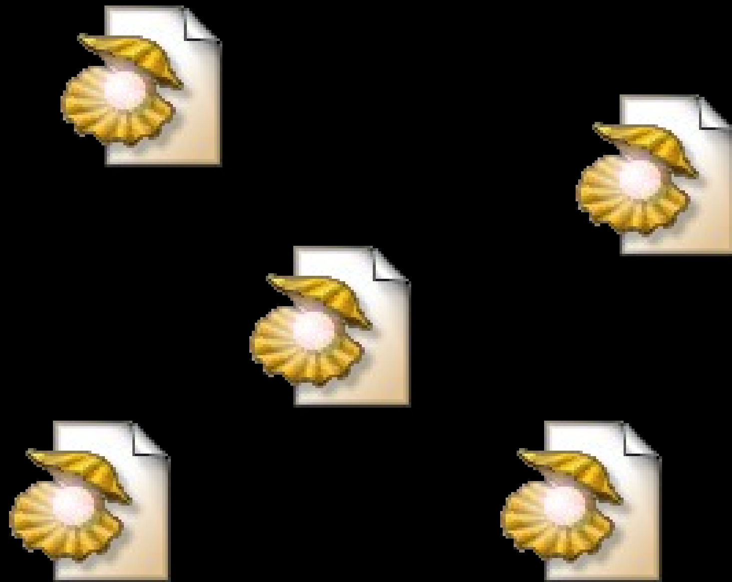
Programação em Shell Script



Jacson RC Silva <jacsonrcsilva@gmail.com>

Conteúdo

- O que é shell
- Comandos mais utilizados em shell
- Condicionais e laços
- Scripts
- Informações do Sistema
- Obtendo informações com os scripts
- Curiosidades com os scripts
- Diálogos

Primeiras Perguntas:

- O que é shell ?

- Onde se encontra o shell ?

O usuário está aqui!

Shell

Comandos e Programas

Kernel

Hardware

Aqui também!

Aqui também!

Apresentação ao Shell

- Acesso ao terminal
- Digitar uns comandos

  > ls

  > echo Seu Nome
- Terminais do linux modo texto:
  - CTRL+ALT+F[1-6]
- Terminais do linux modo gráfico:
  - CTRL+ALT+F[7-..]

Quais os processos que fornecem os terminais?

- Executem o comando:

  > *ps a*

- Verifiquem o getty

```
5585 tty1     Ss+    0:00 /sbin/getty 38400 tty1
...
5590 tty6     Ss+    0:00 /sbin/getty 38400 tty6
```

- Verifiquem o conteúdo do */dev*,

  os arquivos que começam com tty

Atribuições de variáveis

- A linguagem possui tipagem dinâmica
- Os nomes das variáveis:
  - devem ser precedidos por um sinal de igual:

    ```
    > nome=jeiks
    ```
  - podem ser formados por letras, números e sublinhado:

    ```
    > numero_pos_0_0=10
    ```
  - não podem iniciar com números:

    ```
    > 4variavel=3
    ```
  - não podem ter espaço em branco

    ```
    > nome final=jeiks
    ```

Manipulação simples de variáveis

- Utilização simples:

```
> echo $variavel $nome $etc
```

- Utilização protegida:

```
> echo ${variavel}
```

  - Quando é necessário:

```
> echo $nome_
> echo ${nome}_
```

Outras expansões de variáveis

- ${var:-texto}
  - Se var não está definida, retorna 'texto'
- ${var:=texto}
  - Se var não está definida, defina-a com 'texto'
- ${var:?texto}
  - Se var não está definida, retorna o erro 'texto'. *echo $?*
- ${var:+texto}
  - Se var está definida, executa 'texto', senão retorna o vazio

Expansões de String

- ${#var}
  - retorna o tamanho da string
- ${!texto*}
  - retorna os nomes de variáveis começadas por 'texto'
- ${var:N}
  - Retorna o texto à partir da posição 'N'
- ${var:N:tam}
  - Retorna 'tam' caracteres à partir da posição 'N'

Expansões de String

- ${var#texto}
  - corta 'texto' do início da string
- ${var##texto}
  - também corta 'texto' do início(*guloso)
- ${var%texto}
  - corta 'texto' do final da string
- ${var%%texto}
  - também 'texto' do final (*guloso)

Expansões de String

- ${var/texto/novo}
  - substitui 'texto' por 'novo', uma vez
- ${var//texto/novo}
  - substitui 'texto' por 'novo', sempre
- ${var/#texto/novo}
  - se a string começar com 'texto', substitui 'texto' por 'novo'
- ${var/%texto/novo}
  - se a string terminar com 'texto', substitui 'texto' por 'novo'

Trabalhando com bash 4.0

- bash --version
  - NOME=jacson
  - echo ${nome^}
  - echo ${nome^^}
  - NOME=${NOME^^}
  - echo ${nome,}
  - echo ${nome,,}

Linha de comandos

- Cada comando costuma receber argumentos e opções:
  - ex. comando:

    > echo
  - ex. argumentos:

    > echo “frase”
  - ex. opções:

    > echo -e “\tfrase”

Quer mais ajuda sobre os comandos?

- man
  - O que é? Use-o para lhe explicar:

    > man man
- info
  - O que é? Pergunte-o:

    > info info

Condicionais e laços

```
if CONDIÇÃO
then
  COMANDOS
else
  COMANDOS
fi
```

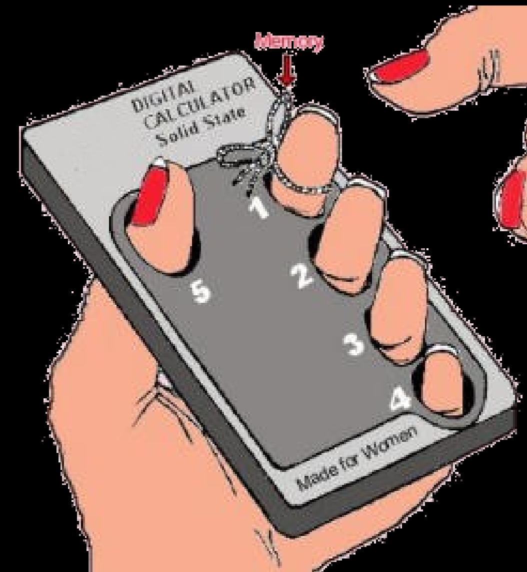
```
case $VAR in
  txt1) comandos;;
  txt2) comandos;;
esac
```

```
for var in lista
do
  comandos
done
for ((exp1;exp2;exp3))
```

```
while CONDIÇÃO
do
  comandos
done
```

Que tal uma calculadora em bash?

```
clear ; while true;do
    echo -e "\tCalculadora TabaShell”
    echo -e "Pressione CTRL+C para terminar\n"
    read -p "Digite num1: " num1
    read -p "Digite operador: " op
    read -p "Digite num2: " num2
    echo $(($num1 $op $num2))
done
```



Obs: para que serve o ponto-vírgula em shell
\t \n → man ascii

Comandos básicos

> cd → modificar pastas

> rm → apagar arquivos

> ls → listar os arquivos

> pwd → mostrar o diretório atual

> whoami → saber quem você é

> mkdir → criar diretórios

> rmdir → apagar diretórios vazios

> read → leitura de dados

> echo → escrita na tela

cd

- Para onde ir?
  - cd → vai para o diretório do usuário
  - cd .. → sobe um nível
  - cd . → entra no diretório atual
    - PWD=/home/jeiks
    - pwd
    - cd .
  - cd / → vai para a raiz do sistema
  - cd /*pasta* → vai para *pasta*

rm

- Como remover?
  - rm
    - remove o arquivo
  - rm -f
    - força remover e não produz erro
  - rm -r
    - remove de forma recursiva
  - rm -i
    - pergunta antes de remover (evita muita @#$!%)
  - rm -v
    - remove verbosamente (te mostrando a @#$!% que você fez)

mkdir

- Como criar diretórios?
  - mkdir pasta
    - cria a *pasta*
  - mkdir -p pasta/a/b/c
    - o -p permite criar subníveis
    - também não retorna erro se já existir
  - mkdir -v pasta
    - cria de forma verbosa

rmdir

- Como apagar com rmdir?
  - rmdir pasta
    - remove a pasta se estiver vazia
  - rmdir -p a/b/c/d
    - remove toda a estrutura se estiver vazia

read

- Como ler?
  - read VAR → *lê e coloca o conteúdo em $VAR*
  - read -d 'C' VAR
    - novo delimitador 'C', ao invés do '\n'
  - read -n N VAR
    - retorna após ler N caracteres
  - read -p “Frase: ” VAR
    - exibe a frase antes de ler
  - read -s VAR
    - não exibe na tela os caracteres digitados
  - read -t N VAR
    - Retorna erro se não digitar o conteúdo em N segundos

echo

- Como escrever na tela?
  - echo -n →não imprime '\n'
  - echo -e →habilita caracteres especiais
  - echo -e '\033[33;1m amarelo \033[m' #\e ou \033

prompt$ funcoeszz cores

	41;30	42;30	43;30	44;30	45;30	46;30	47;30
40;30;1	41;30;1	42;30;1	43;30;1	44;30;1	45;30;1	46;30;1	47;30;1
40;31		42;31	43;31	44;31	45;31	46;31	47;31
40;31;1	41;31;1	42;31;1	43;31;1	44;31;1	45;31;1	46;31;1	47;31;1
40;32	41;32		43;32	44;32	45;32	46;32	47;32
40;32;1	41;32;1	42;32;1	43;32;1	44;32;1	45;32;1	46;32;1	47;32;1
40;33	41;33	42;33		44;33	45;33	46;33	47;33
40;33;1	41;33;1	42;33;1	43;33;1	44;33;1	45;33;1	46;33;1	47;33;1
40;34	41;34	42;34	43;34		45;34	46;34	47;34
40;34;1	41;34;1	42;34;1	43;34;1	44;34;1	45;34;1	46;34;1	47;34;1
40;35	41;35	42;35	43;35	44;35		46;35	47;35
40;35;1	41;35;1	42;35;1	43;35;1	44;35;1	45;35;1	46;35;1	47;35;1
40;36	41;36	42;36	43;36	44;36	45;36		47;36
40;36;1	41;36;1	42;36;1	43;36;1	44;36;1	45;36;1	46;36;1	47;36;1
40;37	41;37	42;37	43;37	44;37	45;37	46;37	
40;37;1	41;37;1	42;37;1	43;37;1	44;37;1	45;37;1	46;37;1	47;37;1

Praticando...

- Leia uma frase de um usuário
- Escreva somente a primeira palavra
- Escreva a frase sem a última palavra
- Escreva a frase em maiúsculo
- Crie um variável com a frase:
  - “Digite a senha em 3s, senao eu vou explodir”
- Leia a senha do usuário, exibindo essa frase

Continuando os comandos básicos...

test []

> man test

- teste de arquivos (-e,-d,-r,-f,...)
- comparação numérica (-eq,-ne,-gt,...)
- comparação de string (=,!=,...)
- operadores lógicos (!,-a,-o)
- Testes:

```
> if [ -e arquivo ];then
>   echo "arquivo existe"
> fi
```

praticando...

- Teste se o arquivo /etc/passwd pode ser lido e se pode ser escrito
- Teste se o diretório /etc pode ser lido e se o arquivo /etc/shadow pode ser lido
- Leia uma variável e verifique se o usuário digitou algo
- Leia um número e verifique se ele é maior que 20
- Cria uma variável sexo com M e idade com 20, teste se é homem e maior de idade

E que tal fazer pesquisa no conteúdo dos arquivos?

grep

- É um filtro de strings no arquivo:

  > grep palavra arquivo

  – Pesquisa pela palavra no arquivo

  > grep -i palavra arquivo

  – Pesquisa pela palavra em case insensitive

  > grep -r palavra *

  – Pesquisa pela palavra em todos os arquivos do diretório

  > grep -n palavra *

  – Pesquisa pela palavra e demonstra a linha e arquivo em que foi encontrada

  > grep -q palavra *

  – Pesquisa sem exibir nada na tela

grep

- Mais opções:

  > grep --color palavra arquivo
  - Exibe o resultado colorido

  > grep ^palavra arquivo
  - Filtra linhas que iniciam com palavra

  > grep palavra$ arquivo
  - Filtra linhas que terminam com palavra

  > grep '[:lower:]' arquivo   # ou:

  '[:upper:]' '[:space:]' '[:alphanum:]' '[:alpha:]' '[:blank:]' '[:digit:]'

  > grep j..iks arquivo
  - Palavras com *j*, tem 2 caracteres e *iks*

praticando...

- Pesquise pelo nome do usuário logado (whoami) no arquivo /etc/passwd
- Exiba o conteúdo do menu.lst na tela, excluindo as linhas que iniciam com '#'
- Agora exiba o conteúdo anterior sem mostrar as linhas em branco
- Pesquise por palavras linux-image terminando com 686 no menu.lst

find

- Sintaxe

  find <caminho> <expressão> <ação>

- Caminho:
  - caminho ou diretório a partir do qual irá procurar pelos arquivos
- Expressão:
  - define quais os critérios de pesquisa
- Ação:
  - ação a executar com os arquivos que atenderem ao critério de pesquisa

find

- Principais critérios de pesquisa:
  - name: por nomes
  - user || group: por usuário || por grupo
  - size [+|-]<tam>[]: procura por tamanho
  - type: por tipo especificado pela letra:
    - d: diretório
    - p: named pipe
    - f: arquivo comum
    - l: link simbólico
- Mais de um critério de pesquisa:
  - -a → condicional E
  - -o → condicional OU

find

- Principais ações definidas
  - -print
- escreve os nomes dos arquivos
  - -exec <cmd> {} \;
    - executa o comando cmd
    - {} é substituido pelo nome do arquivo
    - \; encerra o comando
  - -ok <cmd> {} \;
    - o mesmo que o anterior, porém pergunta se pode executar o comando

find

- -printf <formato>
  - Permite formatar a saída e escolher campos a escrever
  - Campos:
    - %g: grupo do arquivo
    - %f : nome do arquivo
    - %h: caminho do arquivo
    - %p: caminho e nome
    - %m: permissão do arquivo
    - %s: tamanho do arquivo
    - %a: data do último acesso
    - %c: data de criação
    - %t: data de alteração
- Mais detalhes de formatação: man find

Praticando...

- criar uma saída similar ao ls -l
- faça uma pesquisa por arquivos que começam por “i” no diretório /etc
- faça uma pesquisa por arquivos que terminam por “.rc”
- encontre os arquivos que possuem alguma letra maiúscula no nome, dentro do diretório /etc e imprima o nome do grupo e o nome do arquivo na saída

xargs

- xargs
  - Recebe linhas de strings e passa como parâmetros para outros programas
- echo jeiks | xargs echo -e '\t'
- ls -l | \

  xargs -I{} bash -c \
  'for i in {};do echo $i;done'
- echo jeiks em seu curso | \

  xargs -n 1 echo -e '\t'
- echo jeiks em seu curso | \
  - xargs -n 2 echo -e '\t'

xargs

- Mais opções:
  - -nNum

    Manda o máximo de parâmetros recebidos, até o máximo de Num para o comando a ser executado
  - Lnum

    Manda o máximo de linhas recebidas, até o máximo de Num para o comando a ser executado
  - -p

    Mostra a linha de comando montada e pergunta se deseja executar
  - -t

    Mostra a linha de comando montada antes de executar

Mais comandos para auxiliar

- cat
  - -vet : mostra caracteres invisíveis
  - -n : mostra o número da linha
- wc
  - -l : conta linhas
  - -w : conta palavras
  - -c : conta caracteres
- sort
  - -m : intercala dois arquivos
  - -n : classificação numérica
  - -r : inverte a ordem de classificação

Trabalhando com redirecionamentos de entradas e saídas...

Redirecionamento

- | → envia a saída de um comando para a entrada do outro
- tee → envia a saída para um arquivo e para a tela e um arquivo
- > → envia a saída do comando para um arquivo:

```
echo Jeiks > arquivo
```

- < → envia para a entrada do comando o conteúdo do arquivo:

```
./a.out < arquivo
```

Redirecionamento de saídas

- Redirecionar a saída padrão:
  - comando > /dev/null
- Redirecionar a saída de erros:
  - comando-falho 2> /dev/null
- Redirecionar a saída de erros na entrada padrão:
  - comando 2>&1
  - Ex: comando-falhou 2>&1 2> /dev/null

Redirecionando mensagens

- Escrevendo na saída de erro:
  - echo oi >&2
- Confirmando:
  - echo oi >&2 | grep -v oi
- Tem outra forma!?
  - echo oi > /dev/stderr | grep -v oi

Quê isso fessô maluco??

- /dev
  - /stdout → saída padrão
  - /stdin → entrada padrão
  - /stderr → saída de erros

Redirecionamento de entrada

- << → indica que o escopo só acabará ao terminar o rótulo definido, ex:
  - cat << rotulo-teste
  - Teste de texto
  - rotulo-teste
- <<< → direciona o conteúdo para o comando como se o mesmo fosse escrito no terminal, ex:
  - read nome <<< jeiks
  - echo $nome

Mas como é isso??

Exemplo em C...

praticando...

- Em uma mesma linha de comando, faça:
  - Liste os arquivos retirando os que tem letra maiúscula e salvando o resultado em um arquivo
  - Como usuário normal, pesquise todos os arquivos do /etc
    - Direcione a lista para um arquivo
    - Direcione os erros para outro arquivo
    - Percorra o arquivo de lista e exiba o nome de todos os arquivos em maiúsculo

Mais manipulação

- head
  - -l N | -n N | --lines N
    - mostra as N primeiras linhas do arquivo
  - -c N
    - mostra os primeiros N caracteres
- tail
  - -l N | -n N | --lines N
    - mostra as N últimas linhas do arquivo
  - -c N
    - mostra os últimos N caracteres
  - -f ou -F
    - exibe na tela à medida que o arquivo cresce

Mais manipulação

- cut
  - seleciona campos de cada linha
    - -d : delimitador de campos
    - -f : lista de campos a cortar
- uniq
  - omite linhas idênticas

praticando...

- Coloque diversos nomes em um arquivo:
  - Com nomes repetidos;
  - Iniciando com letras maiúsculas;
  - Iniciando com letras minúsculas.
- Selecione as primeiras 5 linhas e ordene-as
- Retire as linhas repetidas
- Selecione as linhas que iniciam com letras minúsculas,

  transforme-as em maiúsculas e

  as ordene sem repetir nenhuma palavra igual

Mais comandos utilizados no shell...

Date e cal

- date
  - Exibe a data e hora atual
  - Formatos de data: man date
- cal
  - Exibe o calendário
  - cal <mes>
  - cal <ano>
  - cal <mes> <ano>
  - Curiosidade: cal 09 1752

Trocando as letras ;)

- “criptografia”...

```
$ echo "Jeiks" > arquivo
$ tr 'aeiou' 'mnpyt' < arquivo \
  > arquivo-criptografado
$ cat arquivo-criptografado
Jnpks
```

- Maiúsculas para minúsculas:

```
$ echo JEIKS | tr [A-Z] [a-z]
```

Encontrando arquivos

- Para encontrar arquivos utilize o comando find:

  > find local -name "*arquivo*"

  > find local -iname "*arquivo*"

- Outros parâmetros:
  - -type, -o, -a, -exec
  - Procure mais com: man find

Controle de sistema e processos

- Processos em execução:

  > ps

  - parâmetros:
  - f a u x
- Exercício:
  - Filtre os processos executados pelo seu usuário
  - Vamos encontrar o gerenciamento de um desses processos?

Informações Obtidas do Sistema

/proc

- O /proc é um pseudo sistema de arquivos de informações de processos.

```
> cd /proc
> gedit &> /dev/null &
> ps faux | grep `whoami` \
      grep -v grep | \
      grep gedit
> ls PID
```

Subshells

- Mas o que foi o `whoami` que apareceu no exemplo anterior?
- É a forma de executar comandos em uma subshell
- Outra forma: $(whoami)
- Utilização:

```
usuario=`whoami`
# ou
usuario=$(whoami)
```

Mais informações obtidas no /proc

- Qual a CPU do sistema?

```
> cat /proc/cpuinfo
```

- Informações da memória:

```
> cat /proc/meminfo
```

- Qual driver de framebuffer:

```
> cat /proc/fb
```

- Quais as partições do sistema:

```
> cat /proc/diskstats
```

Mais informações obtidas no /proc

- Módulos do sistema:

```
> cat /proc/modules
```

- Parâmetros de inicialização:

```
> cat /proc/cmdline
```

- Informações de rede:

```
> cat /proc/net/dev
```

- Procure mais com:

```
> man proc
```

Automatizando processos

- Todos esses comandos podem ser utilizados para formar "textos" que serão interpretados pelo shell.
- Estes "textos" são os *scripts*

- Vamos fazer nosso primeiro script?

Primeiro Script

- Iniciando um script, faça:
  - Crie o arquivo teste.sh
  - Adicione as linhas:

```
#!/bin/bash
# meu primeiro comentário
echo "Meu primeiro arquivo"
```

  - Salve, vá no terminal, na pasta que salvou, e dê permissão de execução ao seu script:
  - chmod +x teste.sh
  - Agora execute: ./teste.sh

Ops... permissão? Como assim?

- Execute o comando: `ls -l`

  Nesta lista de arquivos, a primeira coluna indica as permissões:

  - d → diretório
  - - → arquivo normal
  - l → link
  - p → pipe

u g o

754

	u			g			o		
access	r	w	x	r	w	x	r	w	x
binary	4	2	1	4	2	1	4	2	1
enabled	1	1	1	1	0	1	1	0	0
result	4	2	1	4	0	1	4	0	0
total		7			5			4	

E funções, existem?

- As funções nos scripts podem ser criadas com a seguinte sintaxe:

```
function teste {
  echo teste
}
```

ou

```
teste() {
  echo teste
}
```

E como eu recebo os parâmetros?

Através dos parâmetros posicionais:

- $0 → Parâmetro número 0 (nome do comando ou função)
- $1 → Parâmetro número 1 (da linha de comando ou função)
- ${2} → Parâmetro número 2 (da linha de comando ou função)
- $# → Número total de parâmetros da linha de comando ou função
- $* → Todos os parâmetros, como uma string única
- $@ → Todos os parâmetros, como várias strings protegidas

usuários e homes

- Obter o nome dos arquivos e os diretórios *home* de cada um através do */etc/passwd*
- Dicas:
  - cut → comando para obter sessões das linhas dos arquivos:

    ```
    echo "nome jeiks
    nome jacson" | cut -d ' ' -f2
    ```

  - getent → obtém as entradas dos bancos de dados administrativos do sistema:

    ```
    getent passwd
    ```

Informações do CPU

Informações do CPU:

```
function cpu_info {

   modelo=$(cat /proc/cpuinfo    |  \
           grep -i "model name" | cut -d: -f2)

   freq=$(cat /proc/cpuinfo   |  \
        grep -i "cpu MHz"     | cut -d: -f2)

   cache=$(cat /proc/cpuinfo  |  \
         grep -i "cache size" | cut -d: -f2)

 echo -e "Você tem um computador $modelo
 com $freq Mhz e um cache de $cache"

}
```

Informações de Rede

Interfaces de redes:

```
function interfaces {
   inter=$(cat /proc/net/dev | grep ':' | cut -d: -f1)
   echo "Suas interfaces de rede são:"
   for i in $inter;do
       echo "[$i]"
   done
}
```

Utilizando os scripts para obter informações de comandos e arquivos shell...

Configurações básicas de rede

- Obtendo ip:

  `/sbin/ifconfig`

- Obtendo rota:

  `/sbin/route`

- Obtendo DNS:

  `cat /etc/resolv.conf`

- Porém ele mostra muita coisa, e queremos somente o básico, então temos que filtrar...

Filtrando

- Primeiro vamos obter a rota padrão, para então sabermos qual interface obter o IP

```
/sbin/route -n
```

- Porém existem várias rotas, como fazer? Basta obter somente a rota padrão:

```
/sbin/route -n | grep '^0\.0\.0\.0'
```

- ok... este comando fornece a rota padrão e o nome da interface, mas fornece outras informações... como obter somente o que queremos?

Conceitos básicos do awk

- Que tal utilizar o awk?

```
/sbin/route -n | \
  awk '{if ($1 == "0.0.0.0")
          print $2" "$8
        }'
```

- Agora temos o que queremos, mas como utilizar?
- Podemos representar melhor a saída do awk e utilizar o eval...

Awk e eval

- Primeiro nos preocupamos com a saída do eval, queremos a escrita como se fosse definir as variáveis:

```
/sbin/route -n | \
    awk '{if ($1 == "0.0.0.0")
           print "ROTA="$2" INTER="$8
          }'
```

- E agora, para que ele avalie a expressão resultante e defina as variáveis:

```
eval $( comando acima )
```

Continuando a análise

- Agora que possuímos uma variável com a interface, podemos obter o IP da interface correta:

```
LOCALE=en /sbin/ifconfig $INTER | \
          grep inet\ end.:
```

- Ainda temos muita coisa... que tal aproveitarmos e já definirmos o BroadCast e a Máscara da rede (dessa vez com *sed*):

```
comando_anterior | \
    sed 's/.*inet end.: /IP=/' | \
    sed 's/Bcast:/BCAST=/'     | \
    sed 's/Masc:/MASC=/'
```

Melhorando o sed e definindo as variáveis

- ```
  LOCALE=en /sbin/ifconfig $ip | \
    grep inet\ end.:             | \
    sed 's/.*inet end.: /IP=/   ;
    s/Bcast:/BCAST=/            ;
    s/Masc:/MASC=/'
  ```
- Definindo as variáveis:

  ```
  eval $(comando anterior)
  ```
- Agora, falta só o DNS...
  ```

Obtendo o DNS

- ```
  DNS=$(grep -e '^nameserver'   \
          /etc/resolv.conf       | \
          head -n 1 | cut -d' ' -f2)
  ```
- ou:
- ```
  DNS=$( awk '{
              if ($1 == "nameserver")
                 { print $2; exit; }
                 }' /etc/resolv.conf )
  ```

Obtendo os resultados...

```
cat << EOF
INTERFACE PRINCIPAL: $INTER
IP: $IP
BroadCast: $BCAST
Máscara: $MASC
DNS: $DNS
ROTA: $ROTA
EOF
```

Resultados para um arquivo

```
cat << EOF > arq-resultados
INTERFACE PRINCIPAL: $INTER
IP: $IP
BroadCast: $BCAST
Máscara: $MASC
DNS: $DNS
ROTA: $ROTA
EOF
```

ok... e onde o sistema utiliza esses scripts?

- Processo de inicialização
- GRUB → menu.lst
- Kernel e primeiro sistema de arquivos
- Chamada do init
  - */sbin/init* → pai de todos os processos
    - ps fax | grep init
    - Tente matar: kill -9 1
    - kill é novidade?
      vamos aprender como funciona...
  - */etc/inittab* → arquivo de configuração
  - */etc/init.d* → pasta com todos os scripts de inicialização

init

/etc/rcS.d/

/etc/rc[1-6].d/

- Estas pastas possuem *links* com os arquivos de */etc/init.d*
- *Link*? O que é isso?

Vamos aprender:

```
$ echo informacao > arquivo
$ ln arquivo arquivo2
$ ln -s arquivo arquivo3
```

Voltando ao init

- Os scripts *linkados* em rcS.d são sempre executados
- Os demais scripts dependem da linha referente no /etc/inittab. Exemplo:

  id:2:initdefault:

- Então, todos os *links* de /etc/rc2.d serão executados
- A nomenclatura define o **S***tart* ou **K***ill* e a ordem de execução dos scripts. Ex: S13kdm (13° a ser executado)

Outras coisinhas interessantes...

- Trap
  - Sinais Mais Importantes:

    0 → EXIT → Fim normal do programa

    1 → SIGHUP → Quando recebe um *kill -HUP*

    2 → SIGINT → Interrupção pelo teclado (<CTRL+C>)

    3 → SIGQUIT → Interrupção pelo teclado (<CTRL+\>)

    15 → SIGTERM → Quando recebe um *kill* ou *kill -TERM*
  - Utilizando: `trap "echo 'pressionou CTRL+C'" 2`
  - Desfazendo: `trap 2`
  - Teste interessante:

```
trap "echo Mudou o tamanho da janela" 28
```

Outras coisinhas interessantes...

- Comunicação entre processos:

```
> mkfifo cano

  # vai travar, pois ninguém ouve:
> echo oi > cano    # vai falar
> cat cano          # e vai ouvir

  # vai travar, pois ninguém fala:
> cat cano          # vai ouvir
> echo oi > cano    # e vai falar
```

Outras coisinhas interessantes...

- Comandos in line:

```
> test -f arquivo      && \
  echo arquivo existe || \
  echo arquivo não existe
> [ $num = 1 ] && {
      num=3
      echo \$num valia $num, agora vale 3
  }
```

E como se comunicar com os usuários no modo gráfico?

- Diálogos
  - Kdialog
  - Gdialog
  - Zenity

```
> kdialog --help
> zenity  --help
```

Frontend - GDialog

```
res=$( gdialog --menu "Título do menu" \
     0 0 0 \
     1 "Opção 1" \
     2 "Opção 2")

if [ $res -eq 1 ]; then
    gdialog --msgbox “Você escolheu 1”
else
    gdialog --msgbox “Você escolheu 2”
fi
```

Ajuda em: www.unixref.com/manPages/gdialog.html

Referências Interessantes

- Página do Júlio Neves:
  <http://jneves.wordpress.com>
- Página do Aurélio:
  <http://aurelio.net>
- Página do Thobias:
  <http://thobias.org>
- Advanced Bash-Scripting Guide:
  <http://tldp.org/LDP/abs>

Agora é maré mansa....