Universidade de São Paulo
Brasil

Aula 8
bash - parte 1

MAC0216 - Técnicas de Programação I

Professores: Alfredo, Daniel, Fabio e Kelly

Departamento de Ciência da Computação
Instituto de Matemática e Estatística

1.

O bash (Bourne-Again shell)

O bash

- É o shell padrão do GNU/Linux
- É um sucessor evoluído do sh (e bastante compatível com ele)
- Incorpora as melhores características do csh e do ksh
- Sua implementação respeita as normas POSIX (para a portabilidade entre sistemas operacionais)
- Assim como os demais shells no UNIX, o bash combina duas funcionalidades:
  - interpretador de comandos
  - linguagem de programação
- Funciona em dois modos: interativo (executa comandos digitados no prompt) e não-interativo (executa comandos lidos de um arquivo de entrada)

A linguagem de programação do bash

- O interpretador de comandos permite que usuários executem programas utilitários (`man <comando>` é muito útil)
- A linguagem de programação permite que os utilitários sejam combinados
- *Scripts* (= arquivos contendo comandos) podem ser criados, e esses arquivos têm status de novos comandos
- *Scripts* possibilitam que usuários personalizem seus ambientes, criando novos comandos para automatizar suas tarefas rotineiras de manutenção e otimização do funcionamento do computador.

Funcionamento básico de um shell

Um shell gerencia a interação entre um usuário e o SO executando (geralmente) o seguinte conjunto de passos:

1. aguarda a entrada de uma ou mais linhas de comando (que podem ser digitadas pelo usuário no terminal ou então lidas de um arquivo, chamado de script)
2. interpreta uma linha por vez, traduzindo-a em chamadas a comandos existentes no shell ou no SO
3. cria processos-filhos para executar os comandos requisitados
4. espera que a execução dos processos criados termine
5. disponibiliza o resultado da execução (para o usuário ou para um outro programa)
6. volta ao passo inicial

Utilitários para a manipulação de diretórios

- `ls <opções> <diretório>`

  Lista as entradas em um diretório.

- `cd <diretório>`

  Muda de diretório.

- `mkdir <opções> <nome>`

  Cria um novo diretório.

- `rmdir <opções> <diretório>`

  Apaga um diretório se ele estiver vazio.

Utilitários para manipulação de arquivos

- `cp <opções> <origem> <destino>`

  Copia arquivos ou diretórios.

- `mv <opções> <origem> <destino>`

  Move arquivos ou diretórios.

- `rm <opções> <arquivos>`

  Apaga arquivos ou diretórios.

- `cat <arquivo(s)>`

  Concatena arquivos e imprime na saída padrão.

- `find <local> -name <arquivo>`

  Localiza arquivos ou diretórios (Muito útil também com `-exec comando {} \;` no final)

- `sort <arquivo>`

  Ordena alfabeticamente as linhas do arquivo.

Outros utilitários muito usados

- `more <opções> <arquivo>`

  Exibe o conteúdo do arquivo, mostrando uma página por vez

- `less <opções> <arquivo>`

  Exibe o conteúdo do arquivo uma página por vez, possibilitando a navegação e outras funcionalidades (como busca de termos)

- `grep <opções> <padrão> <arquivo(s)>`

  Procura as linhas do(s) arquivo(s) que contêm o padrão indicado

- `ps <opções>`

  Exibe informações sobre os processos ativos

- `top <opções>`

  Provê uma visão dinâmica dos processos em execução

- `kill <opções> <ID do processo>`

  Envia um sinal ao processo; o sinal padrão é o sinal TERM (que solicita o término do processo)

Operadores bash para redirecionamento

- `>`

  Redireciona a saída de um comando para um arquivo em disco. Se o arquivo existir, será sobrescrito. Exemplo:

  ```
  ls -la > dir.txt
  ```

  ⇒ redireciona a saída produzida pelo `ls` para o arquivo `dir.txt`.

- `>>`

  Redireciona a saída, mas acrescentando os dados ao final do arquivo. Exemplo:

  ```
  ps ux >> dir.txt
  ```

  ⇒ redireciona a saída produzida pelo `ps` para o final do arquivo `dir.txt`, sem sobrescrever o arquivo.

Operadores bash para redirecionamento

- `<`

  Redireciona a entrada. Exemplo:

  `meu_prog < testes.txt` ⇒ faz com que o arquivo `testes.txt` forneça a entrada para o programa `meu_prog`.

- `2>`

  Redireciona a saída de erros. Exemplo:

  ```
  find -name Makefile 2> /dev/null
  ```

  ⇒ faz com que os erros produzidos pelo comando `find` sejam redirecionados para o dispositivo de sistema `/dev/null` (ou seja, as saídas de erro serão descartadas).

- `&>`

  Redireciona a saída padrão e a saída de erros. Exemplo: `ls -R /home/user/ &> /dev/null`

  ⇒ nenhuma saída aparecerá na tela, pois ambas foram redirecionadas para o "lixo" (`/dev/null`).

Operadores bash – Pipe (|)

Permite que um programa utilize como entrada a saída de outro programa. Exemplos:

- `ls -a | sort`

⇒ faz com que o comando `sort` receba como entrada a saída produzida pelo comando `ls`.

- `cat /home/eu/mac0216.txt | wc -l > lista.txt`

⇒ grava no arquivo `lista.txt` o número de linhas do arquivo `mac0216.txt` (que contém a lista de todos os matriculados em MAC0216).

- `cat /home/eu/mac0216.txt | sort >> lista.txt`

⇒ coloca no final do arquivo `lista.txt` a lista em ordem alfabética dos alunos de MAC0216.

- `ls -l /etc | sort | less`

⇒ mostra, em ordem alfabética, a listagem do diretório `/etc` e permite navegar dentro da listagem (`less`).

Outros tipos de execução de comandos no bash

- **Execução em background** – faz com que um comando seja executado assincronamente, em um subshell. Em uma execução em background, o shell não espera que a execução do comando disparado termine. É feita quando o caracter de controle '`&`' é usado no final do comando (`screen` costuma ser mais útil) . Exemplo:

```
sort mac0216.txt > saida.txt &
```

- **Execução sequencial** – comandos separados por ';' são executados sequencialmente. Exemplo:

```
sort mac0216.txt > saida.txt ; cat saida.txt
```

Outros tipos de execução de comandos no bash

- **Execução paralela** – feita por meio do comando GNU Parallel permite que comandos sejam executados paralelamente. Exemplos:

```
cat mac0216.txt | parallel -k echo Aluno:
```

Mostra o conteúdo de `mac0216.txt` incluindo o prefixo "`Aluno:`" em todas as linhas. A opção `-k` garante que a ordem das linhas seja mantida.

```
cat mac0216.txt | parallel -k echo {} ' -Aluno'
```

Mostra o conteúdo de `mac0216.txt` incluindo o sufixo "`-Aluno`" em todas as linhas.

Execução de listas de comandos no bash

- **Lista E** – é uma sequência de um ou mais comandos separados pelo operador de controle '`&&`', como em:

```
comando1 && comando2
```

  O `comando2` é executado se e somente se a execução de `comando1` terminou com sucesso (ou seja, status de saída = `0`).

- **Lista OU** – é uma sequência de um ou mais comandos separados pelo operador de controle '`||`', como em:

```
comando1 || comando2
```

  O `comando2` é executado se e somente se a execução de `comando1` não terminou com sucesso (ou seja, status de saída != `0`).

Comandos builtin do bash

Comandos builtin são comandos contidos no próprio shell, ou seja, são comandos que o shell executa diretamente, sem invocar outros programas. Alguns comandos builtin do bash:

- `history`

  Exibe uma lista dos comandos já executados

- `pwd`

  Mostra o caminho do diretório atual

- `alias <nome>=<comando>`

  Cria um “apelido” para um comando

Material recomendado

- Manual do bash
  http://www.gnu.org/software/bash/manual/bashref.html
- Guia para iniciantes do bash
  http://www.tldp.org/LDP/Bash-Beginners-Guide/html/
- HowTo da linha de comando do bash
  http://tldp.org/HOWTO/Bash-Prompt-HOWTO/
- Manual do screen
  http://www.gnu.org/software/screen/manual/screen.html