Laboratório de Sistemas Operacionais (ECOS11A)

Prof Otávio Gomes (otavio.gomes@unifei.edu.br)

Roteiro 02

Terminal Linux – dialog e regex

1) Ambientação

O pacote ***dialog*** é uma ferramenta de linha de comando para criar interfaces de usuário baseadas em texto em sistemas Unix-like. Ele permite que os desenvolvedores criem menus, caixas de diálogo e formulários interativos diretamente no terminal, tornando as interações com o usuário mais amigáveis e intuitivas. Segue uma breve apresentação das funcionalidades básicas do **dialog**:

- **Menus:** O *dialog* permite criar menus interativos que permitem ao usuário fazer seleções a partir de uma lista de opções. Os menus podem ser exibidos verticalmente ou horizontalmente e podem incluir itens selecionáveis, como caixas de seleção e botões de rádio.
- **Caixas de Diálogo:** É possível exibir caixas de diálogo com mensagens de aviso, informações ou solicitações de confirmação. Estas caixas de diálogo podem incluir botões de ação para que o usuário possa responder às solicitações apresentadas.
- **Formulários:** O *dialog* permite a criação de formulários interativos onde o usuário pode inserir informações, como textos, números ou seleções de listas. Os formulários podem incluir campos obrigatórios e validações de entrada.
- **Personalização:** O *dialog* oferece várias opções de personalização, como alterar as cores e o estilo dos elementos da interface, definir tamanhos de janelas e posicionar elementos na tela.

- **Facilidade de Uso:** O dialog é projetado para ser simples de usar e pode ser integrado facilmente em scripts bash ou shell para criar interfaces de usuário interativas em scripts de automação, instalação ou configuração de sistemas.

No geral, o **dialog** é uma ferramenta poderosa para criar interfaces de usuário baseadas em texto em sistemas Linux, oferecendo uma maneira conveniente de interagir com os usuários diretamente no terminal, sem a necessidade de uma interface gráfica de usuário (GUI).

1.1) Menus

Os menus são úteis para apresentar uma lista de opções para o usuário escolher. Aqui está um exemplo de como criar um menu simples:

```
#!/bin/bash
# Exemplo de menu com o dialog
dialog --menu "Escolha uma opção:" 10 40 3 \
  1 "Opção 1" \
  2 "Opção 2" \
  3 "Opção 3" \
  2> escolha.txt
opcao=$(cat escolha.txt)
echo "Você escolheu a opção $opcao."
```

Neste exemplo é apresentado um menu com 3 opções para o usuário escolher. O comando *dialog* **--menu** cria o menu, especificando o título, altura e largura da janela, seguido das opções disponíveis.

1.2) Caixas de Diálogo

As caixas de diálogo são úteis para exibir mensagens, solicitar confirmação ou entrada do usuário. Aqui está um exemplo de como criar uma caixa de diálogo de confirmação:

```
#!/bin/bash
# Exemplo de caixa de diálogo de confirmação com o dialog
dialog --yesno "Você deseja continuar?" 10 40
# Verifica o código de retorno do dialog
if [ $? -eq 0 ]; then
   echo "Usuário escolheu SIM."
else
   echo "Usuário escolheu NÃO."
Fi
```

Neste exemplo é exibida uma caixa de diálogo de confirmação com a mensagem "Você deseja continuar?" e esperando que o usuário escolha "SIM" ou "NÃO".

1.3) Formulários:

Os formulários permitem que o usuário insira informações. Aqui está um exemplo de como criar um formulário simples:

```
#!/bin/bash
# Exemplo de formulário com o dialog
dialog --inputbox "Digite seu nome:" 10 40 2> nome.txt
nome=$(cat nome.txt)
echo "Olá, $nome!"
```

Neste exemplo é solicitado que o usuário digite seu nome em um campo de entrada e armazenando o valor inserido em um arquivo chamado **nome.txt.**

1.4) Personalização de Cores e Estilo:

O dialog permite personalizar as cores e o estilo dos elementos da interface usando as opções --colors e --backtitle. Aqui está um exemplo de como personalizar a cor do texto e do fundo da janela:

```
#!/bin/bash
# Exemplo de personalização de cores com o dialog
dialog --colors --backtitle "Meu Aplicativo" \
  --infobox "\Zb\Z4Texto em negrito e cor azul\Zn" 10 40
```

Neste exemplo, estamos usando as sequências de escape **\Zb** para tornar o texto em negrito e **\Z4** para definir a cor do texto como azul. A opção **--backtitle** define um título de fundo para a janela do **dialog**.

2) Funções em shell script

Funções em shell script permitem organizar e reutilizar blocos de código em seus scripts. Elas são definidas com a sintaxe **nome_da_funcao() { ... }** e podem ser chamadas em qualquer lugar do script. Aqui está um exemplo de como criar e usar funções em um script shell:

```
#!/bin/bash
# Definição da função de saudação
saudacao() {
   echo "Olá, bem-vindo ao meu script!"
}
# Definição da função de soma
soma() {
   resultado=$(($1 + $2))
   echo "A soma de $1 e $2 é igual a $resultado"
}
# Chamada da função de saudação
saudacao
# Chamada da função de soma com argumentos
soma 10 20
```

Neste exemplo:

- Definimos a função saudacao() que imprime uma mensagem de saudação.
- Definimos a função soma() que recebe dois parâmetros, realiza a soma e exibe o resultado.
- Chamamos a função saudacao() para exibir a mensagem de boas-vindas.

- Chamamos a função soma() com os argumentos 10 e 20 para calcular e exibir a soma desses valores.
- Quando você executa esse script, ele exibe a mensagem de saudação seguida do resultado da soma.

Funções em *shell script* são úteis para modularizar o código, facilitando a manutenção e reutilização. Elas também ajudam a tornar seus scripts mais legíveis e organizados.

3) **Atividades Práticas** – Execução de scripts

Leia, entenda e, depois, execute os scripts de **01 a 06** fornecidos pelo professor. Analise cada uma das aplicações apresentadas.

4) **Dialog + script**

Leia, entenda e, depois, execute o script *ECOS11A_Rot02.7_cadastro* fornecido pelo professor.

5) **Dialog Menu** – Execução de scripts

Leia, entenda e, depois, execute o script *ECOS11A_Rot02.8_menu.sh* fornecido pelo professor.

6) **Dialog + BIOS** – Execução de scripts

Leia, entenda e, depois, execute o script *ECOS11A_Rot02.9_BIOS.sh* fornecido pelo professor.

7) Regex

Expressões regulares, comumente abreviadas como **regex**, são padrões de busca que permitem encontrar sequências de caracteres em textos. No contexto do terminal de comandos do Linux, expressões regulares são frequentemente utilizadas em comandos como **grep**, **sed**, **awk**, entre outros, para filtrar, buscar e substituir texto de forma eficiente.

As expressões regulares permitem realizar buscas complexas em textos, como encontrar padrões específicos, extrair informações específicas ou substituir partes de um texto por outra informação.

No Linux, a maioria dos comandos que suportam expressões regulares utiliza uma sintaxe semelhante. Alguns dos operadores mais comuns incluem:

. (ponto): Corresponde a qualquer caractere único.

*: Corresponde a zero ou mais ocorrências do caractere anterior.

+: Corresponde a uma ou mais ocorrências do caractere anterior.

?: Corresponde a zero ou uma ocorrência do caractere anterior.

[]: Corresponde a qualquer caractere dentro dos colchetes.

^: Corresponde ao início da linha.

$: Corresponde ao final da linha.

\: Escape, usado para interpretar caracteres especiais literalmente.

A seguir estão exemplos de scripts que utilizam **regex**:

```
#!/bin/bash
grep -E '\([0-9]{3}\) [0-9]{3}-[0-9]{4}' arquivo.txt
```

Neste exemplo, o comando grep é usado com a opção -E para habilitar o uso de expressões regulares estendidas. A expressão regular '\([0-9]{3}\) [0-9]{3}-[0-9]{4}' busca por números de telefone no formato (xxx) xxx-xxxx no arquivo arquivo.txt.

Exemplo com sed:

```
#!/bin/bash
# Script para substituir todas as ocorrências de 'olá' por 'oi' em um arquivo
sed -i 's/olá/oi/g' arquivo.txt
```

Neste exemplo, o comando sed é usado para substituir todas as ocorrências de 'olá' por 'oi' no arquivo arquivo.txt.

Exemplo com awk:

```
#!/bin/bash
awk '{ if ($2 ~ /^a/) print $1 }' arquivo.txt
```

Neste exemplo, o comando awk é usado para verificar se a segunda palavra de cada linha começa com 'a' usando a expressão regular ^a. Se a condição for verdadeira, a primeira palavra da linha é impressa.

Outros exemplos:

- Buscar todas as linhas que contêm a palavra "Linux" em um arquivo:

```
#!/bin/bash
grep "Linux" arquivo.txt
```

- Buscar todas as linhas que começam com a palavra "Erro" em um arquivo:

```
#!/bin/bash
grep "^Erro" arquivo.txt
```

- Buscar todas as linhas que terminam com um número em um arquivo:

```
#!/bin/bash
grep "[0-9]$" arquivo.txt
```

- Buscar todas as linhas que contêm números de telefone no formato (xxx) xxx-xxxx em um arquivo:

```
#!/bin/bash
grep -E '\([0-9]{3}\) [0-9]{3}-[0-9]{4}' arquivo.txt
```

- Buscar todas as linhas que contêm um endereço de e-mail em um arquivo:

```
#!/bin/bash
grep -E '\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b' arquivo.txt
```

- Buscar todas as linhas que começam com uma data no formato dd/mm/aaaa em um arquivo:

```
#!/bin/bash
grep -E '^[0-3][0-9]/[0-1][0-9]/[0-9]{4}' arquivo.txt
```

Realize o teste destes scripts no arquivo disponibilizado pelo professor, denominado *arquivo.txt*