Uso Eficiente da Linha de Comando com os Shells do Unix

Fábio Olivé
(fabio.olive@gmail.com)

Tópicos

- História da “Linha de Comando”
- Shells mais comuns
- Interpretação e transformações da linha digitada
- Sequências de comandos em linha
- Shell scripts
- Utilitários voltados para shell e manipulação de texto

AVISO

Conteúdo meio denso!
Vamos com calma. :-)

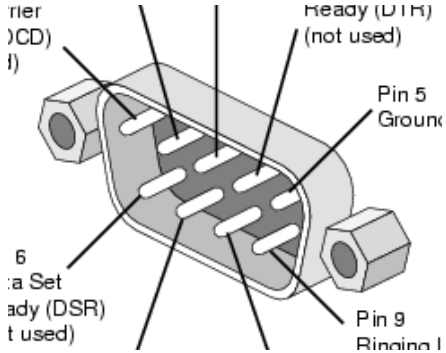
livre OeSC

História da “Linha de Comando”

- Primeiro nível de abstração da máquina: switchboard
  - Primeiras interfaces eram interruptores e luzinhas
  - Agiam diretamente sobre a memória da máquina
  - Leitores de cartão perfurado ou fita de papel perfurada
    - Entrada do código de boot da máquina
  - “Consoles” primitivos com teclado e display hexadecimal

História da “Linha de Comando”

- Segundo nível de abstração: texto
  - Surge a comunicação serial!
  - Mais antiga que os próprios monitores de vídeo
  - Terminais “hard copy” (impressora com teclado, telex)
  - Com três fios já sai comunicação bidirecional

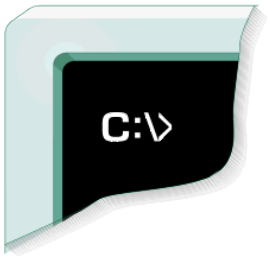


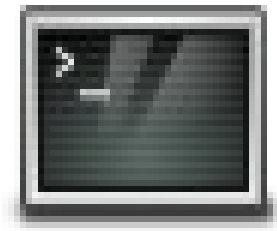
História da “Linha de Comando”

- As portas seriais mudaram a forma de INTERAGIR com os computadores
  - Fáceis de implementar com poucos componentes
  - Baratas, padronizadas, simples de usar
  - Permitiam interagir a grandes distâncias
- Operação absolutamente básica:
  - Um fluxo de caracteres de entrada, outro de saída
  - O computador boota o “monitor” ou o Sistema Operacional, e este então abre a porta serial para aceitar **comandos do usuário**
  - “console”

História da “Linha de Comando”

- O usuário interage com um programa especializado em ler linhas de texto, interpretá-las e executá-las
  - Programa avisa que está pronto (prompt)
  - Usuário digita a linha e aperta Enter (entra a linha)
- **Interpretador de comandos** em linha de texto
  - Inicialmente primitivo:
    - `E 45fa 12 17 (debug do DOS)`
  - Hoje em dia mais poderoso:
    - `for i in *.mpeg3; do mv $i ${i%mpeg3}mp3; done`





História da “Linha de Comando”

- Muita coisa hoje em dia ainda é “porta serial”
  - Terminais virtuais (Ctrl+Alt+F2)
  - Terminais gráficos (xterm, gnome-terminal)
  - Conexões SSH (socket TCP funciona igual à porta serial)
  - Console server (caindo em desuso)
  - Service processor
  - Porta serial atrás da televisão! (modelos da LG)
- Serial é fácil de usar para debugar hardware
  - Para computadores novos, adaptadores seriais USB
  - Desmonta o aparelho e procura por algo que pareça uma porta serial :-)

Shells mais comuns

- Primeiro shell Unix, “sh”, Stephen Bourne
- C Shell, csh, horrível de usar
- COMMAND.COM no tempo do DOS (CP/M) ;-)

- Hoje em dia:
  - Linux: BASH (Bourne-again Shell)
  - BSDs: KSH (Korn Shell)

- Shell, “concha”, esconde os detalhes do sistema

Interpretação e transformações

- Anatomia da linha de comando: (simples!)
  <comando> [param1] [param2] [... paramN]

- Palavras separadas por espaços
  - Primeira palavra é o comando/programa a ser executado
  - Demais palavras são parâmetros a serem passados para o comando/programa
  - Comandos podem ser implementados pelo próprio shell, ou serem executáveis instalados
  - Formas especiais podem aparecer e serem **expandidas** em qualquer ponto da linha

Interpretação e transformações

- A primeira palavra: comando ou programa
  - Geralmente é um programa no $PATH (ls, mount, rm, ...)
  - Pode ser um alias!
    - alias “ll”=”ls -lh” (mais curto, opções default, etc)
  - Pode ser uma função shell!
    - copia() { scp $* olive@servidor.com:diretório/comprido }
  - Pode ser uma primitiva de controle como if, for, while, until, case, etc
    - Implementado pelo próprio shell
  - Pode ser uma atribuição de variável
    - a=1 ou nome=”foobar” ou arqs=$(ls *.ogg)

Interpretação e transformações

- Substituições variadas:
  - Chaves:
    - Gera sequências de palavras: arquivo.{ogg,mp3}
    - Gera sequências de números: echo {1..10}
  - Til (~):
    - Diretório home do usuário: cp arquivo ~
    - Diretório home de outro usuário: ls -l ~jackl
  - Substituição de comando:
    - $() ou ``
    - echo Arquivos Ogg: $(ls *.ogg)
  - Já se perdeu? Debug do shell com “set -x”! :-)

Interpretação e transformações

- Substituições variadas:
  - Expansão aritmética: $((1 + 2)) ou $(($a * 5))
  - Expansão de nomes de arquivo: *, ?, [a-z]
    - É o mais comum, todos sabem usar
    - ls -l *.mp3
    - mv relatorio?.odt /mnt/pendrive
    - scp partes[A-F].zip user@servidor.estranho.com
    - IMPORTANTE: quem expande os nomes de arquivo é o shell, e não o programa sendo executado (diferente do DOS/Windows)
      - O programa recebe a lista pronta

Interpretação e transformações

- Substituições variadas:
  - Parâmetros e variáveis:
    - mv $teste /tmp substitui o valor da variável teste na linha
    - ${var#prefixo} expande variável removendo prefixo do valor
    - ${var%sufixo} expande variável removendo sufixo do valor
    - ${#var} expande pro tamanho da string contida em var
    - ${var:offset:length} expande a substring do valor da variável

      ...

      muitas possibilidades mais!

Interpretação e transformações

- Redirecionamentos de entradas e saídas
  - comando > arquivo.txt
    - Grava a saída do comando em um arquivo
  - comando < dados.txt
    - Roda o comando lendo dados do arquivo ao invés do teclado
  - comandofiltro < dados.txt > listafinal.txt
    - Combinando os dois
  - while read linha; do echo lalala $linha; done < dados.txt
    - Comandos complexos também podem usar redirecionamento!

Sequências de comandos

- Um depois o outro: comando1 ; comando2
- Ao mesmo tempo: comando1 & comando2
  - & é conhecido como “roda em background”
- Se um falhar roda o outro: comando1 || comando2
  - O que significa falhar? Status de saída diferente de 0!
- Se **não** falhar, roda o outro: comando1 && comando2
- Saída de um na entrada do outro: comando1 | comando2
  - **Pipelines** são a essência do shell Unix

Sequências de comandos

- Laços e condicionais:
  - Laço for:
    - **for** i **in** 1 2 3 4 5; **do** echo $i; **done**
    - **for** i **in** {1..5}; **do** ...
    - **for** i **in** $(seq 1 5); **do** ...
    - **for** arq **in** *.pdf; **do** mv $arq documento-$arq; **done**
    - **for** arq **in** $(find . -name *.pdf); **do** ...
  - O shell expande variáveis, nomes de arquivo, chaves, saídas de comandos, etc e então executa o laço com a variável indicada assumindo cada um dos valores expandidos

Sequências de comandos

- Laços e condicionais:
  - Laço while:
    - **while** comando; **do** outros; comandos; **done**
  - Executa **enquanto** “comando” retorna sem erro (status 0)
  - Laço until:
    - **until** comando; **do** outros; comandos; **done**
  - Executa **até que** “comando” retorne sem erro
    - Bom pra ir tentando até conseguir
    - **until** copiaproservidor; **do** sleep 60; **done**

Sequências de comandos

- Laços e condicionais:
  - Condicional simples, if:

```
if comando; then
    outros comandos
else
    ainda outros
fi
```

Sequências de comandos

- Laços e condicionais:
  - Condicional múltipla, case:

```
case $variável in
    valor1)
        comandos
        ;;
    valor2)
        comandos
        ;;
    *)
        default
        ;;
esac
```

Sequências de comandos

- Funções

```
soma() {
    echo Digite um número:
    read A
    echo Digite outro:
    read B
    echo A soma é $(( $A + $B ))
}
```

- Chama-se uma função como se fosse um comando normal

Shell scripts

- Arquivos de texto contendo comandos
- Mesma sintaxe que comandos digitados na CLI
- Muito usados para automatizar funções de administração dos sistemas (backup, limpezas)
- Podem ser programas bastante complexos

- Grande parte da funcionalidade dos sistemas Unix e Linux é implementada via shell scripts

Shell scripts

```
#!/bin/bash

echo -n Digite um número:
read A
echo -n Digite outro número:
read B
echo A soma é $(($A + $B))
```

Shell scripts

- Exemplos:
  - Um initscript
  - Um comando no /usr/bin
  - pretty_bug_list.sh
  - pcap_nfsstat.sh
  - rfc_mirror.sh
- Quando usar? Sempre que uma sequência não trivial de comandos possa ser útil no futuro!
  - Ou mesmo um único comando comprido demais

Shell utils, text utils

- O shell em si, por melhor que seja, implementa poucas funcionalidades
- Vários comandos simples do Unix são feitos para serem chamados pelo shell e combinados entre si
- cut, grep, sed, awk, wc, fold, head, tail, seq, sort, tr, tee, cat, ...
- Cada um é documentado em separado
- Consulte as man pages!

Referências

- Shell Script Profissional
  - Aurélio Marinho Jargas, http://aurelio.net/
- In the Beginning was the Command Line
  - Neal Stephenson, http://www.cryptonomicon.com/beginning.html
- Linux: man bash
- OpenBSD: man ksh
- Qualquer livro sobre Unix
- http://en.wikipedia.org/wiki/Computer_terminal
- http://en.wikipedia.org/wiki/Teletype
- http://www.linusakesson.net/programming/tty/index.php

Perguntas?

Dúvidas?

Participem!

OeSC-Livre.org
TcheLinux.org