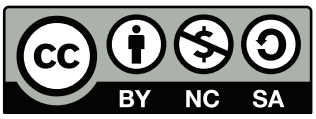
How to Bash

Rafael de Paulo Dias

28 de novembro de 2018



Sumário

1 Resumo

Este trabalho procura apresentar as ferramentas essenciais para que o usuário linux tenha habilidade para trabalhar com o bash. Serão demonstrados comandos, atalhos e algumas dicas da sua utilização e também será apresentado no capítulo Trabalhos Práticos algums exemplos de aplicação do bash.

2 Introdução

Este texto foi desenvolvido durante as aulas de 'Tópicos em Programação de Computadores (CI320)', com a intenção de apresentar ao aluno do curso de Ciência da Computação o bash, visando ir além de um simples terminal, mas como uma ferramenta de trabalho, ferramenta de busca, execução de scripts e mais.

Após a leitura do trabalho o aluno conseguirá ganhar muito tempo, principalmente com funções **awk**, **regex** e **grep**. De início parecem complicadas mas o aluno deve usar os exemplos apresentados no trabalho para consolidar o conhecimento.

3 Guia Fundamental para Iniciantes em Linux

Os comandos são uma forma poderosa de manusear o Linux e ajuda a evitar repetição de linhas de comandos. Também serão apresentados os argumentos, diretórios, coringas, entrada, saída e pipeline, todas ferramentas que são importantes tanto para o trabalho quanto para o uso diário.

O Bash é uma ferramenta de comunicação entre os serviços do sistema operacional e usuário. O bash tem seu nome proveniente de bourne again shell, que na verdade é uma versão do shell criada para ser utilizada como interface do sistema operacional.

Para poder acompanhar os comandos, é necessário acessar o terminal utilizando o atalho "Ctrl + Alt + t" para abrir o terminal, ou "alt + F2" e digite "gnome-terminal". Em alguns exemplos de comandos será utilizado '**$**', o cifrão, para representar a partir do terminal, digite o comando que está posterior a ele. Alguns exemplos também utilizarão os comandos "**ls**" que lista os diretórios da pasta atual, e "**cd**" que navega entre os diretórios.

3.1 Resumo

Algumas das ferramentas mais utilizados no dia-a-dia do usuário Linux serão apresentadas através de comandos e seus argumentos, também será apresentado o uso da man page para conhecer as possíveis formas de usar um determinado comando. Além disso será demonstrado exemplos de navegação, visto que a todo momento você pode andar diretório a diretório ou utilizar os atalhos que serão apresentados no texto.

Será também realizado uma apresentação do unix que é muito significativo para explicar a criação e sua relação com o shell, com suas características e alguns modelos devido à grande quantidade que existe como ash, bash, dash, fish, zsh.

3.2 Shell

O shell é uma forma de processar comandos, no caso um programa que lê e interpreta comandos que você usa como entrada. Pode-se utilizar entradas que relembram, editam e utilizam comandos usados recentemente (HAHN, 1993).

Além de ser um interpretador de comando o shell é também uma linguagem de programação.

Existem diversas variações do shell, essa variedade de opção tem direta relação com a quantidade de unix e suas ramificações, como exemplo a relação do sh com bash, este segundo foi gerado para complementar o primeiro e também para ser utilizado no Linux. Existem dois tipos de shell a versão gráfica (GUI) e a versão baseada em texto (CLI).

Será apresentado abaixo em ordem alfabética alguns exemplos de shell e informações extra.

- ash

  - nome : Almquish shell;

  - clone do bash, porem uma versão mais leve;

  - padrão do Ubuntu

- bash

  - nome : Bourne Again shell;

  - modelo software livre baseado no sh, utilizado como padrão no Linux;

- csh

  - nome : C shell;

  - Iniciou o uso de aliases e trouxe comando history, baseado na linguagem shell;

  - Lançado com a distribuição do unix BSD (Berkeley Software Distribution);

- dash

  - nome : Debian Almquist shell;

  - Debian criou uma versão do ash;

  - Ubuntu usa como default para operações não iterativas, e normalmente é 4 vezes mais rápido que o bash;

- fish

  - nome : Friendly Interactive shell;

  - conforme o próprio nome indica, é um shell que procura ser mais amigável com novos usuários com auto-completar e auto-sugestão baseado no seu histórico;

- ksh

  - nome : Korn shell;

  - Baseado no C shell;

  - Inicialmente software fechado, 20 anos mais tarde se tornou código aberto nomeado como pdksh;

- sh

  - nome : Bourne shell;

  - padrão da versão 7 do unix, compatibilidade com a maioria dos unix;

  - substitui PWB shell;

  - por 29 anos obteve suporte e atualização;

- PWB shell

  - nome : Mashey shell;

  - implementação do if e goto;

- tcsh

  - nome : "t"proveniente do sistema operacional Tenex;

  - Versão com melhorias de bug do csh;

- Zsh

  - nome : "z"derivado do professor de Yale Zhong Shao;

  - Versão bash que possui *features* encontradas em bash, kshe e tcsh;

  - Possui uma comunidade que desenvolve plugins em constante crescimento conhecida como “Oh My Zsh!"

3.3 Bash vs Zsh

O Zsh possui sugestão ao pressionar TAB, auto-completar, auto-corretor que não é *case-sensitive*, possui plugins como git que indica no terminal a *branch* que o usuário se encontra e se houve alguma modificação pendente de commit.

O Bash acompanha os distros do linux como default. E também possui compatibilidade com a maioria dos shell's.

3.4 Comandos

A melhor maneira de aprender comandos através do sistema operacional Linux é utilizando a man page. Para acessar a man page, é necessário digitar **$ man** “comando”, existe um guia de como utilizar o próprio man através do comando $ man man.

Alguns comandos básicos para se iniciar no Linux são:

- “**ls**” lista os diretórios e arquivos;
- “**cd**” navega entre os diretórios;
- “**mkdir**” cria uma pasta;
- “**rmdir**” exclui uma pasta;
- “**vim**” editor de texto;

3.5 Diretórios

Os diretórios do Linux possuem uma estrutura definida através de uma árvore e cada diretório possui seu propósito. No terminal ao utilizar o comando **$ cd** ‘/’, você se encontra na raiz dos diretórios. será apresentado abaixo a funcionalidade de alguns diretórios filho da raiz.

- /bin Contém programas executáveis que são necessários para carregar ou reparar o sistema;
- /boot Contém arquivos que são necessário durante o processo de boot do sistema;
- /dev Contém os dispositivos;
- /etc Contém arquivos de configuração para o usuário local;
- /home Contém arquivos relacionado ao usuário atual da máquina;
- /lib Contém as bibliotecas compartilhadas do sistema;
- /srv Contém arquivos relacionados a dados de sites específicos;
- /tmp Contém arquivos temporários;

A lista de outros diretórios podem ser encontradas através do comando **$ man hier**.

3.5.1 Navegando nos diretórios

Para navegar através do terminal será utilizado o comando **$ cd** "endereço". O endereço pode ser qualquer repositório em que se possua acesso, alguns atalhos para a navegação nos diretórios bastante úteis são o '~' e '/', o primeiro irá direcionar ou considerar que você esteja na sua home o segundo na raiz dos diretórios. Outro comando para ajudar na navegação é "../" que irá retornar para o diretório anterior, e a tecla "tab" ajuda a verificar se um arquivo ou diretório está no diretório atual.

3.6 Coringas

Os coringas existem para facilitar os comandos, exemplos:

- '?' Substitui qualquer letra;
- '*' Substitui qualquer conjunto de letras;
- "[a-z]" Substitui qualquer letra contida no intervalo;
- "[!a-z]" Substitui qualquer letra não contida no intervalo;

Alguns exemplos dos coringas utilizados acima, supondo que exista uma diretório onde você se localize no momento.

Existem os itens {a.pdf, b.pdf, c.txt, d.out, el.php, le.int}.

$ ls ?.pdf, resulta em {a.pdf, b.pdf}

$ ls *, resulta em {a.pdf, b.pdf, c.txt, d.out, el.php, le.int}

$ ls [e-l].* resulta em nenhum pois os elementos que contidos no intervalo possuem duas letras.

$ ls *[e-l].* resulta em {el.php, le.int}

Uma forma de visualizar o que acontece ao utilizar os coringas é interpretar que os comandos são expandidos(NEWHAN, 2005).

3.7 Entrada, Saída e Pipeline

As interpretações de entrada e saída do unix são respectivamente teclado e monitor. Por exemplo o comando "grep" que procura padrão em um conjunto de letras, um exemplo de sua utilização será encontrar um comando que foi utilizado no passado. E através do comando **$ history** |**grep** "comando passado", ira listar no monitor todas as combinações que foi utilizado com o comando passado.

Um símbolo especial foi utilizado no ultimo exemplo '|' " shift + barra " invertida, serve para utilizar a saída do primeiro comando que foi "history" e utilizar como entrada do comando posterior ao pipeline "grep".

3.7.1 Redirecionamento

São usados como redirecionamento as teclas '<' e '>'. Pode-se através das teclas citadas anteriormente adicionar o conteúdo em um arquivo ao invés de apresentá-lo na tela através do comando **$ man hier** >hier.txt, o resultado da busca será armazenado no arquivo txt. E ao criar um código por exemplo de ordenação, se entrada utilizada são dez números, pode-se inserir manualmente os dez, para toda vez que for fazer o teste ou criar um arquivo que contenha 10 números e utilizar o comando **$./ordena** '<' dez.txt.

3.8 Conclusão

Existem de vários modelos de shell, o usuário (Linux) tem a liberdade de testar, analisar e editar o código como se sentir a vontade com os que são software livre. Esta pesquisa também buscou demonstrar as ferramentas

mais básicas porém essenciais do shell. Os comandos apresentados são um guia para o usuário utilizar diariamente, não a necessidade de memorizar todos os comandos, e sim aprender pelo uso.

3.9 Questões

1. (fácil) Os comandos do shell acompanham o dia a dia do usuário, portanto os mais básicos serão aprendidos aos poucos conforme sua utilização. A atividade requer a utilização dos comandos **ls**, **cd**, e **man**.

Resposta:

Livre

2. (médio) O comando **$ echo $SHELL** serve para apresentar qual shell esta sendo utilizado. O desafio proposto é selecionar um shell apresentado no capitulo Shell, então utilize o comando **$ man apt-get**, e descreva o comando "install", que é bastante utilizado automaticamente em seguida baixe o shell de sua escolha.

Resposta:

O comando apt-get é utilizado para manusear pacotes utilitários. O comando "install", pode instalar um pacote, atulizar, ou mudar a sua versão, e precisa estar acompanhado na linha de comando o nome do pacote a ser modificado.

3. (difícil) O comando **sed** é bastante utilizado na programação e uma ferramenta muito poderosa, para filtrar e transformar textos. Utilize o comando man sed para se familiarizar, em seguida utilize o comando sed três vezes utilizando ou a flag 'a', 'i' ou 'c'.

Resposta:

1. sed 'a\' arquivoA arquivoB > arquivoC
2. sed -i 'i insere texto no final do arquivo' arquivoA
3. sed -i '/string/ c\substitua a linha ' arquivoB

4 Personalizando seu Ambiente

Da mesma forma que pode arranjar sua mesa de trabalho de uma forma única, o bash também possui formas de personalizar e organizar seu ambiente. Através dos arquivos .bash_profile, .bash_logout e .bashrc, os arquivos mencionados se encontra em sua home , para visualiza-los utiliza **$ ls -la**.

Estes arquivos tem um sentido especial para o bash, o .bash_profile é lido ao inicializar a sessão, .bash_logout é lido no final e .bashrc é lido e interpretado para funções, alias e algumas configurações adicionais.

4.1 Resumo

As ferramentas que permitem editar seu ambiente de trabalho, agilizando e facilitando o dia-a-dia do usuário, são as variáveis de ambiente, alias e opções, que serão apresentas nos capítulos seguintes.

4.2 Aliases

O alias é uma forma de criar um atalho a partir de qualquer comando ou conjunto de comandos, deve ser adicionado no arquivo .bashrc e o formato a ser utilizado é alias nome=“comando”, é necessário dar atenção a não existência do espaçamento antes e após o sinal de igual, para a implementação do comando.

Exemplos:

alias work="cd /git/UFPR/BASH", ao utilizar **$ work**, o bash interpretará o comando adicionado através do alias, me redirecionado ao diretório especificado pelo comando.

alias ls="ls -a –color=auto", ao utilizar ls irá mostrar como default os arquivos ocultos e ira diferenciar os tipos de arquivo com cores.

alias cd='cd ' e alias work=" /git/UFPR/BASH", o comando **$ cd** work, ira interpretar o alias work como diretório, este truque só é possível por declarar o primeiro alias.

Observe que a alteração não é aplicada após salvar o arquivo é necessário usar o comando **$ source** ~/.bashrc, source é uma ferramenta do shell que executa o conteúdo de um arquivo passado por argumento para o shell que esta sendo usado atualmente.

4.3 Opções

As opções podem ser usadas para modificar o comportamento do shell. As opções pode estar no estado ‘on’ ou ‘off’, o comando para mudar de estado é **$ set** -o nomeopção para ligar ‘on’ e **$ set** +o para desligar ‘off’.

- **$ set** -o vi Utiliza o editor vi no shell atual;

- **$ set** -o noaliases Desabilita os alias no shell atual;
- **$ set** -o errexit Aborta a execução do script caso aconteça um erro;
- **$ set** -o verbose Apresenta no stdout o comando ao ser utilizado;

Para uma lista completa de todas as opções contidas no seu shell use o comando **$ set** -o.

4.4 Variáveis shell

Variáveis de ambiente da mesma forma que o alias é um nome que tem um valor associado a ele. A sintaxe de declaração é **$** teste=argumentodeteste, e para acessar a variável utilizada como exemplo ela precisa esta acompanhada do simbolo '**$**'.

$ echo "$teste"

O efeito do comando acima é 'argumentodeteste', se a variável não tiver sido declarada será um campo vazio. Segundo a declaração não pode existir espaçamento.

No exemplo do comando echo acima foi usado aspas duplas, que interpretam a variável, caso opte por usar somente a palavra deve-se usar aspas simples, e em caso de dúvida utilize sempre aspas simples.

As variavéis shell são muito utilizadas na programação bash e também pode ser usada para configurações, exemplo de ambos serão apresentados nos próximos capítulos.

4.5 Histórico de comandos

As variáveis abaixo são de grande utilidade para editar como armazenar os comandos history, e podem ser definidas em .bashrc.

HISTSIZE - define o numero de comandos salvo no arquivo HISTFILE, e HISTFILESIZE o numero máximo de linhas que o arquivo HISTFILE.

4.6 Prompt de comando

Bash usa prompt de strings, sendo armazenadas em PS1 e PS2, a maioria das pessoas gostam de setar o prompt com seu login, da seguinte forma **PS1**="\u->". Outra opção interessante é apresentar o diretório através do prompt **PS1**="\w->".

A variável PS2 por padrão é '>', é usada após utilizar um comando incompleto ou após usar barra invertida e pressionar enter.

Algums exemplos que podem ser utilizados são:

- \A horário atual;
- data no formato "dia da semana mês dia";
- \s nome do shell;
- \v versão do bash;

(NEWHAN, 2005)

4.7 Conclusão

As maneiras apresentadas de editar o bash, proporciona uma forma de agilizar e oferece ao usuário liberdade para construir ferramentas que possam facilitar e automatizar tarefas repetitivas. Além de poder personalizar e deixar o bash com seu estilo seja para programar ou como usuário.

4.8 Questões

1.(fácil) Acesse .bashrc e crie um alias.

Resposta: Livre

2.(médio) Acesse o arquivo contido na variável HISTFILE, análise o comando que possui mais ocorrências e crie um alias. Encontre padrão entre 2 ou mais linhas seguidas de comandos e crie outro alias para este padrão.

Resposta: Ao trabalhar quando logo na máquina posso ficar dando cd entre os diretórios para meu local de trabalho ou criar alias direto para ela $ alias work="cd /trabalho/git/c3sl/repo/blendb".

3.(difícil) Na sessão opções foram apresentados alguns exemplos de atributos que podem modificar o comportamento do bash, o exercício é simples porém trabalhoso, crie uma lista com o resultado dos comandos **$ set** -o &&

shopt -o, e execute pelo menos um por dia.

Resposta: O exercício demonstra uma forma de aprender sem sobrecarregar o aluno, e utilizando na pratica para sentir a utilidade da ferramenta.

5 Processos

Uma das ferramentas que ajudou a construir a reputação do sistema operacional UNIX foi o controle que o usuário possui sobre *multitasking*, nesta seção serão apresentados as ferramentas disponibilizadas pelo bash para trabalhar com *multitasking*.

5.1 Introdução

Será apresentado como controlar os processos do linux. Exibindo ferramentas disponibilizada pelo linux como: job control, foreground e background ,signals, kill, ps, wait e subshell.

E também neste tópico será apresentado o CTRL+alt+del(Windows) , na versão linux porém muito mais potente e com muita mais opções de controle, como o comando ps.

5.2 Controle de Processos no Linux

O sistema Unix atribui a cada processo números , esse chamados IDs, e caso o programa esteja sendo executado com & será atribuído job number.

Executando o comando : "**evince** example̊pdf & [1] 2330"

No exemplo acima [1] é o número do job,2330 é o ID do processo.

Job refere-se ao programa rodando em background, sobre seu shell, enquanto o ID do processo se refere a todos os processos rodados por todo o sistema.

Se você executar outros comandos como :

evince pdf1 & [2] 2479

evince pdf2 & [3] 2496

E após a execução do programa em background o shell retorna caso completo:

[2] + 2479 done

Caso tenha ocorrido algum erro:
[3] + 2479 Exit 1

5.3 Job Control

O modo mais simples de controlar um job é criar um e coloca-lo em modo background com &.

5.3.1 Foreground and Background

Ao entrar com o comando **fg**, o job que foi utilizado como & ira tomar conta do seu terminal.

Se você só possui um único job rodando em background, pode-se usar fg sem argumentos e o shell irá trazer esse job para "foreground".

Caso não se lembre de todos os comandos que estão rodando em *background* use o comando "**jobs**".

As flags que jobs aceita são "-p" que ira listar somente os job number, "-l" lista os processos id além dos dados jobs.

A flag -n lista as opções na qual os status do job mudou desde sua última verificação através do comando jobs.

A flag -s lista os jobs que estão parados esperando por input do teclado.

Para trazer o processo para foreground, pode-se utilizar o comando fg, se utilizado sem argumento o último ou mais recente processo será levado para "foreground", no entanto fg %1, irá trazer o primeiro processo para frente.

Se mais que um background job possui o mesmo comando então %, irá escolher entre o mais recente. Se caso isso não é o que você deseja , você precisará usar o número do job.

Lista com todos os jeitos de se referir a um job:

- %N - Numero do job 'N'
- %string : Job na qual o comando começa com a string
- %?string : Job na qual o comando contem string
- %+ : O job invocado mais recente
- %- : O segundo job invocado mais recente

5.4 Suspendendo um JOB

Da mesma forma que se pode colocar um job de foreground to background existe uma forma de fazer o inverso, através do comando CTRL+Z, enquanto ele estiver sendo executado em foreground o programa será suspenso, mas se utilizar bg ele irá se mover para background.

Uma forma de se visualizar essa transição é através do editor vim, ou vi.
vim test.c editar o arquivo
Salvar a edição
Sai do modo inserção
Use o comando CTRL+Z
Execute gcc ou debug
fg %vim irá devolver o comando para executar o editor de texto novamente.

5.5 Signals

Um sinal é uma mensagem que um processo enviar para outro quando algum evento anormal acontece ou requer que outro processo faça algo.

Dependendo cada versão do unix existem sinais diferentes.

Através do comando "**kill -l**" pode-se obter uma lista de todos os sinais, que o seu unix possui.

Existem atalhos que estão relacionados com esses sinais como:

- CTRL-C voce requista o shell enviar um INT que significa interromper o job atual.
- CTRL-Z envía TSTP que significa "parar o terminal"
- CTRL- envia o sinal QUIT que é uma versão mais forte do CTRL-C

Uma observação sobre o atalho "CTRL-\", use se somente se CTRL-C não funcionar, ele é um caso que não lida com as devidas tarefas que o processo executa quando está se encerrando.

Além desse atalhos pode-se criar novos ou editar atalhos, porém não é recomendado trocar os sinais caso você compartilhe sua máquina com outro usuário.

5.6 Kill

Para terminar o processo pode-se enviar kill para qualquer processo você ou seu sistema operacional criou, kill recebe como parâmetro o id do processo.

Por padrão kill envia TERM "terminate" que tem o mesmo efeito que CTRL-C. Os sinais TERM e QUIT que são a forma mais comum de se terminar o programa, pois dá a chance do programa executar algumas tarefas caso eles tenham necessárias para sua estrutura. O comando kill -KILL deve ser usado em último caso, pois termina o processo e ignora qualquer tipo de handler que o processo precisa executar para finalizar o programa com sucesso.

5.7 PS

O comando ps lhe dá a informação do ID de um processo, através do índice PID. Além desta informação ele devolve o tempo em que o processo foi ou está sendo utilizado, o índice CMD é o comando.

O comando ps sem flags, irá retornar todos os processos que se iniciaram a partir do terminal onde foi executado, implicando que não irá encontrar processos de outras shell. Ele não consegue relacionar o ID com o número do job.

Cada sistema terá um comportamento diferente para a execução do comando ps.

O comando ps também pode ser visualizado através da man page.

5.8 wait

Um comando interessante é o wait ele espera a execução completa de um processo, caso o usuário coloque seu ID.

Se nenhum ID for determinado ele irá esperar a execução completa de todos os processos.

5.9 SubShell

O subshell é utilizado quando se é executado um script, ou quando se inicia um shell a partir de outro shell.

A característica mais importante que se precisa conhecer de subshell é que eles herdam as características de seu parente.

- o diretório atual
- as variáveis de ambiente
- os padrões de entrada, saída ,erro e outros descritores de arquivo.
- Sinais que são ignorados

Da mesma forma que as coisas que o subshell não herda:

- Variaveis shell
- handlers de sinais que são ignorados

5.10 Questões

1.(fácil) Cite quais são as diferenças entre ctrl+alt+del do windows em relação ao ps -aux?

R:. O comando do windows é bastante limitado em relação ao ps -aux , o primeiro lista somente os programas em execução referente ao usuário , o segundo lista qual usuário/root inicializou o tempo de execução PID, e o comando que se inicializou.

2.(médio) Execute 5 programas com a mesmo executável e coloque-os em background com &. Traga para foreground um a um e devolva-os para background a sequência deve ser do menor para o maior.

R:. Será utilizado a ferramenta evince para carregar 5 pdfs.
evince pdf0 &
evince pdf1 &
evince pdf2 &
evince pdf3 &
evince pdf4 &
fg %1 , CTRL+z , **bg**
fg %2 , CTRL+z , **bg**
fg %3 , CTRL+z , **bg**
fg %4 , CTRL+z , **bg**
fg %5 , CTRL+z , **bg**

3.(difícil) Leia as man page de todos os comandos que foram apresentados nesta seção, se existe algum comando novo que você sempre procurou execute com as primeiras 5 flags que a man page apresentar.

R:. Livre.

5.11 Conclusão

Conforme apresentado nos capítulos anteriores, existem vários níveis para controlar o processo desde foreground e background que são os mais simples para utilizar diariamente, e o kill que é mais específico e utilizado raramente pelo usuário. No entanto é possível criar scripts para automatizar vários processos, com as ferramentas apresentadas.

6 Hackerman não usa mouse

Para o usuário ser considerado programador avançado, independente da linguagem, somente após abandonar totalmente o uso do mouse. Consequentemente irá aumentar a velocidade na digitação e sua performance na programação.

6.1 Lidando com seu terminal

Nesta seção serão apresentados os comandos e atalhos mais úteis para o uso do seu terminal.

Relacionados a tela:

- Ctrl+alt+t: Abrir terminal
- Alt+f2: Digite “gnome-terminal” ou “firefox”
- Ctrl+Shift+t: Abrir nova aba terminal
- Alt+1,Alt+2,Alt+3 ... : Navegar entre as abas abertas do terminal
- Alt+Tab: Para transitar entre as telas

- Ctrl+Alt+‘ “Seta direita” ou “Seta esquerda”: Transitar entre as estações de trabalho
- Ctrl+Alt+‘Seta cima’: Mostra as estações de trabalho
- Ctrl+Alt+‘Seta baixo’: Mostra os aplicativos/programa em execução na estação de trabalho atual
- Ctrl+‘-’: Diminuir a fonte
- Ctrl+‘+’: Almentar a fonte
- Ctrl+‘l’: Limpa a tela, diferente do comando ‘clear’ que remove todos os dados da tela.
- Ctrl+Shift+Alt+‘: Seta direita ou seta esquerda’ transita a tela entre os workspace

Relacionados a Programas em execução:

- Ctrl+c: Interrompe o programa atual
- ‘Ctrl+d’ ou ‘exit’: Encerra a sessão bash atual
- Ctrl+z: Encerra o job

Um dos atalhos mais interessantes do linux é o Alt+f2, que lhe oferece acesso a uma “linha de comando”, em que o usuário pode executar qualquer comando, deste que este esteja instalado ou com link simbólico setado.

O comando “resize -s <linhas> <colunas>”,pode ser usado para redefinir o tamanho do seu terminal.

Além do viés deste trabalho de remover a necessidade do uso de mouse, existem alguns terminais que também são voltados a programação, e bastante ricos em atalhos como:

- terminator
- screen

A documentação destes terminais podem ser encontrada através do seu respectivo man page. Uma vantagem visual interessante dentre os três apresentados é a facilidade de gerar multiterminais e evitar de ficar usando o mouse para determinar onde o terminal atual ira se situar.

6.2 Digitação

A transição do uso do mouse para atalhos do SO ou de algum programa, pode de início ser difícil no entanto recompensadora no futuro. Com o intuito de ajudar na transição o usuário pode dedicar um tempo a aprimorar sua velocidade na digitação, com o programa Klavaro, que é um curso de datilografia, e é software livre. Para obter mais informações sobre o software visite http://klavaro.sourceforge.net/pt/index.html.

7 Conclusão

Com a base que foi apresentada nos capítulos anteriores o leitor pode se desenvolver com a utilização dos comandos no dia-a-dia, procurando combinar comandos e utilizando sempre o que você aprendeu evitando usar as built-in das ide a fim de aprofundar o conhecimento do Bash. O usuário agora com o conhecimento da man page, pode tirar dúvidas e conhecer qualquer ferramenta que o linux apresenta através do bash. Caso alguma parte que foi escrito tenha ficado ambígua ou o leitor possua alguma dúvida sobre os códigos que serão apresentados nos trabalhos práticos envie e-mail para rpd17@inf.ufpr.br.

8 Trabalhos Práticos

As próximas sessões contém atividades resolvidas durante o semestre da matéria Técnicas alternativas de programação CI320

8.1 Primeira Atividade

Consiste em resolver três tarefas propostas em sala de aula, buscando testar o conhecimento do aluno, adquirido durante a disciplina.

Parte 1: eliminar automaticamente as linhas do arquivo de entrada (patrimonio.csv) que contenham o símbolo do ponto e vírgula (;) que estão entre aspas, como no exemplo abaixo.

"460819";"VENTILADOR; DE TETO (5234)";"127V ";;"2100.13.02.21";

A primeira iniciativa foi entender o problema e quais as ferramentas para resolver o problema. As ferramentas utilizadas foram **awk**, **grep** e **sed**.

Como awk contém ferramentas mais avançadas do que os demais, ela foi utilizada para resolver a parte 1.

O script abaixo retorna todas as linhas que possuem ';' e estão entre texto porém não está entre aspas seja pela esquerda, direita ou ambos.

$ awk '/([\^"];[\^"])/ {print}' patrimonio.csv

Entretanto como a atividade é remover as linhas, resolve-se invertendo a pesquisa com '!', e adiciona o resultado no arquivo 'novoPatrimonio.csv'.

$ awk '!/([\^"];[\^"])/ {print}' patrimonio.csv > novoPatrimonio.csv

Parte 2: obter em um arquivo separado a lista de locais (coluna 5 do arquivo) de forma única e ordenada.

O primeiro pipe, utiliza -F 'field separator' para filtrar o campo especificado do arquivo novoPatrimonio.csv, o segundo pipe elimina as aspas, o terceiro pipe ordena, e o quarto pipe elimina os repetidos (uniq) e adiciona os dados no arquivo data.

$ awk -F\; '{print $5}' novoPatrimonio.csv | sed 's/"//g' | sort | uniq > datas

Observação: É possível pular a primeira parte, utilizando o separador composto disponibilizado por awk 'FS = "\";"\"'.

$ awk 'BEGIN {FS = "\";"\"'}; {print $5}' novoPatrimonio.csv | sed 's/";//g' | sort -u | uniq > datas

Parte 3: para cada código de local da lista de locais obtida na parte 2 (exemplo de local: 2100.13.01.02), criar um arquivo cujo nome seja o código com a extensão .csv (exemplo de arquivo: 2100.13.01.02.csv) que contenha todas, e apenas, as linhas do arquivo de entrada que contenham este padrão. (exemplo de conteúdo para o arquivo 2100.13.01.02.csv, ele contém todas as linhas que contém a string "2100.13.01.02").

O comando 'for' foi utilizado para resolver a terceira tarefa. A primeira linha do código lê linha a linha do arquivo datas, a segunda linha do script procura o padrão e cria um arquivo no diretório com os dados encontrados pelo padrão.

```
for i in $(cat datas); do
grep "$i"novoPatrimonio.csv > diretorio/$i.csv;
done
```

8.2 Segunda Atividade

O objetivo deste trabalho é contabilizar o número de matrículas em cada disciplina do Departamento de Informática entre os anos de 1988 (primeiro

semestre) até 2002 (segundo semestre).

Primeira atividade: deve-se construir um script que dê como saída um arquivo que contém o total de matrículas semestre a semestre considerando que os GRR's não se repetem. O objetivo é ver a evolução do número de matrículas semestre a semestre no período em análise. A notação usa a concatenação do ano com o semestre, por exemplo, o primeiro semestre de 1988 é denotado 19881.

Caso o leitor não tenha acesso aos arquivos segue o exemplo da formatação:

Diretório/Turmas/dados:cabeçalho
DadosMatricula/CI057/19992.dados:curso:grr
Diretório/Turmas/dados:informações
DadosMatricula/CI061/20002.dados:21:199812652

O desafio de resolver a atividade foi encontrar os padrões e como filtrar os dados, após entender os padrões, foi necessário combinar as ferramentas aprendidas durante o curso.

Para realizar esta atividade foram usadas as ferramentas **grep**, **cut**, **echo**, e variáveis. As flags serão explicadas para compreender o código.

grep '-R' executa recursivamente, '-h' elimina os campos de arquivo e '-v' resultado invertido.

cut '-f' representa o field ou campo a ser selecionado, -d o delimitador.

sort '-u' para serem campos sem repetição.

wc '-l' representa o numero de '\n', do arquivo.

for i in array executa uma operação sobre cada elemento da array, após concluir esta operação, o próximo elemento sera selecionado, até que seja executado sobre o ultimo elemento do vetor.

echo '>>saída' concateno o resultado no arquivo saída.

Resolução da primeira atividade:

```
> atividade1
for i in $(grep -R ":" DadosMatricula | cut -d/ -f3 |
cut -f1 -d. | sort -u);
      do
      sum=0
      var=$(grep -R ":" DadosMatricula/*/$i.dados -h |
      grep -v curso | sort -u | wc -l)
      echo "$i : $var" >> atividade1
```

done

Segunda atividade: o objetivo é acompanhar a evolução do número de matrículas em cada disciplina semestre a semestre ao longo do período analisado, por isso neste caso os GRR's podem se repetir. Um mesmo GRR cursa normalmente várias disciplinas. A saída pode ser formatada de maneira que cada linha inicie com o ano/semestre (ex. 19881) e que tem em suas colunas o total de matrículas para cada curso. Os cursos que não tem matrículas em uma disciplina não devem aparecer neste relatório.

Como a segunda atividade é relacionada a primeira, já se tem a ideia do padrão dos dados e como se resolver o exercício, no entanto será cobrado a concatenação e edição da saída, usando a função printf ajuda na resolução e o comando 'echo -n', ajuda a elaborar a matriz.

As flags que foram usada no algoritimo abaixo e que não foram apresentadas na resolução da primeira atividade serão apresentados.

echo '-n' imprime sem pular a linha ou seja sem usar '\n'.

if e else possui uma sintaxe diferente da linguagem de programa c, dentro do colchetes precisa ter um espaçamento entre as variáveis e o colchetes, '-le' compara se o valor é menor ou igual.

grep '-c' quantidade de padrões encontrados.

printf foi usado pois é mais poderoso para edição do que o echo, "%7d" representa um inteiro e contido em um espaçamento de sete casas decimais.

Resolução da segunda atividade:

```
> atividade2
cont=0;
tam=0;
echo -n "       " >> atividade2
for j in $(grep ":" -R DadosMatricula | cut -f2 -d/
| sort -u);
      do
      printf "%7s" $j >> atividade2
done
echo  >> atividade2
# i => anos
# j => cursos
for i in $(grep -R ":" DadosMatricula | cut -f3 -d/
| cut -f1 -d. | sort -u); do
```

```
        echo -n "$i" >> atividade2
        for j in $(grep -R ":" DadosMatricula/*/$i.dados |
                cut -f2 -d/ | sort -u); do
                for k in $(grep -R ":" -c DadosMatricula/$j/$i.dados); do
                        # remove o cabeçalho
                        ((k--))
                        if [ $k -le 0 ]; then
                        echo -n "       " >> atividade2
                        else
                        printf "%7d" $k  >> atividade2
                        fi
                done
        done
        echo " " >> atividade2
done
```

Os códigos foram apresentados pois sua visualização e exemplos ajudam no aprendizado, no entanto a leitura da man page é essencial.

8.3 Terceira Atividade

Para resolver a atividade em um tempo que fosse menor que o script utilizado no dinf (3 segundos), a ferramenta utilizada foi **awk** e hash.

O comando '{a[$6]++ }', cria uma tabela hash da sexta coluna do arquivo de entrada no awk, e os índices serão os dados a ser filtrados, e os dados da tabela serão a quantidade de vezes que o índice apareceu. Também no awk foi utilizado " PROCINFO["sorted_in"] = "@ind_str_desc" ", ele define a ordenação do vetor e "@ind_str_desc", seleciona o índice interpretado como sring descendente.

O comando " if [$# -ne $EXPECTARGS] ", testa para verificar se esta sendo utilizado como entrada o arquivo log, $# é a quantidade de argumentos sendo $0 o nome do programa, e EXPECTARGS é uma variável definida como 1, pois o script depende da entrada $1 que será o arquivo de log, caso teste seja inválido, será enviado uma mensagem ao usuário de como se deve executa-lo, o programa será finalizado com exit 1.

O comando " if ! [-d $DIR] ", se o diretório não existir faça algo. Os testes são executados a partir de um diretório relativo, por exemplo se o teste for falso, pode-se usar o diretório atual através de DIR=$PWD.

Para enviar e-mail, foi utilizado a ferramenta mutt, a flag "-a" anexa o arquivo, "-s" o assunto e o e-mail é adicionado após "—".

Referências

HAHN, H. *Students Guide to Unix*. [S.l.: s.n.], 1993. páginas 4

NEWHAN, C. *Learning the bash Shell*. [S.l.: s.n.], 2005. páginas 8, 12