Tutorial do Bash

Guilherme Bittencourt Bueno da Silva

Universidade Federal do Paraná

gbbs14@inf.ufpr.br

3 de Setembro de 2018

Conteúdo

Introdução

Computadores são imprescindíveis em diversas áreas de aplicação, possuíndo formas específicas para cada tipo de aplicação. Para que os computadores sejam amplamente acessíveis, eles possuem uma quantidade imensa de interfaces para que um usuário sem nenhum conhecimento seja capaz de utilizar um computador. Uma interface, em ciência da computação, é a fronteira que define a forma de comunicação entre duas entidades. Ela pode ser entendida como uma abstração que estabelece a forma de interação da entidade com o mundo exterior, através da separação dos métodos de comunicação externa dos detalhes internos da operação [Wikipédia, 2018]. Existem interfaces em vários níveis, como interfaces gráficas para que usuários básicos possam selecionar aplicações, ou interfaces das próprias aplicações que permitem que o usuário faça uso do computador através de abstrações. O Shell é uma interface entre o usuário e o sistema operacional (SO), usada para simplificar o uso desse, escondendo detalhes específicos de cada versão do SO. Diferente das interfaces citadas acima, o Shell é direcionado a um público menos abrangente e não possui um sistema de apontar e clicar (que é mais lento e repetitivo que uma interface textual). A intenção dessa interface não é de facilitar o aprendizado para usuários iniciantes, e sim de facilitar e otimizar as tarefas de usuários que utilizam constantemente o sistema. Por isso, é necessário (ou, ao menos, muito importante) o uso de um material como este para aprender a usar o Shell da maneira correta, eliminando a redigitação e aproveitando ao máximo os recursos oferecidos. Quando foi criado, o Shell era a interface padrão de acesso ao computador, e nessa época, os SOs apenas tratavam do gerenciamento de processos e da memória, sem incluir aplicações e interfaces adicionais, então, para o usuário (ou, para os operadores do mainframe) escolher um bom Shell era tão importante quanto escolher um SO. Com o avanço da tecnologia, a maneira de interagir com um computador mudou tanto que muitos

usuários não precisam saber o que é um shell, a maioria dos usuários, nem precisam utilizar um shell. Considerando que há 30 anos atrás, os usuários só tinham acesso ao mainframe através de uma estação com teclado e monitor para utilizar os comandos do shell, é possível dizer que hoje, computador é algo completamente diferente. Porém, hoje em dia, ainda usamos os mesmos termos para definir partes do sistema que hoje funcionam de maneira diferente. por isso, antes de apresentar a história do Shell, é bom diferenciar conceitos que são facilmente confundidos.

Parte I
História e Conceitos relacionados a Shell

Resumo

O Shell é uma interface eficiente de acesso aos serviços do kernel. Shells mais recentes permitem operações além da simples interpretação de comandos e facilitam o uso com funcionalidades que automatizam tarefas repetitivas, porém, o custo dessa facilidade é uma interface pouco intuitiva, por isso, esse material explica brevemente a história do desenvolvimento de alguns dos Shells mais relevantes, suas estruturas e funcionalidades.

Introdução

Com o avanço da tecnologia, a maneira de interagir com um computador mudou tanto que muitos usuários não precisam saber o que é um shell, a maioria dos usuários, nem precisam utilizar um shell. Considerando que há 30 anos atrás, os usuários só tinham acesso ao mainframe através de uma estação com teclado e monitor para utilizar os comandos do shell, é possível dizer que hoje, computador é algo completamente diferente. Porém, hoje em dia, ainda usamos os mesmos termos para definir partes do sistema que hoje funcionam de maneira diferente. Então antes de apresentar a história do Shell, é bom diferenciar conceitos que são facilmente confundidos.

Terminal vs Console vs Shell

Terminal

Se você usa alguma distribuição UNIX, provavelmente já deve ter usado um terminal, mas certamente, não da mesma maneira que terminais eram utilizados antigamente. É fácil confundir e achar que o terminal está executando as operações, mas na verdade o terminal é apenas uma interface gráfica de um console. Pensando no terminal de antigamente é mais fácil entender que não é o terminal que executa os comandos. O terminal era algo físico, um ponto de acesso com meios de entrada e saída (teclado e monitor) para iteragir com a máquina, pois antes um computador era compartilhado por vários operadores. Atualmente, cada usuário trabalha em uma máquina diferente, e um terminal é a janela representada na figura 1.

Console

Então, terminal é um meio de acesso a um console, mas os comandos que você insere também não são do console, e sim do Shell, mas voltando ao console, não é necessário um terminal para acessar um console conforme a figura 2. Em um sistema linux, apertando "ctrl"+ "alt"+ "F1 , F2 , ... , F6"é possível acessar um dos consoles (F7 para sair). Hoje, isso também é conhecido como o modo texto do seu sistema operacional, pois é mais intuitivo que usar o termo console. Se um terminal era uma estação com tela e teclado, o console era a conexão física e digital entre o terminal e o mainframe. Então, Para cada sistema

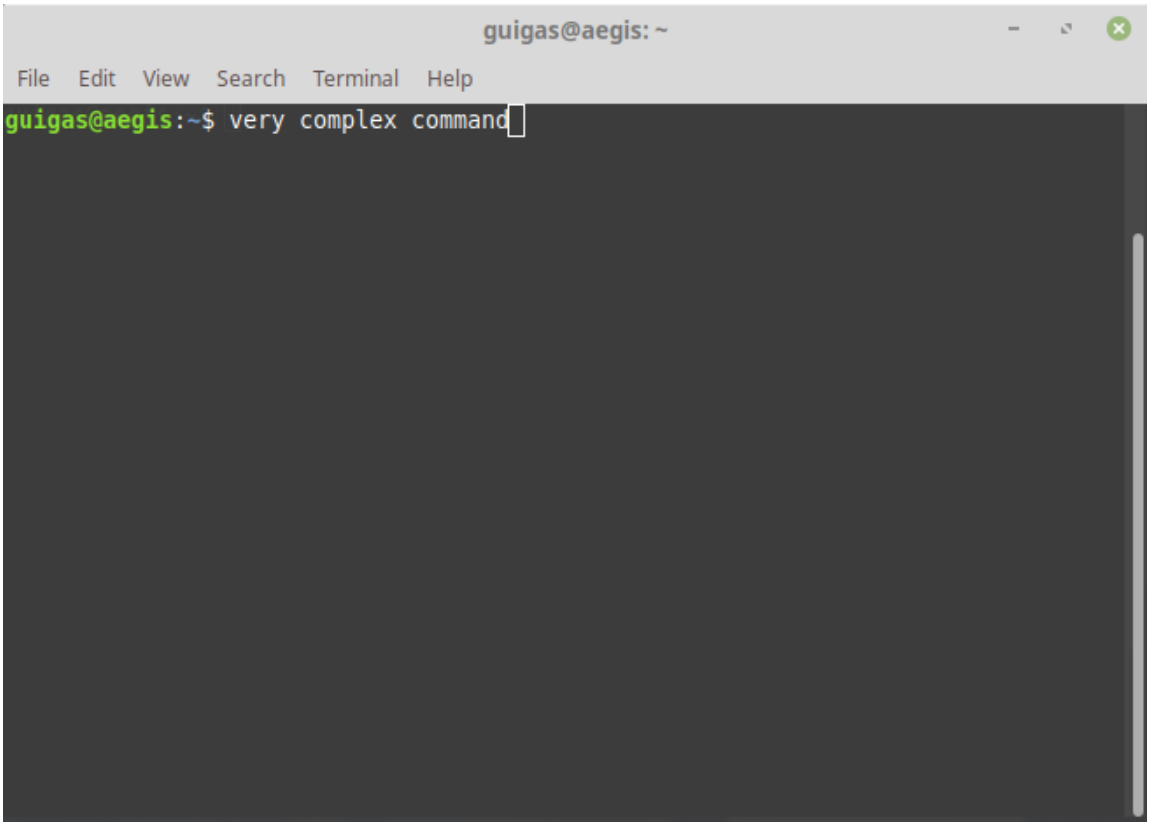

Figura 1: *A figura mostra a interface gráfica de um terminal atual.*

existe um console diferente, ligando cada sistema, a uma interface (geralmente, de texto), existem diferentes consoles que servem como interfaces para diferentes sistemas: BIOS, Boot Loader, init (processo incial do boot de sistemas unix). O console que está em uso quando você usa um terminal te da acesso ao Shell.

Shell

Shell possui esse nome pois ele funciona como uma casca separando o kernel do exterior. Sua função é servir de interface para acessar as funções do sistema operacional, tendo a vantagem de funcionar da mesma maneira em qualquer sistema operacional baseado em unix, escondendo detalhes específicos de cada SO. Portanto, os comandos que você insere no terminal, são comandos do Shell que você está utilizando, que, na maioria dos sistemas linux atualmente, é o Bash. Diferentes Shells possuem funcionalidades diferentes, possivelmente implementadas com comandos diferentes, essas funcionalidades incluem:

- Executar comandos.
- Manipulação de diretórios.
- Controle de processos (jobs).
- Expansões.
- Redirecionamento.

- pseudônimos (aliases).
- Histórico de comandos.

E muitas outras funcionalidades para facilitar, agilizar e automatizar as atividades realizadas.
Essas são algumas funcionalidades que estamos acostumados a utilizar hoje, mas nas primeiras versões de Shell, muitas delas não existiam

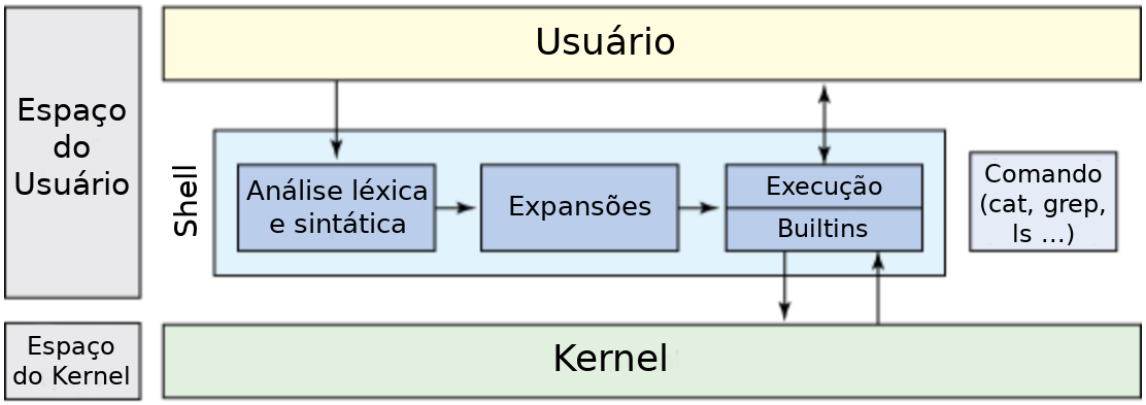

Figura 2: *Relação entre Shell, Kernel e usuário*

Unix, GNU, Linux e Shell

Em 1971, Ken Thompson desenvolveu o primeiro UNIX Shell, chamado de V6 Shell (foi criado no Unix versão 6), e suas funcionalidades muito básicas, ele permitia pipes e redirecionamentos, mas ele estava mais próximo de um interpretador de comandos, ao invés de scripts

Dessa necessidade surgem novos Shells, com novas funcionalidades. Em 1977 foi criado o Bourne Shell, com dois objetivos: Servir como um interpretador de comandos e rodar scripts. Além disso, o Bourne Shell acrescentou ferramentas de controle (até então "if"existia como uma extensão, mas não era implementado diretamente no Shell V6), loops e variáveis, facilitando o uso de comandos e a criação de scripts. A partir disso surgem muitos outros Shells, e todos seguiram essas funcionalidades. Os principais foram: C-Shell, com o objetivo de deixar os scripts mais parecidos com a linguagem C. Korn Shell, que adicionou novas funcionalidades ao mesmo tempo que manteve forte compatibilidade com o Bourne Shell, e por fim, o Bourne Again Shell (Bash), que além de manter a compatibilidade com o Bourne Shell, também implementou as funcionalidades do C-Shell e Korn shell, ele também continuou a evoluir com o tempo, porém, um dos grandes facilitadores para o Bash ser o mais usado dos 3 é o fato de ele ser um projeto Open Source do GNU.

O projeto GNU não se trata apenas de criar uma versão open source do UNIX, mas sim de criar um ambiente onde sistemas abertos respeitam especificações e padrões, de modo que os usuários tivessem a liberdade para escolher, usar, distribuir e modificar software, e para isso é necessário que todo o sofwate: Sistema operacional, drivers, programas fossem

software livre. Para isso a criação do Bash foi necessária, e com o sucesso do [GNU, 2018b], o Bash foi o Shell mais difundido.

Z Shell

Ainda existem Shells sendo criado até hoje, geralmente adicionando funcionalidades para facilitar o uso das ferramentas já existentes no Bash, por exemplo, o Z shell (zsh) permite que usuário selecione o resultado das expansões de nomes de arquivos antes de executar o comando, além disso, quando o usuário digita parte de um caminho da árvore de diretórios, o zsh deixa visível os arquivos e diretórios visíveis à partir do caminho digitado, assim, unindo a intuitividade da interface gráfica com a agilidade da interface textual.

Conclusão

A maior evolução de qualquer Unix Shell foi a ideia de transformar o shell em uma ferramenta para Scripts, e não apenas um interpretador de comandos, essa mudança surgiu no Bourne Shell, e é bem provável que, qualquer que seja o Shell mais comum no futuro, continue sendo algum derivado do Bourne Shell.

Parte II
Uso Básico do Bash

Resumo

As próximas seções explicam o sistema de arquivos e diretórios e maneiras de navegar entre eles, além do uso de pipelines, expansões e ferramentas de entrada e saída para usar o Bash como um interpretador de scripts, e não apenas de comandos separados.

Introdução

Se você já precisou usar comandos do Bash para buscar partes específicas de um arquivo de texto, você sabe que muitas vezes é necessário usar mais de um comando, e simplesmente não é prático usar vários arquivos intermediários para chegar no seu resultado final. O próximo parágrafo desse material explica como usar qualquer comando, porém, o Bash não se tornou o Shell mais utilizado por ser um simples interpretador de comandos.

Estrutura dos comandos do Bash

Uma linha de comando consiste de uma ou mais palavras, separadas por espaços. A primeira palavra é o próprio comando, seguido de um ou mais parâmetros(alguns comandos não exigem nenhum parâmetro). Um tipo comum de parâmetros são as opções, geralmente na forma de um traço seguido de uma letra. Algumas opções têm seus próprios parâmetros. Para exemplificar, vamos usar um comando que ordena linhas, o "sort". Suponha que queremos ordenar as linhas de um arquivo chamado lista.txt e queremos uma ordenação numérica, para isso usamos o seguinte comando:

```
sort -n lista.txt
```

Neste caso, "sort"é o comando, -n"é a opção que indica ordenação numérica e o "lista.txt"é o parâmetro que seleciona qual arquivo deve ser ordenado. Para saber quais opções e quantos parâmetros um comando pode ter, utilize o comando man, que também detalha o funcionamento do comando, então, em caso de dúvidas sobre como usar algum comando, leia a man page. basta digitar:

```
man comando # troque "comando" pelo comando a ser  consultado
```

Neste caso, "man"é o comando e "comando"é o parâmetro para escolher qual man page deve ser consultada. Além disso, o conteúdo da linha à partir do '#' é lido como comentário.

Arquivos e Diretórios

Em sistema UNIX, o conceito de arquivo é bem amplo, não há nenhuma estrutura definida, o significado da sequência de bytes armazenada em um arquivo, é dependente de qual aplicação o utiliza. O nome do arquivo também não é definido. Todo tipo de padrão

como headers, conteúdo e extensões(".txt, .mp3") são maneiras de organizar os arquivos. É possível separar os tipos de arquivos em 3:

- Arquivo de texto, que contém caracteres legíveis.
- Arquivo executável, o conteúdo desse tipo de arquivo não são caracteres legíveis, então não é uma boa ideia abrir um arquivo como esse em um editor de texto.
- Diretório. Conhecidos pela abstração "pasta". São criados para guardar outros arquivos (qualquer tipo, até outros diretórios).

Navegação

Uma abstração importante dos sitemas UNIX é a árvore de diretórios. Mesmo que os dados não sejam armazenados dessa forma, diretórios possuem uma estrutura hierárquica, onde um diretório é o pai de todos os diretórios que ele contém. Diretórios são irmãos, se possuem o mesmo pai, além disso, diretórios possuem apenas um pai. O diretório raiz é chamado "root".

Para poder navegar pela árvore de diretórios e não precisar saber ou digitar o caminho completo de um arquivo, existe o conceito de diretório atual, assim, é possível se referir a arquivos tanto de maneira relativa a esse diretório quanto de maneira absoluta(partindo do root, para isso, o caminho deve começar com uma barra (/)). Os caminhos são compostos de diretórios separados por barra. Para que um caminho seja válido, os diretórios em sequência, devem possuir uma conexão na árvore de diretórios.

Além disso, Todo diretório possui uma conexão ao diretório pai (..) e a si mesmo (.), assim, é possível acessar qualquer diretório, a partir de qualquer outro diretório, porém, caso o diretório corrente seja um diretório profundo de um ramo da árvore, pode ser mais rápido acessar o diretório desejado pelo seu caminho absoluto, partindo da raiz, ao invés de retornar vários diretórios usando "..".

Além da árvore de diretórios e do diretório atual, no UNIX, cada usuário possui um diretório "home", esse diretório serve para separar os arquivos do sistema, executáveis de programas, e outros arquivos, dos diretórios pessoais do usuário. Convenientemente, um usuário logado possui um atalho para a sua home, sendo assim, ele sempre pode se referenciar a sua home com o caractere TIL (~). Exemplos:

```
/home/usuario/Documentos
../../media/dispositivo
~/Downloads
```

Os comandos da tabela 1 são os mais usados para navegação na árvore de diretórios:

O comando ls pode ser usado no diretório atual ao omitir o parâmetro de localização, existem opcionais para mostrar metadados e incluir arquivos ocultos (arquivo cujo nome começa com ponto (.)). O comando cd aceita tanto caminho absoluto quando caminho relativo. Ao omitir o parâmetro de localização o diretório atual será a home. Mais detalhes sobre as opções e parâmetros dos comandos podem ser encontrados nas man pages. O mesmo vale para os comandos apresentados até o fim desse material. É indicado que o leitor teste o que é apresentado nesse material, lendo as man pages e testando opções dos comandos.

Nome do comando	Função
pwd	Retorna o diretório atual
ls	Lista os arquivos de um diretório
cd	Muda o diretório atual
mkdir	Cria um novo diretório

Tabela 1: *Comandos de navegação*

Dica: Utilize o auto-complete para evitar digitação desnecessária! TAB para completar quando há apenas uma opção possível e TAB duas vezes para mostras as opções possíveis, tanto para completar comandos quanto para nomes de arquivos e diretórios existentes e acessíveis no caminho digitado(na verdade, os comandos também são arquivos, os comandos do Bash ficam no diretório /bin).

Expansão

A seção anterior detalha como navegar na árvore de diretórios, porém, muitas vezes não mantemos um mapa mental completo da árvore e nesse caso a navegação, mesmo com auto-complete ainda é muito precária. Esse é um dos usos dos caracteres coringas, ou, "meta-caracteres".

O asterísco (*) substitui qualquer sequência de caracteres e o ponto de interrogação (?) substitui um único caractere. Então, suponha que em seu diretório atual, existem os seguintes arquivos: b16c8a16.txt, b4c4a32.txt e b128c2a8.txt, suponha também uma infinidade de arquivos com nomes variados de modo que ls não é a forma mais rápida de identificar arquivos. com o asterísco, é possível selecionar um arquivo sabendo apenas parte de seu nome, por exemplo, o comando "ls b16*"retornaria apenas o arquivo b16c4a32.txt (caso nenhum outro arquivo do diretório corrente comece com b16). o comando "ls b1?c"retorna apenas o arquivo b16c8a16.txt. Usando um "*"no lugar do "?", o comando retornaria tanto o arquivo "b16c8a16.txt"quanto o "b128c2a8.txt".

Suponha agora que você quer imprimir o conteúdo de todos os arquivos que acabam em ".txt". O comando cat imprime o conteúdo de um arquivo na tela. Caso o usuário não saiba o nome de todos os arquivos(ou apenas não queira digitar todos), mesmo que sejam muitos arquivos, ainda é possível realizar essa tarefa apenas com o seguinte comando:

```
cat *.txt
```

Com os comandos mv(mover) e cp(copiar) é possível realizar rapidamente tarefas bem comuns de gestão de arquivos. É indicado que o leitor crie um diretório para testar comandos com '*' e '?', em especial com o comando rm (remover). Cuidado com o uso de ".*", lembre dos diretórios . e .. antes de executar comandos destrutivos.

Com o uso dos colchetes ('[' e ']') é possível criar coringas mais específicos para a expansão de caminhos, por exemplo, a expansão *.[abc] funciona para todos os 3 casos:

"*.a", "*.b"e "*.c". A tabela 2 possui mais exemplos de expansão, lembre que a expansão de colchetes corresponde a um único carectere:

Expansão	**corresponde à**
[b-e.;!]	Qualquer letra minúscula de "b"a "e", ".", ";"e "!"
[!b-e]	Qualquer dígito, exceto letras minúsculas de "b"a "e"
[A-Z0-9]	Qualquer letra maiúscula de "A"a "Z"e qualquer número de um dígito
[a-df-i]	a, b, c, d, f, g, h, i
[abc-]	'a', 'b', 'c', ou '-'

Tabela 2: *Expansão de caminhos*

Expansão de caminhos é o primeiro passo para fazer scripts em Bash, com eles é possível realizar ações para várias entradas, de uma só vez. Porém, nem todo parâmetro é um nome de arquivo ou diretório, para isso existe a expansão de chaves ('{' e '}'), que permite a expansão para um conjunto de strings, separadas por ','. Apesar de funcionar de maneira parecida, nesse tipo de expansão, não há a necessidade de corresponder com nomes de arquivos existentes. A tabela 3 contém exemplos de uso da expansão de chaves:

Expansão	**Resultado**
cat func.{c,h}	conteúdo dos arquivos fun.c func.h
ls rec0{1..4}.mp3	rec01.mp3 rec02.mp3 rec03.mp3 rec04.mp3
touch 201{5..7}/ex{1..4}.txt	cria ex1.txt, ex2.txt, ex3.txt e ex4.txt nos diretórios 2015, 2016 e 2017
echo turma{A..D}	turmaA turmaB turmaC turma D
echo turma{A..G..2}	turmaA turmaC turmaE turma G

Tabela 3: *Expansão de chaves*

Se necessário, consulte as man pages dos comandos echo e touch. Com a expansão de chaves é possível expandir caminhos que não existem, então dependendo do caso você pode obter respostas de erro do comando utilizado.

As mensagens de erro dos comandos do Bash são bem intuitivas, antes de ir ao google, leia a mensagem de erro, e se necessário a man page. Para os casos comuns, isso basta.

Em caso de dúvidas nos resultados das expansões, use o comando echo para conferir se a expansão gera o resultado esperado, antes de usar a expansão em comandos que podem estragar ou mover arquivos importantes. Lembre-se da diferença entre a expansão de chaves da expanção de colchetes.

Redirecionamento de Entrada e Saída

Os redirecionamentos são o próximo passo da evolução de comandos para scripts. As expansões são muito úteis, mas nem tudo pode ser resolvido com apenas um comando. Uma ótima abstração para entender a funcionalidade do redirecionamento no Bash é pensar nos comandos como filtros. Um comando recebe uma entrada e produz uma saída, até esse ponto do material a entrada era sempre lida pelo teclado (stdin - Standard Input) e imprimidas na tela (stdout - Standard Output), porém, se é possível usar comandos como filtros, podemos redirecionar a saída de um comando para a entrada de outro, e assim, combinar vários comandos para obter a saída desejada, com uma estrutura parecida com essa:

```
entrada -> Filtro1 -> Filtro2 -> ... -> FiltroN -> saída
```

A notação para redirecionamento de entrada é '<' e, '>' para saída. Então, o seguinte exemplo redireciona a saída do comando cat para o arquivo teste.txt:

```
cat texto.txt > teste.txt
```

Também é possível redirecionar a entrada de um comando. Suponha um programa que lê na entrada padrão. Caso hajam muitas entradas de teste ou caso a entrada seja muito grande. É possível redirecionar um arquivo como entrada dessa forma:

```
./programa < entrada1.txt
```

Observação: './' é a maneira de executar um arquivo executável (tente './ls' no diretório '/bin').

A última ferramente para usar programas e comandos como filtros, como exemplificado anteriormente é o pipeline (|). O pipeline elimina a necessidade de arquivos intermediários e liga diretamente a saída de um programa a entrada de outro conforme o exemplo:

```
ls /dev | grep std | tee file1.txt file2.txt file3.txt
```

O resultado desse comando imprime 3 arquivos: 'stdin', 'stdout' e 'stderr'. O terceiro ainda não foi apresentado nesse material. stderr é, por convenção, o local para imprimir warnings e erros dos programas, como um log. O comando grep seleciona apenas as linhas do resultado que contém a substring passada como parâmetro. O comando tee redireciona a entrada para todos os arquivos passados como parâmetro.

Um uso comum do pipeline é filtrar diferentes dados dos nomes de arquivos partindo do comando ls, ou do conteúdo desses, partindo do comando cat, e utilizar os comandos grep, cut e tr, respectivamente para filtrar linhas específicas, filtrar seções de uma linhas e substituir caracteres, e, ao final é possível redirecionar a saída para um arquivo.

EXERCÍCIO

1. Utilizando apenas um comando mkdir e expansão de chaves, crie os 7 diretórios com os seguintes nomes: 3 diretórios para turmaAP, turmaREP e turmaFN e os 3 diretórios turmaA, turmaB e turmaC 1 diretório de nome alunos. Dentro de um diretório seguro para testes.

Resposta:

```
mkdir turma{{A..C},{AP,REP,FN}} alunos
```

2. dentro do diretório alunos, crie arquivos de nome de alunos no formato aluCnotaN para cada curso C entre: BCC, IBM, TADS e OUTRO e para cada nota N entre 0 a 10 e NULL. Exemplo de arquivo: aluBCCnotaNULL

   Resposta:

```
touch alunos/alu{BCC,IBM,TADS,OUTRO}nota{{0..10},NULL}
```

3. Copie os arquivos do diretório alunos para o diretório correspondente de acordo com a tabela 4:

Exercício	**Arquivos**	**Copiar para**
3.1	Alunos de BCC com nota diferente de NULL	turmaA
3.2	Alunos de IBM com nota diferente de NULL	turmaB
3.3	Alunos de TADS com nota diferente de NULL	turmaC
3.4	Alunos de BCC, IBM e TADS com nota de 0 a 3	turmaREP
3.5	Alunos de BCC, IBM e TADS com nota de 4 a 6	turmaFN
3.6	Alunos de BCC, IBM e TADS com nota de 7 a 10	turmaAP

Tabela 4: *Relação de arquivos e destinos*

Resposta:

```
cp alunos/aluBCCnota[!A-Z]* turmaA/
cp alunos/aluIBMnota[!A-Z]* turmaB/
cp alunos/aluTADSnota[!A-Z]* turmaC/
cp alunos/alu{BCC,IBM,TADS}nota[0-3] turmaREP/
cp alunos/alu{BCC,IBM,TADS}nota[4-6] turmaFN/
cp alunos/alu{BCC,IBM,TADS}nota{7..10} turmaAP/
```

Conclusão

As ferramentas do Bash apresentadas nesse material, juntamente com estruturas de controle (if e for) e variáveis, permitem que sejam criados scripts que interagem dinamicamente com os dados sem excesso de digitação e trabalho por parte do usuário. Para que esse conteúdo seja entendido pelo leitor é ideal que os conhecimentos sejam aplicados na prática. Para isso, são indicados os materiais do professor Roberto Hexsel [Hexsel, 2018b, Hexsel, 2018a], além de serem bem explicativos, possui vários exemplos, explicam novos comandos e possuem alguns exercícios para praticar.

Parte III
Customização do ambiente

Resumo

A seção seguinte explica os métodos de customização do Shell, sem a intenção de listar e detalhar as opções, porém, bibliografias para consulta das opções serão indicadas. Alguns desses métodos são utilizados por outras aplicações como editores de texto.

Introdução

As duas principais maneiras de customização do bash são opções e variáveis de ambiente, algumas opções são definidas com o comando set, outras com o comando shopt (shell options), e as variáveis de ambiente são variáveis definidas sempre que o bash inicia a execução. Opções e variáveis podem ser alteradas pelo usuário a qualquer momento, mas, para que não seja necessário configurar o bash toda vez ao abrir um bash, elas podem ser definidas nos arquivos: .bash_profile e .bashrc. Para explicar o funcionamento desses arquivos, antes é necessário definir as variáveis do Bash.

Variáveis e Aliases

Para auxiliar os scripts no Bash, variáveis servem para guardar qualquer tipo de valor, incluíndo retorno de programas[1]. A sintaxe para definir variáveis é: **`nome=valor`**, sem espaços antes ou após o '=', e se o valor contém um ou mais espaços, o valor deve estar entre aspas[2]. Para obter o valor de uma variável, a sintaxe é **`$nome`**.

De maneira similar é possível definir aliases, para encurtar comandos muito usados que são difíceis de digitar. Por exemplo, é possível definir que o comando **`pr * | lpr`** será executado ao digitar **`pa`**, usando o comando: **`alias pa='pr * | lpr'`**. É importante ressaltar que aliases podem ser usados apenas no início de um comando, e apesar de existirem outras formas de contornar os problemas gerados por essa limitação, nas próximas partes desse material, serão apresentadas as funções, que permitem mais flexibilidade que os aliases.

Os arquivos de configuração e Run-Control

Existem dois arquivos para customizar o shell, o primeiro deles é o profile, situado no diretório home de cada usuário, geralmente usado para definir variáveis de ambiente e opções, como definições padrão e escolhas de método de funcionamento do bash. Uma lista de variáveis pode ser encontrada, digitando:

```
compgen -v | while read line; do echo \$line=\${!line};done
```

[1] Para atribuir o retorno da execução de um programa em uma variável, deve-se englobar a execução no seguinte formato: variavel=$(execução)

[2] Aspas simples para o valor literal da string e aspas duplas para realizar expansões e traduzir valores de variáveis dentro do conteúdo

O comando while está sendo utilizado apenas para imprimir conteúdos de variáveis que contém mais de uma linha. Mais detalhes sobre o comando while serão tratados nas próximas partes desse material. A lista de opções pode ser encontrada no manual de referencia do bash[GNU, 2018a], nas seções 4.3.1 e 4.3.2. Existem muitas opções e variáveis de ambiente, por isso, é mais produtivo que o autor procure por elas de acordo com a necessidade de alguma configuração específica. Esse não é o caso para a variável $PATH, pois algumas aplicações exigem que o usuário altere seu conteúdo, essa variável contém os caminhos dos arquivos executáveis, então se você quer tornar executáveis próprios acessíveis sem a necessidade de referenciar o caminho até o executável, é indicado criar um diretório para esses executáveis, e adicionar o caminho desse diretório à variável $PATH. Para adicionar um caminho à variável PATH sem sobrescrever seu conteúdo, atribua **`PATH=$PATH":/home/caminho/do/diretorio"`**.

Outro arquivo de customização do bash é o .bashrc, também situado na home do usuário. Esse arquivo geralmente contém definições de preferências pessoais, como mudar o prompt, ou mudar as cores além da definição de aliases. O .bashrc é um arquivo de run-control, usados por diversas aplicações como um arquivo de declarações e comandos, associados ao programa que o interpreta na inicialização[3]. O processo de inicialização, que inclui como o bash interpreta as definições desses dois arquivos é detalhada na seção seguinte.

O conteúdo dos dois arquivos de configuração é apenas uma convenção de boa prática para facilitar a identificação dessas configurações. Idealmente as alterações devem ser seguidas de comentários. Como dito antes, esse material não descreve cada opção de configuração, isso porque são tantas opções que dificilmente o leitor não lembraria de cada detalhe quando seu uso fosse necessário, o mesmo acontece para para os arquivos de configuração, caso seja necessário retornar à configuração, um comentário relembrando a necessidade de cada linha é muito bem vindo.

Ainda existe o arquivo .bash_logout para as definições e principalmente execução de scripts que é executado ao encerrar a execução do Bash.

Inicialização do Bash

Para reforçar, um shell interativo é aberto ao shell através de um terminal, e quando o acesso ao shell exige um login e senha, como no acesso através do modo texto ou ssh, esse é chamado de login shell.

Ao iniciar um login shell, após a autenticação, o shell executa o script /etc/profile, que inicializa variáveis necessárias e executa os scripts (*.sh) do diretório profile.d. Qualquer alteração nesse procedimento deve ser incluída como um script nesse diretório, para que alterações no /etc/profile não sejam sobrescritas em alguma atualização do sistema. Por fim, o perfil pessoal do usuário, que, devido à retro-compatibilidade do bash, pode possuir diferentes nomes, será executado. A ordem de prioridade entre os arquivos de perfil de

[3]Geralmente sistemas possuem um arquivo de run control com definições compartilhadas para todos os usuários, e outro, na home de cada usuário, para definições pessoais

usuário, da maior para a menor, é:

- .bash_profile
- .bash_login
- .profile

Sendo que apenas o arquivo encontrado de maior prioridade será executado.

Ao final da execução do perfil do usuário, o arquivo .bashrc é executado (lembre-se de manter essa definição, ao editar seu profile), também contendo definições específicas do usuário. Em alguns sistemas, também é executado o bashrc compartilhado a todos os usuários.

Para a inicialização de um Shell interativo o processo é mais simples: Os arquivos bashrc são executados (o arquivo para todos os usuários e o específico do usuário que abriu o bash), mesmo que essa execução não seja especificada no .profile, pois a execução do bashrc é realizada diretamente pelo shell interativo.

CONCLUSÃO

Como esperado, o bash não possui um menu gráfico com botões e opções de configuração, ao invés disso, o usuário pode configurar o bash através de variáveis pré-definidas e opções (set e shopt). Para que não seja necessário configurar o bash toda vez que o bash é inicializado, existem os arquivos de perfil e o run control .bashrc. Pela maneira que esses arquivos são executados, suas alterações não serão refletidas em shells iniciados antes da alteração.

O uso de arquivos run control não é exclusivo do bash e se encontra até em editores de texto como o Vim, a lista de opções de configuração do vim podem ser encontradas no manual de referência do vim [vim, 2018]. O escopo desse material é apenas de informar os meios de configuração e o funcionamento do sistema de configuração. É ideal que o leitor se acostume com o fluxo de encontrar uma necessidade e buscar uma configuração para atender essa necessidade.

Referências

[GNU, 2018a] GNU (2018a). Bash reference manual. https://www.gnu.org/software/bash/manual/html_node/index.html#SEC_Contents. Acessado em 03-09-2018.

[GNU, 2018b] GNU (2018b). Manifesto gnu. https://www.gnu.org/gnu/manifesto.pt-br.html. Acessado em 18-08-2018.

[Hexsel, 2018a] Hexsel, R. A. (2018a). Como computeiros ganham tempo com bash. http://www.inf.ufpr.br/roberto/ci064/labBash-1.html. Acessado em 18-08-2018.

[Hexsel, 2018b] Hexsel, R. A. (2018b). Unix para computeiros. http://www.inf.ufpr.br/roberto/ci064/labUnix.html. Acessado em 18-08-2018.

[vim, 2018] vim (2018). Vim reference manual. http://vimdoc.sourceforge.net/htmldoc/options.html. Acessado em 03-09-2018.

[Wikipédia, 2018] Wikipédia (2018). Interface (ciência da computação). https://pt.wikipedia.org/wiki/Interface_(ci%C3%AAncia_da_computa%C3%A7%C3%A3o). Acessado em 18-08-2018.