CURSO DE INTRODUÇÃO AO LINUX

Distribuição openSUSE®

AULA 4 – Operações com arquivos

PET ENGENHARIA DE COMPUTAÇÃO

Objetivos dessa aula

Aprender a:

- Criar, gerenciar e procurar arquivos através do terminal;
- Comparar arquivos e identificar tipos diferentes;
- Comprimir dados e realizar backups.

Procurando e trabalhando arquivos

Trabalhando arquivos

O Linux providencia vários comandos que te ajudam a ver o conteúdo de um arquivo, criar um novo arquivo (vazio ou não), mudar a assinatura de tempo de um arquivo, e remover ou renomear um arquivo ou diretório.

Esses comandos te ajudam a gerenciar seus dados e arquivos, e a garantir que os dados corretos estejam na localização correta.

Procurando e trabalhando arquivos

Visualizando arquivos

As seguintes ferramentas te ajudam a visualizar arquivos:

- **cat** e **tac**;
- **less**;
- **head** e **tail**.

Procurando e trabalhando arquivos

touch

O **touch** é muito usado para definir e atualizar o acesso, mudanças, e alterar tempos em arquivos. Por padrão, ele reseta a assinatura de tempo de um arquivo para coincidir com a hora atual.

É possível, porém, criar um arquivo vazio usando o touch, para ser trabalhado depois:

- $ touch <nome>

O touch possui várias opções, mas de interesse é a opção **-t**, que permite alterar a assinatura de tempo:

- $ touch -t 10201640 teste
  - Isso mudará a assinatura de tempo do arquivo teste para 20/10, 16h40.

Procurando e trabalhando arquivos

mkdir e rmdir

O **mkdir** é usado para criar diretórios, usando a sintaxe **mkdir <caminho>**. Esse caminho pode ser relativo ou absoluto.

Para remover um diretório, basta usar o comando **rmdir**, porém ele deve estar vazio ou o comando falhará. É possível remover um diretório cheio com todo seu conteúdo usando **rm -rf**, mas pode ser arriscado.

Procurando e trabalhando arquivos

Removendo arquivos e diretórios

- **mv** : Renomear arquivo ou diretório
- **rm** : remover um arquivo
- **rm - f** : remover forçadamente um arquivo
- **rm - i** : remover interativamente um arquivo
- **rmdir** : remover um diretório vazio
- **rm -rf** : remover forçadamente um diretório recursivamente

Procurando e trabalhando arquivos

Try-it-yourself: Criar e remover

- Criar dois arquivos vazios **teste1** e **teste2** usando o comando **touch** com a assinatura de tempo **14 de março de 2018 14h00**;
- Checar a existência dos arquivos **teste1** e **teste2** usando o comando **ls -l teste1 teste2**;
- Renomear o arquivo **teste1** para **teste1_novo** usando o comando **mv**;
- Remover o arquivo **teste2** usando o comando **rm** sem nenhuma opção;
- Remover o arquivo **teste1_novo** usando o comando **rm** sem nenhuma opção;
- Criar um diretório **dir3** usando o comando **mkdir**;
- Remover o diretório **dir3** usando o comando **rmdir** sem opções.

Procurando e trabalhando arquivos

Correntes de Arquivo Padrões

Quando comandos são executados, por padrão há três correntes de arquivo padrões, sempre abertas para uso: **entrada padrão** (stdin), **saída padrão** (stdout) e **erro padrão** (stderr).

Geralmente, stdin é seu teclado, stdout e stderr são exibidos em seu terminal, sendo stderr tipicamente direcionado a um arquivo de log de erros.

Stdin pode ser suprido direcionando a entrada a partir de um arquivo ou da saída de um comando anterior através de um **pipe**. Stdout também pode ser redirecionado a um arquivo.

Procurando e trabalhando arquivos

Correntes de Arquivo Padrões

No Linux, todo arquivo aberto é representado internamente por **descritores de arquivo**. De modo simples, eles são representados por números começando em zero.

Stdin é o descritor 0, stdout o descritor 1 e stderr o descritor 2. Tipicamente, se outros arquivos forem abertos além desses três (que são abertos por padrão), começarão no descritor 3 e incrementarão a partir daí.

Procurando e trabalhando arquivos

Redirecionamento de entrada e saída

Através do shell de comando podemos redirecionar as três correntes padrão de modo a conseguir entradas a partir de um arquivo ou outro comando ao invés do teclado, e podemos escrever resultados de saída e erro para arquivos ou enviá-los como entrada para novos comandos.

Por exemplo, se tivermos um programa chamado **faca_algo** que leia de stdin e escreva em stdout e stderr, podemos mudar sua fonte de entrada usando o sinal menor-que (<) seguido do nome do arquivo a ser usado como entrada:

- $ faca_algo < arquivo_in

E para redirecionar a saída para um arquivo, é usado o sinal maior-que (>):

- $ faca_algo > arquivo_out

Procurando e trabalhando arquivos

Redirecionamento de entrada e saída

Como stderr **não é** a mesma coisa que stdout, mensagens de erro ainda serão vistas nos terminais nos exemplos anteriores.

Caso queira redirecionar stderr para um arquivo à parte, é possível usar o número do descritor de stderr, junto do sinal maior-que, seguido do nome do arquivo a guardar a saída de stderr:

- $ faca_algo 2> arquivo_err

Ainda é possível juntar a saída do descritor 2 (stderr) à do descritor 1 (stdout) usando uma notação especial:

- $ faca_algo > arquivo_out_all 2>&1

Procurando e trabalhando arquivos

Pipes

A filosofia UNIX/Linux é ter vários programas (ou comandos) curtos e simples cooperando juntos para produzir resultados complexos, ao invés de ter um programa complexo com muitas opções e modos de operação possíveis.

Para que isso seja possível, o uso extensivo de **pipes** é feito: você pode "canalizar" a saída de um comando como a entrada de outro. Para fazer isso, usamos a barra vertical (|) entre comandos:

- $ comando1 | comando2 | comando3

Procurando e trabalhando arquivos

Pipes

A solução anterior representa o que chamamos de **pipeline** e permite que o Linux combine as ações de vários comandos em um só.

Isso é muito eficiente porque **comando2** e **comando3** não precisam esperar que os comandos anteriores terminem antes de começarem a trabalharem seus dados em suas correntes de entrada. Adicionalmente, não é necessário criar arquivos temporários entre os estágios do pipeline.

Procurando e trabalhando arquivos

Procurando arquivos - locate

A ferramenta **locate** executa uma busca através de uma database previamente construída de arquivos e diretórios do seu sistema, correspondendo todas as inscrições que contenham uma string específica, o que às vezes resulta numa lista muito longa.

Para obter uma lista mais curta e relevante, podemos usar o comando **grep** como um filtro: o grep só imprime as linhas que contêm uma ou mais strings como em:

- $ locate zip | grep bin

Procurando e trabalhando arquivos

Procurando arquivos - locate

- $ locate zip | grep bin

  Esse comando vai listar todos os arquivos e diretórios que contenham ambos **zip** e **bin** em seus nomes.

  O **locate** usa a database criada por outro programa, **updatedb**. A maioria dos sistemas Linux o executam diariamente, mas é possível atualizar manualmente executando **updatedb** na linha de comando como superusuário.

Procurando e trabalhando arquivos

Procurando arquivos - Coringas

É possível buscar um arquivo contendo caracteres específicos usando **caracteres coringas**:

- **?** : Corresponde a qualquer caractere;
- * : Corresponde a qualquer string;
- **[conjunto]** : Corresponde a qualquer caractere no conjunto, e.g. [asd] corresponderá qualquer ocorrência de “a“, “s“ e “d“;
- **[!conjunto]** : Corresponde a qualquer caractere que não esteja no conjunto.

Procurando e trabalhando arquivos

Procurando arquivos - Coringas

Para buscar arquivos usando o coringa **?**, troque cada caractere desconhecido por ?, e.g. caso você saiba apenas que as duas primeiras letras de um nome de três caracteres são “ba“ com uma extensão “.out“, digite $ ls ba?.out

- Para buscar arquivos usando o coringa *, substitua a string desconhecida por *, e.g. caso você se lembre apenas que a extensão é “.out“, digite $ ls *.out

Procurando e trabalhando arquivos

Procurando arquivos em um diretório

A ferramenta **find** é extremamente útil e muito utilizada no dia-a-dia de administradores Linux. Ela recorre abaixo a árvore de arquivos a partir de qualquer diretório específico (ou conjunto de diretórios) e localiza arquivos que correspondam às condições especificadas. O caminho padrão é sempre o diretório atual.

Por exemplo, adminstradores de vez em quando buscam por grandes **arquivos núcleo** (que possuem informação de diagnóstico após falhas) mais de semanas de idade para removê-los. Também é comum remover arquivos em **/tmp** que não foram acessados recentemente. Várias distribuições usam scripts automatizados que periodicamente executam tal limpeza.

Procurando e trabalhando arquivos

Procurando arquivos - find

Quando não são dados argumentos, **find** listará todos os arquivos no diretório atual e em todos seus subdiretórios. Opções comumente utilizadas para encurtar a lista são:

- **-name** : Listar apenas arquivos com um determinado padrão no nome;
- **-iname** : Não distinguir entre maiúsculas e minúsculas na busca por arquivos;
- **-type** : Restringir a busca a arquivos de um tipo específico, como **d** para diretórios, **l** para links simbólicos ou **f** para arquivos regulares, etc.

Procurando e trabalhando arquivos

Procurando arquivos - find

Buscando por arquivos e diretórios chamados **gcc**:

- $ find /usr -name gcc

Buscando apenas diretórios chamados **gcc**:

- $ find /usr -type d -name gcc

Buscando apenas arquivos regulares chamados **testel**:

- $ find /usr -type f -name testel

Procurando e trabalhando arquivos

Comparando arquivos

O programa **diff** é usado para comparar arquivos e diretórios. Esse utilitário tão usado tem muitas opções úteis (ver **man diff**) inclusive:

- **-c** : Providencia uma lista de diferenças que inclui 3 linhas de contexto antes e depois das linhas que diferem;
- **-r** : Comparar recursivamente subdiretórios, além do diretório atual;
- **-i** : Ignorar a distinção maiúscula/minúscula
- **-w** : Ignorar diferenças em espaços e tabulações

Para comparar dois arquivos/diretórios no terminal, basta digitar

- $ diff <nome1> <nome2>

Procurando e trabalhando arquivos

Comparando arquivos

Você pode comparar três arquivos de uma vez usando **diff3**, que usa um arquivo como referência para outros dois. O diff3 mostrará as diferenças dos dois arquivos comparados ao primeiro, e a sintaxe de utilização é a seguinte:

- $ diff3 arquivo1 arquivo_base arquivo2

Procurando e trabalhando arquivos

Aplicando correções (patches)

Muitas modificações a códigos fonte e arquivos de configuração são distribuidos usando patches que são aplicados com o programa **patch**. Um arquivo de correção contém os **deltas** (mudanças) necessários para atualizar um arquivo antigo para sua nova versão.

Os arquivos de correção são produzidos executando o diff com as opções corretas, como em:

- $ diff -Nur arquivo_original arquivo_novo > arquivo_patch

Distribuir apenas a correção é mais conciso e eficiente que distribuir o arquivo todo, pois possui apenas algumas linhas, comparado ao tamanho total do arquivo. Para aplicar uma correção, segue a sintaxe:

- $ patch -pl < arquivo_patch (corrige um diretório inteiro)
- $ patch arquivo_original arquivo_patch (corrige apenas um arquivo)

Procurando e trabalhando arquivos

Utilitário file

No Linux, a extensão de um arquivo nem sempre o categoriza como pode ocorrer em outros sistemas. Não é possível afirmar que um arquivo chamado **arquivo.txt** é um arquivo de texto e não um executável.

No Linux um nome de arquivo faz mais sentido para o usuário do que para o próprio sistema, aliás a maioria das aplicações examinam diretamente o conteúdo de um arquivo para determinar seu tipo ao invés de confiar na extensão.

A natureza de um arquivo pode ser averiguada com o utilitário **file**. Para os nomes dados como argumentos, ele verificará o conteúdo e certas características para determinar os arquivos são puro texto, bibliotecas compartilhadas, executáveis, scripts ou outra coisa.

Procurando e trabalhando arquivos

Realizando backup

Há várias maneiras de fazer backup de dados ou mesmo do sistema inteiro. As maneiras mais básicas incluem o uso de cópia com **cp** ou usar o **rsync**.

Ambos podem ser usados para sincronizar árvores inteiras de diretórios, porém, o **rsync** é mais eficiente pois verifica se o arquivo a ser copiado já existe e não possui diferença no tamanho ou horário de modificação, poupando tempo ao evitar uma cópia desnecessária.

E mais, como o rsync copia apenas as partes dos arquivos que possuem alterações, pode ser muito rápido.

Procurando e trabalhando arquivos

Realizando backup

O **cp** pode apenas copiar arquivos de e para destinos na máquina local, porém o **rsync** pode também ser usado para copiar arquivos de uma máquina em rede para outra.

O **rsync** é muito eficiente ao copiar recursivamente um diretório para outro, pois só as diferenças são transmitidas.

Procurando e trabalhando arquivos

Utilizando o rsync

Uma maneira útil de fazer o backup de um diretório de um projeto é através do seguinte comando:

- $ rsync -r projeto -archive maq2:arquivos/projeto

Note que o uso do rsync pode ser destrutivo! O uso da opção **--dry-run** é aconselhado para testar os resultados antes da execução.

Para usar o rsync no terminal, digite

- $ rsync arquivo_fonte arquivo_destino

Procurando e trabalhando arquivos

Comprimindo dados

Dados de arquivos são comumente comprimidos para poupar espaço em disco e reduzir o tempo necessário para transmiti-los em rede. O Linux usa alguns métodos de compressão, como:

- **gzip**
- **bzip2**
- **xz**
- **zip**

Adicionalmente, o utilitário **tar** pode ser usado para agrupar arquivos para então serem compactados.

PET ECO
ENGENHARIA DE COMPUTAÇÃO

Procurando e trabalhando arquivos

Comprimindo dados - gzip

O **gzip** é o utilitário de compressão mais usado no Linux, pois o faz bem e rápido. A seguir, alguns exemplos de uso:

- **$ gzip *** : Compacta todos os arquivos no diretório atual, cada arquivo é comprimido e renomeado com a extensão .gz;
- **$ gzip -r projeto** : Compacta todos os arquivos no diretório **projeto** junto de todos os arquivos em seus subdiretórios;
- **$ gunzip foo** : Descompacta o arquivo/diretório **foo** encontrado no arquivo **foo.gz**.

Procurando e trabalhando arquivos

Comprimindo dados - bzip2

O **bzip2** possui sintaxe similar ao gzip mas usa um algoritmo de compressã diferente, resultando em arquivos menores ao custo de mais tempo. É mais usado para compactar arquivos grandes.

- **$ bzip2 *** : Compacta todos os arquivos no diretório atual, cada arquivo é então reposto por um arquivo com a extensão .bz2;
- **$ bunzip2 *.bz2** : Descompacta todos os arquivos com extensão **.bz2** no diretório atual.

Procurando e trabalhando arquivos

Comprimindo dados - xz

O **xz** é o utilitário de compressão mais eficiente quanto ao espaço usado no Linux, novamente sacrificando velocidade por uma compressão mais forte.

- **$ xz *** : Compacta todos os arquivos no diretório atual, repondo-os com um arquivo de extensão .xz;
- **$ xz foo** : Compacta o arquivo **foo** em **foo.xz** usando o nível de compressão padrão (-6), e remove **foo** caso sucedido;
- **$ xz -dk bar.xz** : Descompacta **bar.xz** em **bar**, mas não remove o arquivo compactado mesmo em caso de sucesso;
- **$ xz -dcf a.txt b.txt.xz > abcd.txt** : Descompacta uma mistura de arquivos comprimidos e descomprimidos para a saída padrão, usando um único comando;
- **$ xz -d *.xz** : Descompacta todos os arquivos .xz.

PETEEC
ENGENHARIA DE COMPUTAÇÃO

Procurando e trabalhando arquivos

Comprimindo dados - zip

O programa **zip** não é muito usado para compactar arquivos no Linux, mas é bastante necessário para examinar e descompactar arquivos de outros sistemas operacionais.

- **$ zip backup *** : Compacta todos os arquivos no diretório atual e os coloca no arquivo **backup.zip**;
- **$ zip -r backup.zip ~** : Arquiva seu diretório home (~), junto de todos os seus arquivos e subdiretórios no arquivo **backup.zip**;
- **$ unzip backup.zip** : Extrai todos os arquivos em **backup.zip** e os coloca no diretório atual.

PETEEC
ENGENHARIA DE COMPUTAÇÃO

Procurando e trabalhando arquivos

Comprimindo dados - tar

O programa **tar** te permite criar arquivos de arquivo, chamados **tarballs**. Ao mesmo tempo, te permite compactá-los ao criar o arquivo, ou descompactar ao extrair seus conteúdos.

- **$ tar xvf mydir.tar** : Extrai todos os arquivos em **mydir.tar** para o diretório **mydir**;
- **$tar zcvf mydir.tar.gz mydir** : Cria o arquivo e o compacta com gzip;
- **$tar jcvf mydir.tar.bz2 mydir** : Cria o arquivo e o compacta com bzip2;
- **$tar Jcvf mydir.tar.xz mydir** : Cria o arquivo e o compacta com xz;
- **$tar xvf mydir.tar.gz** : Descompacta e extrai todos os arquivos em **mydir.tar.gz** para o diretório **mydir**;

PETeco
ENGENHARIA DE COMPUTAÇÃO

Procurando e trabalhando arquivos

Cópia disco a disco - dd

O programa **dd** é muito útil para criar cópias de espaço bruto de disco. Por exemplo, para copiar sua **MBR** (setor dos primeiros 512 bytes no disco que contém a tabela de partições), você pode digitar:

- $ dd if=/dev/sda of=sda.mbr bs=512 count=1

Para usar o dd para realizar a cópia de um disco em outro (DELETANDO TUDO NESTE SEGUNDO), digite:

- Dd if=/dev/sda of=/dev/sdb