O Shell

O Que é o Shell?

Um interpretador de comandos que:

- Atribui valores as variáveis do ambiente
- Executa comandos, seja em background ou foreground

Quando um comando é digitado, o shell:

- Executa a substituição de variáveis
- Expande metacaracteres
- Trata os redirecionamentos de E/S e pipelines
- Executa a substituição de comandos
- Executa o comando

O Sistema Operacional UNIX possui o Bourne shell (sh), o Korn shell (ksh), o C-shell (csh) e outros. Há algumas diferenças importantes entre eles, por exemplo, quando é utilizado o shell Born ou Korn, o símbolo default do prompt é $. O símbolo default do C shell é o %. Veremos algumas direfenças neste capítulo.

Área local de dados

No Born shell e Korn shell

Por default, quando uma variável shell é setada, a memória é alocada na área local de dados. As variáveis desta área são privativas do shell corrente. Sendo assim, processos subsequentes não podem acessar estas variáveis. Contudo, existe uma maneira de mover estas variáveis para o ambiente, utilizando o comando export.

No C-shell

No C-shell as variáveis de um script ficam na Área local de dados, um espaço reservado na memória durante a execução de um script. Elas podem ser atríbudas ao shell que o executou, isto é, ao ambiente, com um comando intrínseco do shell, o source.

Variáveis shell

- Um nome que define um valor
- São inicializadas com NULL (nulo)
- Algumas variáveis podem ser setadas pelo usuário

Variáveis shell são similares a uma variável algébrica, um nome com um valor associado.

Existem variáveis shell especiais que são setadas para o usuário. Estas variáveis podem ser alteradas para customizar o ambiente do usuário. Relembrando algumas delas:

HOME, LOGNAME, PATH, PS1, PS2, TERM...

Outras variáveis deste tipo podem ser vistas na seção 1 do UNIX Reference Manual, para o shell em particular: sh, csh ou ksh.

Setando variáveis shell

No ksh e sh

Sintaxe:

nome=valor

Exemplos:

meu_nome_e=usuario1
outrodir=/producao/dirprg/prgsfol
MSG1="Mensagem1 do usuario"

Variáveis são um lugar para armazenar informação, e estas informações são acessadas através da referência ao nome. Especificamente, variáveis shell vão conter informações que o usuário vai utilizar em uma linha de comando shell.
Variáveis shell podem ser criadas após o prompt do shell. Ao criar variáveis não utilize espaço em branco antes ou depois do sinal de = . Ao utilizar espaços em branco, o shell vai interpretar como a ativação de um comando.
A criação de variáveis shell devem obedecer as seguintes regras:

- O nome da variável deve ser iniciado com letra
- O nome deve conter preferencialmente letras, digitos e underscores (_)

É importante saber fazer a distinção entre o nome da variável shell e o valor da variável shell. No terceiro exemplo acima, o nome da variável shell é MSG1, e o valor da variável shell é Mensagem1 do usuario.

No csh

Utiliza-se dos comandos set e setenv que serão vistos logo a seguir. Imediantamente a variável estará disponível no shell que a executou. Durante a execução de um script também é possível disponibilizá-la no shell corrente com o comando source.

Referenciando variáveis shell

Sintaxe:

$nome

Exemplos:

```
% echo $meu_nome
usuario1
%
% echo $outrodir
/producao/dirprg/prgsfol
% ls $outrodir
```

```
< conteúdo do diretório /producao/dirprg/prgsfol >
%
% echo $MSG1 Mensagem1 do usuario
%
```

e usuários possuem variáveis shell, utilizam o valor associado com o nome da variável. O shell pesquisa o nome e retorna o valor associado a ela. Este procedimento é chamado de substituição de variável. O shell executa a substituição de variáveis em toda a linha de comando que contenha o simbolo $, associado a um nome válido de variável.
O shell executa os seguintes procedimentos para toda linha de comando:

- Pesquisa a linha de comando, procurando pelo simbolo $
- Encontrando o simbolo $, associado a um nome válido de variável, o shell remove $nomevar da linha de comando, salva este nome em memória, pesquisa o valor associado com este nome, e coloca este valor na linha de comando, no local que foi removido $nomevar.
- Após todas as substituições necessárias, o shell executa a linha de comando com o resultado de todas as substituições.

O ambiente

Cada shell possui duas áreas de dados:

- Área local de dados
- Ambiente

Podemos utilizar quatro comandos para alterar estas áreas:

- set
- unset
- env
- export (sh, ksh, kish)
- setenv (csh, tcsh)
- source (csh, tcsh)

O ambiente é outra área de memória utilizada pelo shell para armazenar variáveis shell e seus respectivos valores. Variáveis definidas no ambiente são disponibilizadas para processos filhos. Processos filhos são formados pelo shell para a execução de comandos.
A Área local de dados é aquela utilizada por programas em sua execução, por exemplo, em um script shell são as variáveis declaradas dentro do programa.
Todas as variáveis do ambiente podem ser acessadas por comandos subseqüentes.

O Comando set

No Born shell e Korn shell

Informa nomes e valores de todas as variáveis shell da **área local de dados** e do **ambiente**.

Exemplo:

positron[empada]% set
argv ()
cwd /home/cbpfsu2/empada
history 200
home /home/cbpfsu2/empada
lpath (/usr/openwin/bin/xview /usr/openwin/bin /usr/local/bin /usr/local1/bin /home1/local/bin /opt/SUNWspro/bin)
mychoice openwin
noclobber
path (. /home/cbpfsu2/empada /home/cbpfsu2/empada/bin /usr/openwin/bin/xview /usr/openwin/bin /usr/local/bin /usr/local1/bin /home1/local/bin /opt/SUNWspro/bin /usr/ucb /usr/bin /usr/etc)
prompt positron[empada]%
savehist 1000
shell /bin/csh
status 0
term vt100
user empada

OBS: **PATH** É uma variável do Shell do usuário que define o diretório

No C-shell

Armazena em memória variáveis da área local de dados ou do ambiente e mostra as variáveis setadas com o comando.
Sintaxe:

set (ksh e sh)
set [= [' | "] *valor* [' | "]] (csh)

Exemplo:

mgnrio1> set meu_nome='usuario1'
mgnrio1> set
HOME=/users/usuario1
MSG1=Mensagem1 do usuario
PATH=/bin:/usr/bin:/usr/contrib/bin:/usr/local/bin
PS1=$
PS2=>
TERM=vt100
.
.
.
meu_nome=usuario1
outrodir=/producao/dirprg/prgsfol
mgnrio1>

O C-shell ainda tem a possibilidade de dois ambientes de variáveis, com um comando para cada atribuição de valores. Este comando permite atribuir variáveis minúsculas quando executado do prompt, e tanto minúsculas quanto maiúsculas quando armazenado em memória através do comando "source

arquivo" na linha de comando. O comando para criar variáveis maiúsculas no C-shell é o "setenv *variável*".

O Comando unset

Remove o valor de uma variável. Exite tanto no ksh e sh quanto no csh.
Sintaxe:

unset nomevar

Exemplo:

$ unset outrodir

O comando unset reseta (remove) o valor de uma variável shell, colocando NULL.

Comando env

Informa nomes e valores de todas as variáveis shell do ambiente. Disponível em todos os shells.
Sintaxe:

env

Exemplo:

positron[empada]% env
USER=empada
LOGNAME=empada
HOME=/home/cbpfsu2/empada
PATH=.:/home/cbpfsu2/empada:/home/cbpfsu2/empada/bin:/usr/openwin/bin/xview:/usr/openwin/bin:/usr/local/bin:/usr/local1/bin:/home1/local/bin:/opt/SUNWspro/bin:/usr/ucb:/usr/bin:/usr/etc
MAIL=/var/mail//empada
SHELL=/bin/csh
TZ=Brazil/East
SSH_CLIENT=152.84.253.61 4381 22
SSH_TTY=/dev/pts/5
TERM=vt100
PWD=/home/cbpfsu2/empada
MANPATH=/usr/openwin/man:/usr/man:/opt/SUNWspro/man:/usr/local/man
LD_LIBRARY_PATH=/usr/local/lib:/usr/openwin/lib:/opt/SUNWspro/SC4.0:/usr/local/X11R6/lib
OPENWINHOME=/usr/openwin

O Comando export

Movimenta variáveis da área local de dados para o ambiente. Só disponível nos shells ksh, e sh.
Sintaxe:

export [nome_variável]

Exemplos:

$ export outrodir
$ export MSG1

```
$ export
export MSG1
export outrodir
$
```

Quando utilizado sem argumentos, o comando export informa as variáveis assinaladas como exportáveis.

Comandos intrínsecos do C-shell

O Comando setenv

Permite visualizar ou atribuir valores a variáveis de área local de dados ou ambiente. Normalmente maiúsculas. Múltiplos valores são separados por ":".
Sintaxe:

```
setenv [ nome_variável valor1:valor2:...]
```

Exemplos:

positron[empada]% setenv
_=/usr/bin/csh
MANPATH=/usr/openwin/man:/usr/man:/opt/SUNWspro/man:/usr/local/man
SSH_TTY=/dev/pts/5
PATH=.:/home/cbpfsu2/empada:/home/cbpfsu2/empada/bin:/usr/openwin/bin/xview:/usr/openwin/bin:/usr/local/bin:/usr/local1/bin:/home1/local/bin:/opt/SUNWspro/bin:/usr/ucb:/usr/bin:/usr/etc
OPENWINHOME=/usr/openwin
LOGNAME=empada
MAIL=/var/mail//empada
USER=empada
SHELL=/bin/csh
HOME=/home/cbpfsu2/empada
SSH_CLIENT=152.84.253.61 4381 22
LD_LIBRARY_PATH=/usr/local/lib:/usr/openwin/lib:/opt/SUNWspro/SC4.0:/usr/local/X11R6/lib
TERM=vt100
PWD=/home/cbpfsu2/empada
TZ=Brazil/EastSem argumentos o comando se comporta equivalentemente ao "env".

O Comando Source

Salva no ambiente os dados que seriam apenas da área local de dados de um script.
Sintaxe:

```
source arquivo
```

Quando executado direto no prompt, um script tem suas variáveis locais só durante sua execução. Com este comando o shell executor (aquele que executou o "source") passa a conter as variáveis no ambiente.
Exemplo:

```
% cat script1
#!/bin/csh -f
```

```
set term=vt100
set path=($path /usr/etc)
set teste=true
echo testando...
% source script1
testando...
% set
argv ()
cwd /users/luiz/public_html
echo_style bsd
edit
gid 15
history 40
home /users/luiz
ignoreeof
mail /usr/spool/mail/luiz
path (/usr/local/bin /usr/bin /bin /usr/ucb /usr/etc)
term vt100
teste true
%
```

Execução de Comandos

Objetivos deste capítulo:

- Descrever diferença entre comandos intrínsecos do shell e comandos do UNIX
- Explicar a pesquisa do shell para localizar um comando
- Utilização dos comandos whereis e find para localizar arquivos no UNIX
- Mostrar alguns comandos para a identificação de usuários e manipulação de identificadores (id, su e newgrp)
- Explicar como um processo é formado (fork e exec)
- Utilização do comando ps para monitorar a execução de processos
- Explicar como o shell define quem vai executar o comando

Comandos Intrínsecos do Shell X Comandos do Unix

- Intrínsecos do shell
  - cd
  - pwd
  - echo
- Comandos do UNIX
  - ls
  - lp
  - pg

Alguns comandos utilizados pelos usuários são arquivos localizados nos diretórios /bin ou /usr/bin, são os comandos do UNIX. Outros comandos como cd, pwd e echo, estão embutidos no shell. Estes comandos não existem em arquivos do Sistema Operacional UNIX, são subrotinas do programa shell. Estes comandos são chamados de intrínsecos do shell.
Quando um comando intrínseco do shell é executado, ele não causa a criação de processos filhos.

Comandos que existem em diretórios são executados em processos filhos. O shell deve saber onde localizar estes arquivos. A variável PATH do shell do usuário define os diretórios para a pesquisa do shell e a ordem desta pesquisa.

Comando whereis

Pesquisa a lista de diretórios procurando por comandos.
Sintaxe:

whereis [-b | -m | s] comando

Exemplos:

positron[empada]% whereis vi
vi: /usr/bin/vi /usr/ucb/vi

O comando whereis pesquisa os seguintes diretórios:

- /etc
- /etc/nls
- /sbin
- /usr/bin
- /usr/lbin
- /usr/lbin/spell
- /usr/ccs/lib
- /usr/lib
- /usr/local
- /usr/hosts
- /usr/sbin

Variando de sistema para sistema evidentemente.
As opções b, m ou s são utilizadas para limitar a pesquisa em arquivos binários, páginas de manual ou códigos fonte, respectivamente.

Comando find

Pesquisa o sistema de arquivos procurando arquivos que satisfaçam uma condição.
Sintaxe:

find percurso [condição]

Exemplos:

```
$ find / -print
/
/lost+found
/bin
. . .
$ find . -name arquivo1
./usuario1/arquivo1
./usuario1/dir1/arquivo1
./usuario2/arquivo1
. . .
```

O comando find é um comando que executa pesquisa automática no sistema de arquivos. Ele é muito lento e requer muita utilização de CPU, deve ter uma utilização criteriosa.
O percurso especificado é pesquisado recursivamente, procurando por arquivos que satisfaçam a condição especificada, e, opcionalmente, tarefas podem ser executadas sobre os arquivos localizados.
Este comando possui inúmeras utilizações, baseadas nas suas diversas opções. É um comando muito utilizado em funções de back-up.
Algumas opções muito úteis do comando find:

- **-name arquivo** Especifica o nome do arquivo
- **-user nome** Especifica o dono dos arquivos
- **-group nome** Especifica o grupo dos arquivos
- **-size número** Especifica o tamanho em blocos (512 bytes) dos arquivos
- **-type tipo** Especifica o tipo do arquivo (f,d,c,b...)
- **-exec cmd {}** Especifica o comando a executar nos arquivos localizados

Exemplos:

```
$ find / -user usuario3
/users/usuario3/arq
/users/usuario3/dir1/arqx
. . .
$ find / -size +2
/lost+found
/bin
/bin/cat
. . .
$
$ find / -name "*.bak" -exec rm {} \;
$ find / -name "*.old" -exec mv {} /bkp \;
$
```

Comando id

Informa a identificação do usuário (user-id e group-id).
Sintaxe:

```
id
```

Exemplo:

positron[empada]% id
uid=594(empada) gid=100(staffe)

O comando id é intrínseco do shell. Ele informa o número do identificação e nome do usuário, e o número e o nome do grupo, do usuário logado correntemente. Sendo assim, esta identificação será diferente quando utiliza-se outro nome de login.
A variável LOGNAME e o comando who am i informam a identificação inicial de login, enquanto que os comandos id e whoami informam a identificação do usuário corrente.

Comando w

w mostra o login, hora e o tempo do usuário na máquina.

Exemplo:

positron[empada]% w

```
  9:55am  up 5 day(s), 22:29,  8 users,  load average: 0.06, 0.05, 0.04
User     tty          login@  idle   JCPU   PCPU  what
mg10     pts/1        8:10am  1:26                pine
eller    pts/3        9:08am     8                ssh -l marcelo cyclone
moyano   pts/4        9:54am                      -csh
amaralmg pts/6        9:41am    14                -csh
eller    pts/2        8:29am    48     11     11  ssh -l marcelo cyclone
empada   pts/5        9:37am     2                w
santini  pts/7        9:43am           5      5  pine
aline    pts/9        9:54am                      pine
```

Comando who

Who Mostra o login dos usuários,a sessão em que eles estão, a data e o IP da máquina que eles estão usando para conexão.

Exemplo:

positron[empada]% who

```
mg10       pts/1        May 15 08:10    (152.84.100.159)
eller      pts/3        May 15 09:08    (152.84.253.61)
moyano     pts/4        May 15 09:56    (dhcp231.ifae.es)
amaralmg   pts/6        May 15 09:41    (152.84.253.247)
eller      pts/2        May 15 08:29    (152.84.253.61)
empada     pts/5        May 15 09:37    (152.84.253.61)
santini    pts/7        May 15 09:43    (200.156.7.40)
aline      pts/9        May 15 09:54    (212245.rjo.virtua.com.br)
```

Comando su

Altera a identificação do usuário (user-id).
Sintaxe:

```
su [ usuario ]
```

Exemplos:

```
$ id
uid=201 (usuario1) gid=20 (producao)
$ su usuario3
Password:
$ id
uid=203 (usuario3) gid=20 (producao)
$ pwd
/users/usuario1
$
```

```
$ id
uid=201 (usuario1) gid=20 (producao)
$ su - usuario3
Password:
$ id
uid=203 (usuario3) gid=20 (producao)
$ pwd
/users/usuario3
$
# id
uid=0 (root) gid=3 (sys)
# su - usuario1
$ id
uid=201 (usuario1) gid=20 (producao)
$ pwd
/users/usuario1
$
```

O comando su (set user) permite alterar a identificação do usuário. Quando ativado sem parâmetros, significa alterar a identificação para usuário root.

Comando newgrp

Altera a identificação do grupo (group-id).
Sintaxe:

```
newgrp [ grupo ]
```

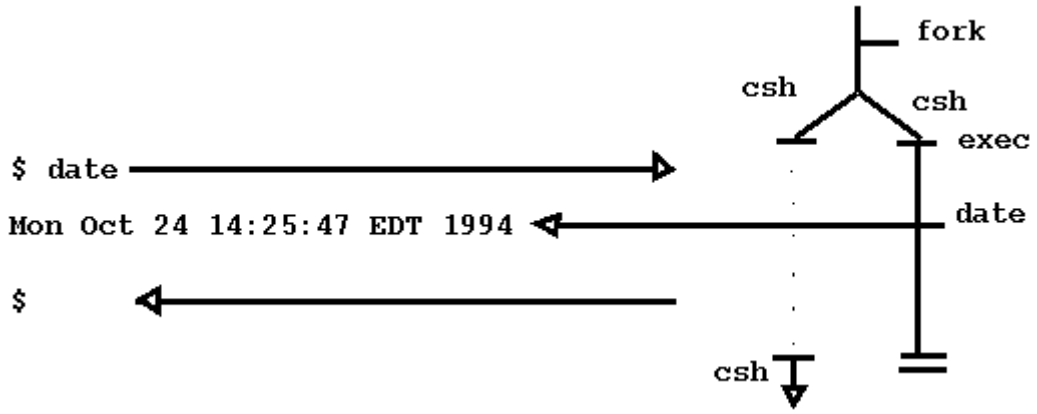
Exemplos:

```
$ id
uid=201 (usuario1) gid=20 (producao)
$ newgrp desenv
$ id
uid=201 (usuario1) gid=30 (desenv)
$ newgrp producao
ou
$ newgrp
$ id
uid=201 (usuario1) gid=20 (producao)
$ newgrp audit
Sorry
$
$ newgrp qualquer
Unknown group
$
```

O comando newgrp (new group) é similar ao comando su. Permite ao usuário alterar a identificação do grupo. A diferença em relação ao comando su é que, quando o comando newgrp é ativado sem parâmetros, retorna ao grupo de trabalho original.
Usuários só podem alterar a identificação do grupo se estiverem cadastrados em outros grupos (/etc/group) pelo Administrador do Sistema.

Como um Processo é Formado

Processos no UNIX são criados, executam e morrem. Quando um shell é executado e um usuário requisita a execução de um comando, um processo filho é criado. O processo filho executa até ser completado. O processo pai (o shell do usuário), aguarda que o processo filho seja finalizado, e continua a sua operação. A criação de processos é realizada pelo Sistema Operacional através de duas chamadas ao sistema, conhecidas como fork e exec.
Quando um processo filho é formado, é criada uma cópia do processo pai. Isto é chamado de fork. O código executável do processo pai, a área local de dados e o ambiente são copiados. Esta cópia é chamada de processo filho. Enquanto isto, o processo pai dorme e espera que o Sistema o acorde após a finalização do processo filho.
Em seguida, o código executável e a área local de dados do processo filho vão ser compartilhados com o código executavel e a área local de dados para o comando requisitado, este procedimento é chamado de exec. O ambiente do processo pai torna-se o ambiente do processo filho. Assim, todas as variáveis exportadas vão estar disponiveis ao processo filho.
O processo filho é executado, e quando finalizado é enviado um sinal ao processo pai, que aguardava a conclusão do processo filho.
Quando um processo filho morre, mas o pai ainda não comunicou que ele morreu, é chamado de processo zumbi. O processo filho cujo pai morre antes da sua conclusão, é chamado de processo órfão.



Comando ps

Apresenta informações (status) sobre os processos.
Sintaxe:
ps [-e | -f | -l]
Exemplos:

positron[empada]% ps

```
  PID TT       S  TIME COMMAND
12791 pts/5    S  0:00 -csh
12837 pts/5    S  0:00 ksh
12842 pts/5    S  0:00 -sh
```

positron[empada]% ps -e

```
  PID TT       S  TIME COMMAND
 12791 pts/5          S      0:00 -csh USER=empada LOGNAME=empada
HOME=/home/cbpfsu2/ empada PATH=/usr/bin:/bin:/usr/sbin:/sbin:/opt/b
 12837 pts/5          S      0:00 ksh USER=empada LOGNAME=empada
HOME=/home/cbpfsu2/
empada PATH=.:/home/cbpfsu2/empada:/home/cbpfsu2/
 12842 pts/5                 S           0:00      -sh      _=/usr/bin/csh
MANPATH=/usr/openwin/man:/usr/man:/opt/
SUNWspro/man:/usr/local/man SSH_TTY=/dev/
 12938 pts/5                 S           0:00      -csh      _=/usr/bin/csh
MANPATH=/usr/openwin/man:/usr/man:/opt/
SUNWspro/man:/usr/local/man SSH_TTY=/dev
```

O comando ps apresenta informações sobre os processos do sistema. Quando ativado sem opções, apresenta informações resumidas sobre os processos associados ao terminal no qual foi ativado o comando.
Algumas das opções são:

- -e Apresenta informações sobre todos os processos do sistema.
- -f Apresenta mais informações sobre os processos
- -l Apresenta a lista mais completa de informações sobre os processos

As opções podem ser utilizadas em conjunto. O comand ps é um grande consumidor de CPU; utiliza-lo para investigações sobre tempo de resposta não é uma boa idéia. Execução de Comandos.