Aprendendo o Shell do Linux

Introdução

Na computação, uma **interface com o usuário** (UI, User Interface) é o espaço onde ocorre a interação entre o usuário e a máquina. No caso de **sistemas operacionais**, a interface com o usuário é onde se dá o acesso aos serviços providos pelo sistema e aos programas instalados nele. Os sistemas operacionais oferecem dois tipos de interface com o usuário: **interface gráfica** e interface por **linhas de comando**. Na interface gráfica (GUI, Graphical User Interface) as interações ocorrem através da manipulação direta com elementos gráficos na tela, tais como ícones, janelas e botões. Na interface por linha de comando (CLI, Command-Line Interface), as interações ocorrem no que é chamado de **terminal**, **console** ou **shell**, que é onde o usuário digita os comandos e recebe as mensagens do sistema ou do programa sendo executado, tudo através de textos. O **prompt** de comandos é onde o usuário digita os comandos dentro do terminal.

O terminal do sistema operacional **Linux** utiliza a linguagem de comando **Bash**, uma ferramenta muito poderosa, que fornece muitos recursos para auxiliar no trabalho. O Linux é baseado no sistema **UNIX**, portanto, a maioria dos comandos do terminal é compatível com o de outros sistemas UNIX tais como o **Mac**.

As interfaces gráficas de usuário (GUIs) são úteis para muitas tarefas, mas não são boas para todas as tarefas. Os computadores deveriam nos livrar do trabalho manual, mas quantas vezes já realizamos alguma tarefa que o computador deveria ser capaz de fazer, mas acabamos fazendo o trabalho sozinho, usando tediosamente o mouse? Apontando e clicando, apontando e clicando…

Certa vez um autor disse que quando somos crianças, aprendemos a usar o computador olhando as imagens. Quando crescemos, aprendemos a ler e a escrever. Agora vamos começar a trabalhar!

(Observação: os comandos aqui foram executados no sistema operacional Ubuntu 20.04. Pode ser que em outras distribuições Linux não haja todos os comandos daqui.)

Terminal

Você pode iniciar qualquer programa a partir do terminal, desde aplicativos gráficos, como o Firefox, até utilitários de linha de comando. No Linux, diferente do Windows, por exemplo, os programas não possuem uma extensão `.exe`. Portanto, para abrir o programa Firefox pelo terminal do Linux, você pode digitar simplesmente o comando `firefox` dentro do terminal. Comandos do terminal também aceitam **argumentos** (ou **parâmetros**), que são opções acrescentadas ao comando. Por exemplo, para iniciar o Firefox diretamente de uma URL, basta passá-la como argumento: `firefox ifms.edu.br`. Esse comando abre o Firefox já na página inicial do site do **IFMS**.

Trabalhando com diretórios e arquivos

Ao abrir um terminal, ele começa no **diretório padrão**, que fica localizado em `/home/nome_usuário`. Isso quer dizer que qualquer comando que digitar será executado dentro

deste diretório, a menos que mude o diretório. Por exemplo, o comando `cat texto.txt` exibe o conteúdo do arquivo `texto.txt` localizado dentro do diretório padrão.

Algumas observações quanto à navegação de diretórios e arquivos pela linha de comando:

- Para representar o diretório atual (**working directory**, onde o terminal se encontra aberto), usa um ponto `.`.
- Para representar o diretório pai (**parent directory**), usa dois pontos `..`. O diretório pai é o diretório anterior ao atual, ou seja, o que contém o diretório atual.
- Para representar o diretório padrão (**home**), usa o til e a barra `~/`. É o diretório do usuário, possui o mesmo nome dele.
- Para representar o diretório raiz (**root**), usa uma única barra `/`. O diretório raiz é o primeiro diretório do sistema, ou o diretório que contém todos os demais. Mas atenção, não mexa nos diretórios que não estejam na home, a não ser que você saiba o que está fazendo, pois é onde residem os arquivos do sistema.
- Em um caminho (**path**) entre diretórios, usa uma barra `/` para separar cada diretório. Para acessar um diretório, pode usar o caminho relativo ou o caminho absoluto. O caminho absoluto começa do diretório raiz e segue cada diretório até chegar ao diretório ou arquivo desejado. O caminho relativo começa do diretório atual. Por exemplo, suponha que o usuário *fulano* tenha um arquivo chamado `fonte.c` dentro do diretório `Documentos`, dentro do diretório padrão. Suponha que o diretório atual no terminal seja o diretório padrão. Então, o caminho absoluto para o arquivo é `/home/fulano/Documentos/fonte.c`, e o caminho relativo é `Documentos/fonte.c`. Observe que o caminho relativo não começa com uma barra, pois a barra representa o diretório raiz.
- Arquivos ou diretórios que contêm espaço ` ` no nome devem ser escritos entre aspas duplas, por exemplo, `"arquivo com espaço.txt"`, senão o terminal interpreta cada palavra como um comando ou argumento diferente.

Você pode abrir um diretório no terminal a partir da janela do explorador de arquivos. Para isso basta clicar com o botão direito do mouse em cima do diretório e depois escolher a opção “abrir no terminal”. Uma alternativa é copiar a localização na barra de localização do explorador de arquivos e colar a localização no terminal, precedida pelo comando `cd ` .

A seguir, são descritos alguns dos principais comandos para trabalhar com diretórios e arquivos pelo terminal:

1. `cd dir`. Troca o diretório atual para o diretório `dir`. Exemplos:
   a) `cd Documentos/programas/`. Troca para o diretório `programas`, dentro do diretório `Documentos`, dentro do diretório atual. Note que pode omitir a barra do último diretório, portanto o comando `cd Documentos/programas` é equivalente.
   b) `cd ..`. Troca para o diretório pai (diretório anterior ao atual).
   c) `cd ~/` ou `cd ~`. Troca para o diretório home, ou padrão.

    d) `cd "~/Downloads/minhas tarefas"`. Troca para o diretório `minhas tarefas`, dentro do diretório `Downloads`, dentro do diretório padrão. Observe que quando algum diretório contém espaços, o caminho (path) deve estar entre aspas duplas.

2. `ls dir`. Lista os arquivos e diretórios do diretório `dir` ou do diretório atual se não for informado o `dir`.
3. `ll dir`. Lista os arquivos e diretórios do diretório `dir` ou do diretório atual se não for informado o `dir`. A diferença pro comando anterior é que lista também os arquivos e diretórios ocultos, e mais informações de cada item, além de acrescentar uma barra aos diretórios. É um apelido pro comando `ls -laF`. Mais informações do comando `ll` na seção Usuários e permissões.
4. `pwd`. Exibe o caminho absoluto do diretório atual.
5. `mkdir dir`. Cria o diretório `dir`. Pode criar vários diretórios de uma vez, como em `mkdir dir1 dir2 dir3`. Para criar uma descendência de diretórios: `mkdir -p avo/pai/filho`.
6. `cp file1 file2`. Copia `file1` para `file2`. Se já existir `file2`, o substitui. Se `file2` for diretório, copia para dentro do diretório, permanecendo o nome de `file1`. Exemplos:
    a) `cp ~/Documentos/teste.txt .`. Copia o arquivo `teste.txt`, dentro do diretório `Documentos`, dentro do diretório padrão, para o diretório atual, representado por um ponto.
    b) `cp aula1.pdf aula2.pdf aula3.pdf aulas`. Copia os arquivos para o diretório `aulas`.
7. `mv file1 file2`. Move/renomeia `file1` para `file2`. Exemplos:
    a) `mv aula1.pdf aula01.pdf`. Renomeia o arquivo `aula1.pdf` para `aula01.pdf`, pois estão no mesmo diretório.
    b) `mv aula01.pdf ../aula01.pdf`. Move o arquivo `aula01.pdf` para o diretório pai, pois estão em diretórios diferentes.
    c) `mv file1 file2 dir`. Move os arquivos `file1` e `file2` para `dir`. Pode mover vários arquivos de uma vez.
8. `rm file1 file2`. Remove os arquivos listados. Pode remover apenas um ou vários arquivos de uma vez.
9. `rm dir1 dir2 -rf`. Remove os diretórios listados. Pode remover apenas um ou vários diretórios de uma vez. Para remover diretórios com o comando `rm`, deve acrescentar as opções `-rf`. A opção `-r` significa remover diretório e a opção `-f` significa forçar a remoção de diretórios que não estejam vazios.

Principais comandos do terminal

Alguns dos principais comandos usados no terminal são descritos a seguir:

- `man programa`. Exibe um manual de instruções de `programa`. Aperte [Q] para sair. Exemplo: `man cp` exibe o manual do comando `cp`.
- `gcc source.c -o program`. Compila o código-fonte em C `source.c` e gera um programa executável `program`. A opção `-o` seguida do nome do programa é opcional, caso seja omitida, é gerado um programa com o nome padrão `a.out`.
- `./program`. Executa o programa `program` localizado no diretório atual.
- `cat arquivo`. Exibe o conteúdo de `arquivo`.
- `find dir -iname "file"`. Procura em todos os diretórios dentro de `dir` pelo arquivo `file`. A opção `-iname` serve para não distinguir entre letras maiúsculas e minúsculas. Atenção: é recomendado colocar `file` entre aspas (casos de arquivos com espaços). Exemplo:
  - → `find . -iname "aula*"`. Procura em todos os diretórios dentro do diretório atual por arquivos que comecem com o nome `aula`. O asterisco `*` é um símbolo especial que representa qualquer combinação de caracteres depois da palavra `aula`.
  - → `find . -maxdepth 1 -iname "*.mp3"`. Procura apenas no diretório atual por arquivos com extensão `.mp3`. A opção `-maxdepth` limita os níveis da recursão.
- `locate -b text`. Localiza por todos os arquivos ou diretórios que contém `text` no nome. Ao omitir a opção `-b`, também são listados os arquivos contidos no diretório localizado. O comando `locate` é uma alternativa mais rápida ao comando `find`, já que a sua busca se dá em uma base de dados do sistema que mantém os nomes de todos os seus arquivos. Se a base de dados estiver desatualizada, o comando `locate` pode trazer resultados desatualizados. Para atualizar a base de dados dos nomes dos arquivos, use o comando `updatedb`. Por padrão, `updatedb` indexa todo o sistema. Para excluir algumas entradas do índice, edite o arquivo `/etc/updatedb.conf`. Para saber mais sobre este arquivo, leia seu manual, que é curtinho, com `man updatedb.conf`. Por exemplo, eu gosto de excluir os seguintes diretórios:

  → `PRUNENAMES=".git .bzr .hg .svn .debris"`
  → `PRUNEPATHS="/bin /boot /dev /etc/dkms /lib /lib32 /lib64 /libx32 /media /mnt /opt /proc /pulse /root /run /sbin /srv /sys /tmp /usr/bin /usr/games /usr/include /usr/lib /usr/lib32 /usr/lib64 /usr/libexec /usr/libx32 /usr/sbin /usr/src /var" # original "/tmp /var/spool /media /var/lib/os-prober /var/lib/ceph"`
- `grep "palavra" arquivo.` Procura por “palavra” em `arquivo`. Você também pode trocar `grep` por `fgrep`, que é mais rápido, mas sem expressão regular. Exemplos:
  - → `grep "teste" temp.txt`. Procura pela palavra “teste” no arquivo `temp.txt`.
  - → `grep -s "teste" *`. Procura por “teste” em todos os arquivos do diretório atual. A opção `-s` serve para ignorar possíveis erros.
  - → `grep -r -i -I "word" .`. Procura recursivamente no diretório atual por arquivos de texto que contêm a palavra `word`. Procurar recursivamente significa procurar em todos os diretórios dentro do diretório. A opção `-r` procura recursivamente. A opção `-i` serve para não distinguir entre letras maiúsculas e minúsculas. A opção `-I` ignora arquivos binários.

→ `grep -r -n -o "temp" --include=*.{js,css,html,htm} ./src`. Procura recursivamente no diretório `src` pela palavra “temp” em todos os arquivos com extensão js, css, html ou htm. A opção `-n` exibe também o número da linha onde encontrou a palavra no arquivo. A opção `-o` exibe apenas o texto correspondente. A opção `--include` filtra a busca apenas nos arquivos que correspondam ao critério especificado.

- `history`. Exibe todos os comandos já digitados no terminal.
- `echo`. Exibe o que foi digitado. Útil para exibir o valor de variáveis de ambiente do sistema. Por exemplo, `echo $PATH` exibe o valor da variável `PATH`, que diz ao terminal quais diretórios procurar por arquivos executáveis.
- `sudo dpkg -i pkg.deb`. Instala o pacote `pkg.deb`. O modo de instalar offline programas em sistemas baseados no Debian é através de pacotes `.deb`. Muitas vezes esses pacotes precisam de dependências que o sistema não possui, portanto, geralmente é comum depois executar o comando `sudo apt install -f` para instalar as dependências necessárias para o programa funcionar.
- `exit`. Fecha o terminal.
- `nano file`. Abre o arquivo `file` no editor de texto pela linha de comandos `nano`. Se `file` for omitido, abre o editor de texto com um novo arquivo vazio.
- `ip a`. Exibe o ip local do computador.
- `nautilus dir`. Abre o diretório com o explorador de arquivos Nautilus.
- `pdfseparate sample.pdf sample-%d.pdf`. Extrai todas as páginas de `sample.pdf`, se, por exemplo, `sample.pdf` tiver 3 páginas, ele produz `sample-1.pdf`, `sample-2.pdf`, `sample-3.pdf`. Para especificar as páginas a serem extraídas, pode usar as seguintes opções: `pdfseparate -f 2 -l 4 sample.pdf sample-%d.pdf`. Em que `-f 2` diz que a extração começa na página 2 e `-l 4` diz que a extração termina na página 4.
- `pdfunite sample1.pdf sample2.pdf sample.pdf`. Junta todas as páginas de `sample1.pdf` e `sample2.pdf` (nessa ordem) e cria `sample.pdf`.
- `convert img1.png img2.png sample.pdf`. Cada imagem se torna uma página de um único arquivo pdf.
- `ping url`. Testa a conexão com algum endereço de internet.
- `shutdow now`. Desliga o computador agora.
- `shutdown -r now`. Reinicia o computador agora.
- `snap info software`. Exibe informações de um software disponível na snapstore.
- `snap install software`. Instala um software disponível na snapstore.
- `apt show pkg`. Exibe informações de um pacote disponível no gerenciador de pacotes do sistema.
- `sudo apt install pkg`. Instala um pacote disponível no gerenciador de pacotes do sistema.

- `sudo apt remove pkg`. Desinstala um pacote instalado.
- `dpkg -l`. Lista todos os pacotes instalados.
- `dpkg -l pkg`. Verifica se o pacote `pkg` está instalado.
- `whereis program`. Exibe a localização do binário, do código-fonte e do manual de `program`.
- `wget url`. Obtém o arquivo do link `url`.
- `xdg-open file`. Abre o arquivo `file` com seu programa padrão. Útil quando se tem um arquivo mas não se sabe qual programa abre ele.

Primeiro plano (foreground) e segundo plano (background)

Se digitar o comando `gedit` para abrir o editor de texto, o terminal fica bloqueado enquanto o programa estiver aberto. Para iniciar um programa em segundo plano, isto é, sem bloquear o terminal, pode anexar o e-comercial `&` ao final do comando. Por exemplo, `gedit &`. Geralmente coloca o e-comercial ao final de comandos que abrem aplicativos gráficos, deixando, assim, o terminal livre enquanto os aplicativos são executados. Para trazer um programa de volta ao primeiro plano no terminal, digita `fg %ID`, em que `ID` é o número de identificação do programa, que aparece entre colchetes logo que é iniciado em segundo plano. Se não se lembrar do ID do programa, pode digitar o comando `jobs`, que lista os programas sendo executados em segundo plano. Exemplo:

1. `gedit &`. Abre o editor de textos `gedit` em segundo plano.
2. `jobs`. Lista os programas rodando em segundo plano e seus IDs.
3. `fg %1`. Traz o programa de ID 1 para o primeiro plano. Se omitir o ID, traz o último programa para o primeiro plano.
4. `bg %1`. Traz o programa de ID 1 para o segundo plano. Se omitir o ID, traz o último programa para o segundo plano.

Principais teclas de atalho

Você pode digitar combinações de teclas para acionar certos comandos. A seguir são descritas algumas delas:

1. [TAB]. Completa o comando com a ocorrência mais próxima do que foi digitado. Se houver mais de uma ocorrência, aperte [TAB] mais uma vez para exibir todas as possíveis opções. Isso evita digitar completamente todo o comando. Ao digitar parte do comando e a tecla [TAB], o terminal completa o comando. Isso funciona para nomes de comandos, argumentos, diretórios, arquivos… Exemplo:
   a) `firef + [TAB]`. Completa o comando com `firefox`.
   b) `cd Down + [TAB]`. Completa o comando com `cd Downloads`.
   c) `cd aul + [TAB] + [TAB]`. Supondo que haja vários arquivos que começam com o nome `aul`, são listados todos esses arquivos.

2. [↑] e [↓]. As setas para cima e para baixo navegam entre os últimos comandos dados. Se quiser repetir o penúltimo comando, basta digitar duas vezes a seta para cima [↑].
3. [CTRL] + [L]. Limpa a tela do terminal.
4. [CTRL] + [C]. Envia o sinal SIGINT, que finaliza a execução do programa atualmente sendo executado em primeiro plano no terminal. Se não houver nenhum programa sendo executado em primeiro plano, apaga o texto digitado no prompt.
5. [CTRL] + [R]. Busca por um comando no histórico. Depois é só digitar a parte que se lembra do comando e o terminal vai mostrando sugestões de comandos conforme vai digitando. Ao digitar seguidas vezes [CTRL] + [R], uma sequência de comandos sugeridos vai sendo exibida. Depois é só teclar [ENTER] ao achar o comando desejado, que ele será executado. Se passar pelo comando, pode voltar ao anterior teclando [CTRL] + [S] (dependendo da versão do sistema, é necessário digitar o comando `stty -ixon` para atribuir o [CTRL] + [S] à função de voltar a sugestão).
6. [CTRL] + [Z]. Envia o sinal SIGTSTP, que interrompe o programa atualmente sendo executado em primeiro plano no terminal e o envia ao segundo plano. Para deixá-lo em segundo plano mas sendo executado, digite o comand `bg`. Para trazê-lo ao primeiro plano, digite o comando `fg`. Como dito antes, pode usar `fg %n` ou `bg %n` para especificar um programa.
7. [CTRL] + [D]. Envia o caractere EOF, usado para indicar o fim do arquivo (end of file). Muitos programas ficam esperando por dados de entrada até receber o sinal EOF. Caso nenhum programa esteja sendo executado, o atalho fecha o terminal.
8. [CTRL] + [SHIFT] + [C]. Copia o texto selecionado.
9. [CTRL] + [SHIFT] + [V]. Cola o texto copiado.
10. Ctrl+U. Apaga do cursor para trás.
11. Ctrl+Shift+↑, Ctrl+Shift+↓. Rola uma linha para cima ou para baixo pelo terminal
12. Shift+PgUp, Shift+PgDw. Rola uma página para cima ou para baixo pelo terminal.

O programa less

O programa `less` permite ler arquivos de texto pelo terminal. Bom para arquivos de textos grandes, que ficariam inviáveis de serem abertos pelo `cat`. Para abrir um arquivo de texto pelo `less`, digita o comando `less caminho/do/arquivo`. Depois de aberto o arquivo pelo `less`, pode digitar algumas teclas de atalho para realizar certas ações. Algumas delas são descritas a seguir:

Comando	Ação
[PAGE UP] ou [B]	Rola uma página acima
[PAGE DOWN] ou [ESPAÇO]	Rola uma página abaixo
[HOME] ou [1][G]	Vai para o início do arquivo
[END] ou [SHIFT]+[G]	Vai para o fim do arquivo
N[G]	Vai para a linha N
/texto	Busca por “texto” no arquivo
[N]	Repete a última busca

[SHIFT]+[N]	Repete a última busca de trás para frente
[H]	Exibe a lista completa de comandos e opções
[Q]	Sai do programa

Esperando mudanças em arquivos com tail ou less

O comando `tail file` exibe as 10 últimas linhas do arquivo `file`. Para exibir as x últimas linhas, adicione a opção `-x`. O comando `tail` geralmente é usado como monitor de arquivos de registros, ou log files. Toda vez que um novo registro é adicionado ao final do arquivo por um programa externo, o `tail` pode exibir essa mudança com a opção `-f`. Faça o teste abrindo um arquivo de texto com `tail -f test.txt` em um terminal e acrescentando algo nele com `echo acrescentado no final >> text.txt` em outro terminal Mas essa opção funciona apenas com programas que adicionam textos no final do arquivo. Se quiser monitorar qualquer mudança no arquivo, feita por qualquer programa, use `tail -F file`.

O comando `less` também permite ficar escutando por mudanças no final do arquivo, basta teclar o atalho [SHIFT]+[F] depois de aberto um arquivo. Para sair do monitoramento, tecle [CTRL]+[C]. uma desvantagem é que o `less` carrega todo o arquivo, enquanto o `tail` carrega apenas o final do arquivo, e para monitorar arquivos de log, geralmente só o final é importante. Mas para carregar só o final do arquivo e já abrir no modo monitoramento, pode executar `less -n +F file`. Agora, se quiser monitorar qualquer mudança no arquivo, feita por qualquer programa, use `less -n --follow-name +F file`.

A vantagem do `tail` é sua simplicidade. A vantagem do `less` é que consegue navegar pelo conteúdo do arquivo.

Caracteres “curingas” (Wildcard)

Sendo franco, muitas tarefas executadas pela linha de comando são mais fáceis de serem feitas usando um explorador gráfico de arquivos, por exemplo, arrastar e soltar um arquivo de um diretório para outro, cortar, colar, excluir arquivos, etc. Mas então qual a vantagem de usar a linha de comando?

A resposta é “poder” e “flexibilidade”. Embora seja mais fácil realizar ações simples de manipulação de arquivos pelo explorador gráfico de arquivos (vulgo “janela”), tarefas mais complicadas sugerem um terminal de linhas de comando. Por exemplo, suponha que queira copiar arquivos HTML de um diretório para outro, mas apenas arquivos que ainda não foram copiados ou que sejam mais recentes do que as versões do diretório destino. Fazer esse trabalho manualmente pelo explorador de arquivos é trabalhoso, mas muito fácil pela linha de comando: `cp -u *.html destination`.

Como o shell usa bastante nomes de arquivos, ele fornece caracteres especiais para ajudar a especificar rapidamente grupos de nomes de arquivos. Esses caracteres especiais são chamados de curingas. Os curingas permitem selecionar nomes de arquivos baseados em padrões de caracteres. A tabela abaixo lista alguns curingas e o que eles selecionam:

Curinga	Significado

*****	Corresponde a qualquer sequência de zero ou mais caracteres
?	Corresponde a um único caractere
[caracteres]	Corresponde a qualquer caractere dentro dos colchetes
[!caracteres]	Corresponde a qualquer caractere que não seja os que estão dentro dos colchetes

Usando curingas, é possível construir critérios de seleção muito sofisticados para nomes de arquivos. Aqui estão alguns exemplos de padrões e o que eles correspondem:

Padrão	Correspondência
g*	Qualquer arquivo que comece com a letra “g”
b*.txt	Qualquer arquivo que comece com a letra “b” e termine com os caracteres “.txt”
Data???	Qualquer arquivo que comece com “Data” e tenha exatamente mais três caracteres quaisquer
[abc]*	Qualquer arquivo que comece com as letras “a”, “b” ou “c”
***[0-9]**	Qualquer arquivo que termine com um número
[A-Z]*	Qualquer arquivo que comece com uma letra maiúscula
[!0-9]	Qualquer arquivo que não contenha números no nome.

Pipelines

Um dos recursos mais poderosos do shell Linux é redirecionar a saída de um programa como entrada para outro programa. Ou seja, conectar múltiplos comandos juntos usando o que é chamado de “pipelines” (canalização). Para fazer isso, você usa a barra vertical `|`. Exemplo, suponha que queira exibir as informações de todos os arquivos dentro de determinado diretório usando o comando `ls`, e suponha que esse diretório tenha muitos arquivos. Pode redirecionar a saída para o `less`, e, assim, usar opções como rolagem da página e busca de palavras mais facilmente: `ls -l | less`. Alguns exemplos de uso de pipelines são descritos a seguir:

1. `ls -lt | head -5`. Exibe os 5 arquivos mais recentes do diretório atual.
2. `ls -l | grep "Jan 20"`. Lista os arquivos do diretório atual modificados em 20 de janeiro.
3. `du | sort -nr`. Exibe uma lista de diretórios e quanto de espaço eles ocupam, ordenados do maior para o menor. A opção `-n` usa a ordenação numérica, já que a padrão é a alfabética. A opção `-r` ordena de trás para frente, ou seja, do maior para o menor. Para listar, ordenado por tamanho, diretórios e arquivos numa profundidade de até duas pastas, use o comando `du -ad 1 . | sort -nr >arquivos.txt`. A opção `-a` lista arquivos também, já que o padrão só lista diretórios. A opção `-d 1` lista diretórios numa profundidade de até 1 distância do diretório pesquisado. E a opção `>arquivos.txt` salva o resultado em `arquivos.txt`.
4. `find . -maxdepth 1 -type f -print | wc -l`. Exibe o número total de arquivos (sem contar os diretórios) no diretório atual.
5. `find . -type f -printf "%s\t%p\n" | sort -n`. Exibe recursivamente todos os arquivos dentro do diretório atual ordenados por tamanho. A opção `-printf` permite

formatar a saída do comando `find`. `%s` imprime o tamanho de cada arquivo em bytes, `\t` imprime uma tabulação, `%p` imprime o nome do arquivo e `\n` imprime uma quebra de linha.

Redirecionamento de entrada/saída

A entrada padrão é tudo aquilo que se digita. E a saída padrão é tudo aquilo que o terminal exibe (mensagens). Você pode facilmente redirecionar a saída de um programa para um arquivo usando o sinal de maior `>` seguido do nome do arquivo que conterá a saída. Ou seja, ao invés do programa exibir as mensagens pelo terminal, elas são salvas diretamente no arquivo. Por exemplo, pode usar o comando `echo ola mundo > helloworld.txt` para escrever "ola mundo" no arquivo `helloworld.txt`. Se o arquivo já existir, ele será sobrescrito. Se quiser adicionar ao arquivo ao invés de sobrescrevê-lo, usa dois sinais de maior `>>`: `echo vamos la >> helloworld.txt`.

Se quiser criar um arquivo com mais de uma linha, pode usar o comando `cat > helloworld.txt`. Pode digitar várias linhas de texto, e quando quiser encerrar, envie o caractere especial EOF com o atalho [CTRL]+[D].

Como outro exemplo, suponha que queira salvar em um arquivo o nome de todos os arquivos PDF dentro de determinado diretório. Pode redirecionar a saída do comando `ls`: `ls *.pdf >arquivos_pdf.txt`.

Agora pode ver o nome de todos os arquivos PDF usando o comando `less arquivos_pdf.txt`. Para adicionar ao arquivo `arquivos_pdf.txt` os nomes de arquivos PDF do diretório Documentos: `ls Documentos/*.pdf >>arquivos_pdf.txt`.

Também pode facilmente redirecionar a entrada de um arquivo usando o sinal de menor `<` seguido do nome do arquivo de onde o programa lerá a entrada. Ou seja, ao invés de digitar os comandos, eles são lidos diretamente do arquivo. Exemplo, suponha que queira exibir o conteúdo do arquivo `arquivos_pdf.txt` em ordem alfabética. Pode fazer: `sort <arquivos_pdf.txt`.

Pode também redirecionar a saída e a entrada ao mesmo tempo. Suponha, então, que queira salvar em outro arquivo o conteúdo de `arquivos_pdf.txt`, só que alfabeticamente. Pode usar: `sort <arquivos_pdf.txt >arquivos_pdf_alfabeticamente.txt`.

Os argumentos de um comando são lidos da entrada padrão até uma quebra de linha (ENTER). Se quiser digitar os argumentos em várias linhas, pode usar o operador delimitador `<<`, chamado de "here document", que lê várias linhas da entrada padrão até encontrar uma linha que corresponda ao delimitador. Por exemplo, ao digitar o comando `wc << fim`, o comando `wc` lerá várias linhas de texto até encontrar uma linha contendo apenas a palavra "fim", usada como delimitador. Depois ele exibe a quantidade de linhas, palavras e bytes lidos da entrada padrão, respectivamente.

Anteriormente você viu que é possível criar um arquivo com o comando `cat > arquivo.txt`, em que o comando `cat` será encerrado ao receber um EOF. Mas pode mudar o delimitador de fim do arquivo com `cat << end > arquivo.txt`, assim, o `cat` será encerrado ao encontrar uma linha contendo apenas a palavra "end".

Em muitos comandos, como o `find` ou o `grep`, as vezes aparecem mensagens de erro. Geralmente mensagens de erros são exibidos no stderr, que por padrão, é direcionado para o mesmo lugar que o stdout. Para não exibir mensagens de stderr, você pode redirecionar o stderr para lugar algum, da

seguinte forma: `cmd 2>/dev/null`, onde `cmd` é algum comando que pode gerar uma mensagem de stderr.

Você não consegue gravar a saída em um arquivo que não tem permissão de escrita. Por exemplo, se você quiser adicionar a opção de não diferenciar maiúsculo de minúsculo no terminal, assim não funciona:

```
echo 'set completion-ignore-case On' >>/etc/inputrc
```

Neste caso, você pode usar o comand `tee`, que redireciona sua entrada padrão para a saída padrão ou para arquivos. Observe que aqui, `tee` é executado como `sudo` para ter permissão de escrita, e a opção `-a` é para adicionar ao arquivo, e não sobrescrevê-lo por completo:

```
echo 'set completion-ignore-case On' | sudo tee -a /etc/inputrc
```

O comando xargs

O comando `xargs` serve para executar determinado comando várias vezes, dependendo do tamanho da entrada. Ele é uma alternativa a ser usada junto com o pipeline `|`. A seguir, alguns exemplos de uso do comando `xargs`.

Para encontrar e remover recursivamente no diretório atual por todos os arquivos `.bak`, execute: `find . -name "*.bak" | xargs rm -f`.

Quando um comando recebe mais de um argumento, usa a opção `-I` para dar um apelido a um dos argumentos. Por exemplo, o comando `mv` precisa de pelo menos dois argumentos, o arquivo de origem e o arquivo de destino. Portanto, para mover todos os arquivos `.bak` dentro de todos os diretórios do diretório atual para o diretório `~/.backup`, execute: `find . -name "*.bak" -print0 | xargs -0 -I file mv file ~/.backup`.

As opções `-print0` e `-0` dos comandos `find` e `xargs`, respectivamente, servem para separar cada entrada que será lida pelo comando `xargs` por um caractere nulo, ao invés de separar cada entrada por um caractere de espaço em branco (delimitador padrão). Usa essa opção quando na entrada houver arquivos ou diretórios que contenham um caractere de espaço em branco. Para cada entrada resultante do comando `find`, a opção `-I` apelida para o nome `file`. Portanto, `xargs` executará o comando `mv` x vezes, sendo x a quantidade de arquivos encontrados pelo `find`. Assim, cada arquivo será movido para o diretório `~/.backup`.

Para encontrar todos os arquivos `.mp3` dentro de um diretório recursivamente e tocá-los com o programa `mplayer`, usa: `find . -iname "*.mp3" -print0 | xargs -0 mplayer`.

Para copiar todos os arquivos MP3 em outro local, pode usar apenas o `cp` da seguinte forma: `cp -r -v -p ./ ~/Music`. Observe que quando num comando há várias opções sem parâmetros, pode juntá-los. O seguinte comando é equivalente ao anterior: `cp -rvp ./ ~/Music`.

No entanto, o comando `cp` pode falhar se ocorrer um erro, como se o número de arquivos for muito grande para o comando `cp`. `xargs` em combinação com `find` pode lidar com essa operação muito bem. `xargs` é mais eficiente em termos de recursos e não vai parar com um erro: `find ./ -type f -name "*.mp3" -print0 | xargs -0rI file cp -vp file ~/Music`. A opção `-r` impede o `xargs` de executar alguma coisa se a entrada for vazia.

O comando diff

`diff` serve para diferenciar dois arquivos. Suponha que você tenha um arquivo `d1.txt`, com o seguinte conteúdo:

```
I need to buy apples.
I need to go to the store.
I need to run the laundry.
I need to wash the car.
I need to get the car detailed.
When I get home, I'll wash the dog.
I promise.
```

Suponha que você também tenha o arquivo `d2.txt`:

```
I need to buy apples.
I need to go to the store.
I need to do the laundry.
I need to wash the dog.
I need to get the car detailed.
Oh yeah, I also need to buy grated cheese.
When I get home, I'll wash the dog.
```

Qual a diferença entre eles? Basta digitar o comando `diff --color -u d1.txt d2.txt`, em que a opção `--color` colore as diferenças, e a opção `-u` exibe as diferenças no modo unificado (mais legível). A seguir, a saída do comando:

```
--- d1.txt      2021-01-14 14:59:21.653153587 -0400
+++ d2.txt      2021-01-14 14:56:06.777097433 -0400
@@ -1,7 +1,7 @@
 I need to buy apples.
 I need to go to the store.
-I need to run the laundry.
-I need to wash the car.
+I need to do the laundry.
+I need to wash the dog.
 I need to get the car detailed.
+Oh yeah, I also need to buy grated cheese.
 When I get home, I'll wash the dog.
-I promise.
```

As linhas que começam com espaço ` ` são iguais nos dois arquivos. As linhas que começam com menos `-` existem apenas no primeiro arquivo, e as linhas que começam com mais `+` existem apenas no segundo arquivo. Para exibir a diferença entre os arquivos lado a lado, basta digitar o comando `diff --color -y d1.txt d2.txt`.

Para ver a diferença da saída entre dois comandos: `diff <(ls old) <(ls new)`.

Comandos extensos

Ao digitar um comando, se der ENTER, ele é executado. Mas se o comando for muito extenso, e quiser dar ENTER para quebrar o comando, pode digitar uma barra invertida e continuar o comando na linha seguinte:

```
find . -maxdepth 1 -iname "*.mkv" -print0 | \
xargs -0 -r -I file \
cp -v -p file ~/Videos/
```

Operadores de controle

É possível executar vários comandos, um após o outro. Você já conhece o pipeline `|` que envia a saída de um programa para a entrada de outro programa. Mas também há outros operadores de controle para executar um comando após o outro.

A expressão `command1 ; command2` executa `command1` em primeiro plano, e depois de terminado, executa `command2`. É claro que pode haver mais de dois comandos concatenados. Exemplo, `ls ; pwd ; whoami` primeiro exibe os arquivos do diretório atual, depois o path do diretório atual e por fim o nome do usuário, um após o outro.

A expressão `command1 & command2` executa `command1` em segundo plano, e logo em seguida `command2` em primeiro plano. Essa expressão já foi explicada em Primeiro plano (foreground) e segundo plano (background).

A expressão `command1 && command2` primeiro executa `command1` e então, apenas se não ocorreu nenhum erro, executa `command2`, ambos em primeiro plano. Mas se colocar um e-comercial ao final, executa ambos em segundo plano. Essa expressão é boa quando para a execução de um programa é necessário garantir o sucesso da execução de outro programa. Exemplo, `mkdir ~/diretorio && cd ~/diretorio`. Se tentar executar a expressão anterior, o comando `cd` não será executado pois não é possível criar um diretório que já existe. Para configurar um despertador, basta combinar os comandos `sleep ns && mpv song &`, em que `ns` é um número seguido de uma das letra ‘s’, ‘m’, ‘h’ ou ‘d’, que representa um tempo em segundos, minutos, horas ou dias, respectivamente, e `song` é o path para um arquivo de áudio.

A expressão `command1 || command2` executa `command2` apenas se ocorreu um erro em `command1`. Ambos são executados em primeiro plano. Essa expressão é boa quando é necessário tomar alguma atitude quando um programa não executa bem sucedidamente. Exemplo, `ping istononecziste.com || mpv tbmnao || echo site e musica inexistente!` testa a conexão com um site, e se falhar, toca uma música, e se falhar, exibe uma mensagem.

Agrupando comandos

Você pode agrupar comandos desta forma: `{ command-list; }`. Quando agrupamos comandos, a saída de cada comando é concatenada. Por exemplo, esse exemplo redireciona a saída de todos os comandos para o mesmo arquivo:

```
{ echo "Hi there"; pwd; uptime; date; } >/tmp/output
```

Para ver todos os arquivo de dois diretórios que foram modificados em determinado dia:

```
{ ls -l ~/Documents/; ls -l ~/Downloads/; } | grep "Mar 25"
```

Outro caso de uso para agrupar comandos é ao aplicar operadores lógicos. Digamos que tentamos fazer ping em um site para verificar se ele está ativo. Somente se o ping falhar, queremos enviar um SMS para admin e escrever uma mensagem de log. Podemos agrupar o comando `sendSMS` e o comando `writeLog`:

```
ping -c1 "some.website" 1>/dev/null 2>&1 || { sendSMS ; writeLog; }
```

Se você não agrupar os comandos sendSMS e writeLog, o erro será redirecionado apenas para o sendSMS, e o writeLog sempre será executado, independentemente se houve erro ou não.

Para criar várias pastas dentro de um diretório: `mkdir /tmp/{folder1,folder2,folder3}`.

Para copiar vários arquivos:

```
cp /usr/share/applications/{org.kde.okular.desktop,google-
chrome.desktop,clementine.desktop} ~/Desktop/
```

Configurando o bash

Você pode definir apelidos para comandos, por exemplo, ao comparar dois arquivos, em vez de usar o comando completo `diff --color -u file1 file2`, você pode definir um apelido `alias dif="diff --color -u"`, e toda vez que quiser comparar dois arquivos, usar o comando `dif file1 file2`. Digite o comando `alias` sozinho para exibir todos os apelidos configurados pelo terminal. Porém, observe que ao fechar o terminal, e abri-lo novamente, o apelido não funciona mais. Isto porque ao encerrar uma seção do terminal, todas as alterações feitas nesta seção são descartadas. Você pode acrescentar esse apelido, bem como qualquer outro comando ou configuração, no arquivo `~/.bashrc`, e toda vez que abrir um terminal, esse arquivo será executado antes. Por exemplo, você pode definir o apelido `alias c='gnome-calculator'`, e simplesmente digitar `c` toda vez que quiser abrir a calculadora do gnome.

Entretanto, apelidos não funcionam ao digitar o comando pelo atalho [ALT]+[F2]. Neste caso, você pode criar um link simbólico de um programa e adicioná-lo no diretório `~/.local/bin/`. Qualquer programa dentro deste diretório pode ser executado tanto pelo terminal quanto pelo [ALT]+[F2]. Para tanto, primeiramente, crie o diretório com o comando `mkdir ~/.local/bin/`, se ainda não existir. Depois, crie o link simbólico com o comando `ln -s /usr/bin/gnome-calculator ~/.local/bin/c`. Neste caso, criamos um link simbólico `c` para o `gnome-calculator`.

Caso você queira apelidar um comando que tenha argumentos, e fazê-lo funcionar pelo [ALT]+[F2], ainda pode criar um script dentro do diretório `~/.local/bin`. Por exemplo, para criar um comando que abre o `firefox` numa janela privada, crie o seguinte arquivo `pfx` dentro de `~/.local/bin`:

```
#!/bin/sh
firefox --private-window # abre o firefox numa janela privada
```

Depois, dê permissão de execução ao script: `chmod +x ~/.local/bin/pfx`. Pronto, para abrir o `firefox` numa janela privada, é só executar o comando `pfx`, tanto no terminal quanto no [ALT]+[F2].

Usuários e permissões

Em um sistema Linux, diretórios também são considerados como arquivos, mas que podem conter outros arquivos. Para cada arquivo, são atribuídas permissões de acesso para (1) o proprietário do arquivo, (2) os membros de um grupo de usuários relacionados e (3) todos os outros. As permissões podem ser de leitura, escrita ou execução (isto é, quando o arquivo é um programa ou script).

Muitos programas ou arquivos, só podem ser executados ou abertos por um usuário especial, que se chama *superuser*, ou *root* (não confundir com o diretório raiz). O root é um usuário especial no Linux, que possui acesso a todos os arquivos do sistema. Para executar ou abrir esses arquivos com permissão root, coloca a palavra `sudo` antes do comando. Por exemplo, para instalar um programa, é necessário ser root: `sudo apt install program`. Para abrir uma sessão no terminal autenticada (logged in) com root, digite o comando `sudo -i`. Qualquer comando executado numa sessão root, será executado pelo usuário root. Para sair da sessão root, digite o comando `exit`.

Para ver os detalhes de permissões de um arquivo, use o comando `ll`. A seguir, a saída do comando `ll /bin/chmod`.

```
-rwxr-xr-x 1 root root 68112 jul 24 15:01 /bin/chmod*
```

Seguindo a ordem dos caracteres, o significado da saída é:

1. `-`: O tipo do arquivo. Na maioria dos casos pode ser arquivo regular(`-`), diretório (`d`) ou link simbólico soft (`l`). Neste caso, é um arquivo regular.

`-` `rwx` `r-x` `r-x` `1` `root` `root` `68112` `jul 24 15:01` `/bin/chmod*`

2. `rwxr-xr-x`: As permissões do arquivo. Os três primeiros caracteres (`rwx`) se referem às permissões de leitura (r), escrita (w) e execução (x) do proprietário do arquivo, respectivamente. Do mesmo modo, os próximos três caracteres (`r-x`) se referem às permissões do grupo de usuários do arquivo, e os últimos três caracteres (`r-x`) às permissões de qualquer outro usuário. Se houver um traço (`-`) no lugar de uma letra, significa que para aquela letra o usuário não tem permissão. Neste caso, o proprietário do arquivo possui todas as permissões (`rwx`), o grupo de usuários do arquivo possui apenas permissão de leitura e execução (`r-x`), e qualquer outro usuário possui apenas permissão de leitura e execução (`r-x`).
3. `1`: O número de links simbólicos que o arquivo possui.
4. `root`: Nome do usuário proprietário do arquivo. Neste caso, o root é o proprietário.
5. `root`: Nome do grupo de usuário do arquivo. Neste caso, o grupo também se chama root.
6. `68112`: Tamanho em bytes do arquivo.
7. `jul 24 15:01`: Data da última modificação do arquivo.

8. `/bin/chmod*`: Nome do arquivo. Um asterisco significa que é executável.

O comando `chmod`, abreviação para *change mode*, serve para alterar as permissões do arquivo. Há dois modos para especificar as permissões. O primeiro modo se chama notação octal. Como cada tipo de usuário (proprietário, grupo e outro qualquer) possui três tipos de permissões (leitura, escrita e execução), há oito combinações possíveis para cada usuário:

Permissões	**Notação em bits**	**Notação octal**	**Descrição**
`rwx`	111	7	Todas as permissões.
`---`	000	0	Nenhuma permissão.
`--x`	001	1	Permissão de execução.
`-w-`	010	2	Permissão de escrita (não usada).
`-wx`	011	3	Permissão de escrita e execução (não usada).
`r--`	100	4	Permissão de leitura.
`r-x`	101	5	Permissão de leitura e execução.
`rw-`	110	6	Permissão de escrita e leitura.

Algumas combinações não são usadas, pois não há muita aplicação prática, por exemplo, não faz sentido poder escrever mas não ler.

Portanto, o comando `chmod` é executado com 3 números octais: o primeiro representando as permissões do proprietário; o segundo, do grupo; e o terceiro, de qualquer outro usuário. A seguir, exemplos mais usados do comando `chmod`:

Comando	**Permissões**	**Descrição**
`chmod 777 file`	`rwxrwxrwx`	Todos têm todas as permissões
`chmod 755 file`	`rwxr-xr-x`	Proprietário tem permissão completa, e demais possuem permissão de leitura e execução.
`chmod 700 file`	`rwx------`	Apenas proprietário possui permissões.
`chmod 666 file`	`rw-rw-rw-`	Todos possuem apenas permissão de escrita e leitura.
`chmod 644 file`	`rw-r--r--`	Proprietário pode ler e escrever, e outros podem apenas ler.
`chmod 600 file`	`rw-------`	Apenas proprietário pode ler e escrever.

O comando `chmod` também pode ser usado para alterar as permissões de diretórios. Mas o significado das letras ‘r’, ‘w’ e ‘x’ mudam um pouco:

1. `r`. Permite que o conteúdo do diretório seja listado se o atributo x também estiver definido.
2. `w`. Permite que arquivos dentro do diretório sejam criados, excluídos ou renomeados se o atributo x também estiver definido.
3. `x`. Permite que um diretório seja aberto (ou seja, `cd dir`).

A seguir, exemplos mais usados do comando `chmod` para diretórios:

Comando	Permissões	Descrição
`chmod 777 dir`	`rwxrwxrwx`	Todas as permissões para todos.
`chmod 755 dir`	`rwxr-xr-x`	O proprietário tem permissão completa. Os demais podem listar o diretório, mas não alterar nada. Geralmente usado quando se quer compartilhar o diretório com outras pessoas.
`chmod 700`	`rwx------`	O proprietário tem permissão completa. Os demais não têm nenhuma permissão. Usado quando se quer manter o diretório privado de outros usuários.

O outro modo de usar o `chmod` é o modo simbólico. O formato simplificado do comando é `chmod [ugoa][+-=][rwx]`. Várias operações simbólicas podem ser fornecidas, separadas por vírgulas. Uma combinação das letras `ugoa` controla quais usuários a permissão do arquivo será alterada: o usuário proprietário (`u`), usuários do grupo do arquivo (`g`), outros usuários que não estão no grupo (`o`), ou todos os usuários (`a`). Se for omitida, considera como todos os usuários (`a`). O operador `+` adiciona as permissões; `-` remove as permissões; e `=` faz com que sejam as únicas permissões que o arquivo possui. As letras `rwx` selecionam as novas permissões para os usuários afetados: ler (`r`); escrever (`w`); executar (`x`). exemplos:

Comando	Descrição
`chmod a-x sample.txt`	Nega permissão de execução a todos.
`chmod +r sample.txt`	Concede permissão de leitura a todos.
`chmod go+rw sample.txt`	Concede permissão de escrita e leitura ao grupo e outros.
`chmod u+x sample.sh`	Concede permissão de execução ao proprietário.
`chmod u=rwx,g=rx,o=r myfile`	Equivalente à notação octal `chmod 754 myfile`.

O programa `chown`, abreviação de *change owner*, altera o proprietário de um arquivo. Um exemplo de uso é: `sudo chown new_user file`, em que `new_user` é o nome do novo proprietário. Também é possível alterar o grupo de um arquivo através do comando `chgrp`, de *change group*: `chgrp new_group file`.

Para listar todos os usuários do sistema, use o comando `getent passwd`. Cada linha contêm informações de um usuário, separadas por dois-pontos. Observe que há vários usuários “não humanos”, que são criados automaticamente na instalação do sistema ou de outros pacotes. Considere a seguinte linha:

`siqueira:x:1000:1000:siqueira,,,:/home/siqueira:/bin/bash`

Na ordem, as informações são: nome do usuário (`siqueira`); senha criptografada (`x`), se for um x, então a senha criptografada se encontra em `/etc/shadow`; ID do usuário, ou UID (`1000`); ID do grupo do usuário, ou GDI (`1000`); comentário sobre o usuário (`siqueira,,,`); diretório padrão do usuário (`/home/siqueira`); o shell padrão do usuário (`/bin/bash`).

Para listar todos os grupos do sistema, digite `getent group`. Para listar todos os membros de um grupo específico, digite `getent group group_name`. Para listar todos os grupos que um usuário

específico pertence, digite `groups user_name`. Todo usuário tem seu grupo primário, com o mesmo nome dele.

Fonte:

- http://linuxcommand.org/lc3_lts0090.php. Acessado em 17/01/2021.
- https://www.tutorialspoint.com/unix_commands/chmod.htm. Acessado em 19/01/2021.

Serviços em linux

Em Linux, serviços são "programas" especiais, conhecidos como *daemons*, que rodam em segundo plano e sem controle direto de um usuário. Eles realizam certas ações em horários predefinidos ou em resposta a certos eventos. Por convenção, muitos *daemons* terminam com a letra 'd', como inetd, httpd, nfsd. Para gerenciar os serviços, usa-se o programa `service`. A seguir, alguns usos de `service`:

1. `service --status-all`. Lista todos os serviços.
2. `service serviço`. Lista os comandos disponíveis para o `serviço`.
3. `service serviço comando`. Executa `comando` no `serviço`. Os principais comandos são `start` para iniciar o serviço, `stop` para encerar o serviço, `restart` para reiniciar o serviço e `status` para ver o estado atual do serviço.

Outro programa usado para gerenciar serviços é o `systemctl`. Ele possui mais recursos que `service`, mas também é um pouco mais complexo. Em `systemctl`, um serviço é dito habilitado (enabled), quando inicia com o sistema, e um serviço é dito ativo (active) quando está em execução. A seguir, alguns usos de `systemctl`:

1. `systemctl -a`. Lista todos os serviços habilitados, ativos ou não. Para navegar pelos resultados, pode usar os mesmos comandos de `less`.
2. `systemctl list-unit-files`. Lista todos os serviços instalados, habilitados ou não.
3. `systemctl comando serviço`. Executa `comando` no `serviço`. Observe que aqui a ordem é diferente do programa `service`. Os principais comandos são `start` (inicia), `stop` (encerra), `restart` (reinicia), `status` (exibe o estado atual), `show` (exibe as propriedades do serviço), `enable` (habilita inicialização com o sistema, precisa de permissão root), `disable` (desabilita inicialização com o sistema, precisa de permissão root).

O service não possui equivalentes ao enable e disable do systemctl. Para habilitar um serviço sem o systemctl, você pode: `sudo update-rc.d foo_service defaults`. Para desabilitar: `sudo update-rc.d foo_service remove`.

Fonte: https://askubuntu.com/a/192060/586845. Acessado em 17/01/2021.

O serviço ssh

Para acessar remotamente um computador, pode usar o protocolo ssh. Para isso, o computador a ser acessado precisa ter um servidor ssh rodando. Para transferir arquivos do computador local para o

remoto, digite `scp path/to/local/file username@ip:path/to/remote/file`, em que `username` é o nome do usuário e `ip` é o endereço ip do computador a ser acessado. Para saber o nome de usuário e ip de um computador, digite `whoami` e `ip a`, respectivamente. E para transferir do remoto para o local, só inverter a ordem. Por exemplo, o comando `scp fulano@192.168.0.100:/home/fulano/file.txt ~/Documentos/` transfere o arquivo `file.txt`, localizado no diretório padrão do computador remoto, para o diretório `Documentos` do computador local. Para acessar remotamente o terminal de outro computador, digite `ssh username@ip`.

Caso queira instalar um servidor ssh em seu computador, digite `sudo apt install openssh-server`. Agora seu computador pode ser acessado remotamente. Para o servidor ssh não iniciar com o sistema, digite `sudo systemctl disable ssh`. Para iniciar ou encerrar o servidor ssh manualmente, digite `systemctl start ssh` ou `systemctl stop ssh`.

Como forçar a interrupção de um programa

Fonte: https://www.booleanworld.com/kill-process-linux/. Acessado em 21/04/2021.

Qualquer programa, serviço, software, que esteja rodando no Linux, é chamado de processo. Cada processo possui um identificador exclusivo (ID) e um nome. Você pode usar o ID ou nome para encerrar um processo, por exemplo, quando ele travou ou está ocupando toda tua memória RAM.

Para listar todos os processos atualmente sendo executados, digite o comando `ps aux`. Para filtrar o processo, você pode usar o grep. Por exemplo, para listar qualquer processo que tenha “fire” no nome, digite `ps aux | grep fire`.

Para encerrar um processo, digite o comando `killall nome_do_processo`. Por exemplo, `killall firefox` encerrará o firefox. Para encerrar forçadamente um processo, digite o comando `killall -SIGKILL nome_do_processo`. Para encerrar um processo pelo seu ID, digite `kill id_do_processo`, com ou sem `-SIGKILL`. Se você sabe apenas parte do nome do processo, você pode usar o comando `pkill parte_do_nome`, por exemplo, `pkill fire` encerrará o firefox.

Terminais virtuais

Fonte: https://ostechnix.com/how-to-switch-between-ttys-without-using-function-keys-in-linux/. Acessado em 21/04/2021.

O Linux possui terminais virtuais, chamados tty, que são terminais executados em telas somente texto. Se você estiver no modo gráfico e quiser ir para o modo texto, tecle os atalhos [Ctrl]+[Alt]+[Fn], sendo n um número entre 3 e 6, que corresponde a um terminal virtual específico. Ao acessar um tty, é necessário informar o nome de usuário e senha. Para voltar ao modo gráfico, tecle o atalho [Ctrl]+[Alt]+[F2].

Quando for o caso de um processo travar o computador no modo gráfico, você pode acessar um terminal no modo texto e encerrar o processo usando o comando `killall` como na seção anterior.

Extra - Compilado de comandos interessantes

Baixar um site inteiro recursivamente

```
wget --recursive --no-clobber --page-requisites --html-extension --convert-links --restrict-file-names=windows --no-parent --domains website.org www.website.org/tutorials/html/
```

Esse comando baixa tudo que estiver dentro de website.org/tutorials/html. Os argumentos são: `--recursive`: download the entire Web site. `--domains website.org`: don't follow links outside website.org. `--no-parent`: don't follow links outside the directory tutorials/html/. `--page-requisites`: get all the elements that compose the page (images, CSS and so on). `--html-extension`: save files with the .html extension. `--convert-links`: convert links so that they work locally, off-line. `--restrict-file-names=windows`: modify filenames so that they will work in Windows as well. `--no-clobber`: don't overwrite any existing files (used in case the download is interrupted and resumed).

Acessar uma pasta remota de Linux em Linux

Você pode montar uma pasta de um Linux remoto no seu Linux local usando um explorador de arquivos (como o nautilius ou o nemo), e assim, conseguir abrir a pasta usando sua IDE preferida. Acesse o endereço `sftp://10.8.6.141/home/usuario` pela barra de endereços do teu explorador de arquivos para montar em seu Linux local a pasta `/home/usuario` do Linux remoto localizado no endereço `10.8.6.141`.

Alternativa ao sftp para montar uma pasta remota numa pasta local é executar o comando `sshfs pi@10.8.6.141:/home/usuario /home/usuariolocal/pastaremota/`. Aqui montamos a pasta `/home/usuario` do Linux remoto na pasta `/home/usuariolocal/pastaremota/` do Linux local. Para desmontar a pasta: `fusermount -u /home/usuariolocal/pastaremota/`.

Acessar uma pasta remota de Windows em Linux

Para acessar uma pasta compartilhada pelo windows:

```
sudo mount -t cifs //[host-name-or-ip]/[shared-folder] [mount-point]/ --verbose -o user=[user-name]
```

Onde [host-name-or-ip] é o nome ou IP da máquina remota, [shared-folder] é o nome ou caminho da pasta compartilhada na máquina remota, [mount-point] é onde a pasta será montada na máquina local, e [user-name] é o nome de usuário com permissão de acesso à pasta compartilhada da máquina remota. Se der um erro, indicando que a versão do smb sendo utilizada pode não ser suportada, tente assim:

```
sudo mount -t cifs //[host-name-or-ip]/[shared-folder] [mount-point]/ --verbose -o vers=2.1,user=[user-name]
```

Para desmontar: `sudo umount [mount-point]`. Para mais informações, acesse aqui.

A maioria dos exploradores de arquivos permitem acessar pastas do Windows remotamente colocando o protocolo `smb://` antes do caminho até a pasta, e também permitem compartilhar pastas com o Windows também, clicando abrindo seu menu de contexto. Para isso você precisa ter o Samba instalado (`sudo apt install samba`). Após compartilhar uma pasta, crie um usuário

samba que consiga acessá-la de outros computadores com `sudo smbpasswd -a nome_do_usuário`, substituindo `nome_do_usuário` por um nome desejado.

Montar um drive (pendrive ou hd externo)

Para listar todos os drives disponíveis: `sudo fdisk -l`.

Para montar um drive formatado como exfat: `sudo mount.exfat-fuse /dev/sdb1 /media/exfat`., onde /dev/sdb1 é a localização do drive e /media/exfat é o caminho onde o drive será montado.

Para montar um drive formatado como fat ou ntfs, você pode usar o udisk, que não exige sudo: udisksctl mount -b /dev/sdc1, supondo que seu drive está em /dev/sdc1.

Para desmontar um drive: sudo umount /caminho/drive/montado.

Comando sed

O programa sed faz buscas e substituições em arquivos. O comando `sed -i '/^[^#]/ s/^/# /' arquivo.txt` coloca um `# ` no início de toda linha que não começa com `#`. Explicação: `-i` faz a substituição diretamente no arquivo (in-place), sem gerar uma saída. Tudo que vai dentro das aspas simples é o que será analisado/feito. O arquivo `arquivo.txt` é de onde a análise será feita. A primeira parte, `/^[^#]/`, é o que estamos procurando. O primeiro circunflexo indica início de linha, e o que vai dentro dos colchetes indica um padrão a ser procurado, no caso, `^#` indica aquilo que não é cerquilha, já que o circunflexo dentro dos colchetes é negação. Ou seja, estamos procurando por qualquer linha cujo início não comece com cerquilha. A segunda parte, `s/^/# /`, é o que será substituído pelo quê. O `s` indica que faremos uma substituição. O `^` indique que o início da linha que será substituído. E o `# ` indica pelo quê será substituído.

Outro exemplo, suponha que temos um arquivo contendo apenas uma linhazona, e que essa linhazona possui vários caracteres '\' seguidos por 'n', que podem ser interpretados como quebra de linha. O comando `sed -i 's/\\n/\n/g' arquivo.txt`. Substitui esses caracteres por uma quebra de linha de fato. Explicação: diferente do primeiro exemplo, já começamos com o `s`, que é substituição, isto é, não estamos filtrando linhas em específico. Como a barra invertida é um caractere especial, precisamos escapá-la, isto é, tratá-la como uma barra invertida de fato, por isso colocamos duas barras invertidas em `/\\n`. Depois fazemos a substituição do que encontramos por uma quebra de linha de fato com `/\n`. Por fim, `/g` indica global, isto é, fazer a substituição em todas as ocorrências, e não apenas na primeira ocorrência.

Comando xrandr

O comando xrandr configura a resolução de saída do monitor. Quando se tem dois monitores de resoluções diferentes, e quer espelhar a saída em ambos, é melhor manter a resolução do monitor de apresentação como a mesma para ambos: `xrandr --output LVDS-1 --mode 1024x768 --output HDMI-1 --same-as LVDS-1 --mode 1024x768`. Nesse exemplo, LVDS-1 é o monitor do meu notebook, que possui resolução 1366x768, mas optei por deixar a resolução 1024x768, que é a nativa do HDMI-1, onde está ligado o projetor. A opção --same-as indica que é para espelhar no HDMI-1 o que está no LVDS-1. Agora, quando não se quer espelhar, então pode

deixar a resolução nativa de cada monitor: `xrandr --output LVDS-1 --mode 1366x768 --output HDMI-1 --mode 1024x768 --right-of LVDS-1`.

Aplicativo padrão para tipos de arquivo

Para saber o tipo de determinado arquivo: `mimetype arquivo`. Por exemplo, o comando `mimetype foo.txt` gera a saída `text/plain`, indicando que `foo.txt` é um arquivo de texto simples. Para saber todos os aplicativos que abrem determinado arquivo: `mimeopen -a foo.txt`, sendo que o primeiro da lista é o padrão para abrir o arquivo. Para definir um novo aplicativo padrão para o tipo de determinado arquivo: `mimeopen -d foo.txt`. Para abrir um arquivo com seu aplicativo padrão: `mimeopen foo.txt`.

Redirecionar saída de comando para uma variável

Você pode atribuir a saída de um comando a uma variável usando substituição de comando. Por exemplo, para salvar o conteúdo de um arquivo em uma variável: `foo=$(cat bar.txt)`.

Compartilhar pasta no virtualbox linux convidado

Instale os adicionais para convidado, clicando no menu “Devices → Insert Guest Addition CD image”, abrindo o CD, e executando o programa `VBoxLinuxAdditions.run`. Compartilhe uma pasta clicando no menu “Devices → Shared folders → Shared filders settings → Adds new shared folder”. Selecione uma pasta do sistema hospedeiro a ser compartilhada, dê um nome à pasta (pode ser o padrão), deixe o ponto de montagem em branco para o virtualbox escolher um automaticamente, e marque as opções “auto-mount’ e “make permanent”. Adicione-se ao grupo `vboxsf` no linux convidado, para ter permissão de acesso sem root à pasta compartilhada: `sudo usermod --append --groups vboxsf $USER`. Reinicie o linux convidado.

Rofi

A window switcher, application launcher, ssh dialog and dmenu replacement.

As opções mais selecionadas ficam no topo da lista. Caso queira remover alguma opção do topo da lista, tecle shift+delete.

À medida que você digita algo, ele vai filtrando. Para apagar o que digitou, tecle ctrl+w.

Quando cada linha é muito grande e não cabe na tela, você pode apertar [alt]+[.] para alternar em ver seu final ou início.

Para ver outros atalhos do rofi: `rofi -show keys`.

Xclip

O `xclip` controla por linha de comando a área de transferência (clipboard). Para instalá-lo, execute `sudo apt install xclip`.

Canalize a saída de um programa para o clipboard: `cat file | xclip -se c`. A opção `-se c` diz para usar (`-se`) o clipboard normal (`c`) (o linux tem outros clipboards…).

Cole o clipboard na saída padrão: `xclip -se c -o`.

Você pode salvar uma imagem do clipboard para um arquivo: `xclip -se c -t image/png -o >temp.png`.

O xclip em conjunto com o base64 é legal para codificar uma imagem em texto: `xclip -se c -t image/png -o | base64 -w0`. Esse comando converte uma imagem do clipboard para base64, e copia o base64 da imagem para o clipboard.

Quando quero colocar uma imagem num arquivo HTML, ao invés de salvar a imagem, coloco sua base64 diretamente na tag img:

```
{ echo "<figure><img src='" ; xclip -se c -t image/png -o | base64 -w0 ; echo -e "' alt=''>\n<figcaption>Figura </figcaption>\n</figure>"; } | xclip -se c
```

O comando rsync

Para adicionar/atualizar arquivos de `source` em `destiny`: `rsync -avP source/ destiny`.

Se não colocar barra depois de `source`, este também será copiado para `destiny`. `-a` sincroniza recursivamente, preservando links simbólicos arquivos especiais, metadados como última modificação e permissões. `-v` para modo verboso. `-P` mostra o progresso e retoma de onde parou. Se estiver sincronizando via rede, pode acrescentar `-z` para comprimir os arquivos antes do envio, e, assim, reduzir o consumo de banda. Por padrão, arquivos de `destiny` não presentes em `source` são mantidos. Para excluí-los, acrescente `--delete`. Exemplo, para sincronizar exatamente o conteúdo de uma pasta remota para uma pasta local: `rsync -avzP --delete username@remote_host:/home/username/dir1/ ~/backup`.

Se apenas quiser ver o que será alterado, sem de fato alterar nada, acrescente `-n`. Para ignorar algumas pastas ou arquivos da sincronização, acrescente `--exclude=pattern_to_exclude`. Para incluir algumas pastas ou arquivos de uma pasta ignorada, acrescente `--include=pattern_to_include`. Exemplo, para apenas ver o que será sincronizado de `src` em `dest`, ignorando as pastas `a` e `b`, mas incluindo `a/c.txt`: `rsync -anv --exclude='a/' --exclude='b/' --include='a/c.txt' src/ dest`.

Referências

https://www.howtogeek.com/140679/beginner-geek-how-to-start-using-the-linux-terminal/. Acessado em 13/01/2021.

http://linuxcommand.org/lc3_learning_the_shell.php#contents. Acessado em 13/01/2021.

https://www.howtogeek.com/howto/ubuntu/keyboard-shortcuts-for-bash-command-shell-for-ubuntu-debian-suse-redhat-linux-etc/. Acessado em 13/01/2021.

https://www.computerhope.com/unix/udiff.htm. Acessado em 14/01/2021.

https://unix.stackexchange.com/a/159514/366802. Acessado em 14/01/2021.

https://stackoverflow.com/a/33930536/4072641. Acessado em 15/01/2021.