Instituto Federal de Brasília
Campus Brasília
Componente: Administração de Serviços para Internet

Habilidades em Linha de Comando

1 Introdução

Se você é como a maioria das pessoas, provavelmente está mais familiarizado com o uso de uma interface gráfica do usuário (GUI) para controlar seu computador. Introduzida às massas pela Apple no computador Macintosh e popularizada pela Microsoft, uma GUI fornece uma maneira fácil e detectável de gerenciar seu sistema. Sem uma GUI, algumas ferramentas para gráficos e vídeos não seriam práticas. Antes da popularidade da GUI, a interface de linha de comando (CLI) era a maneira preferida de controlar um computador. O CLI depende exclusivamente da entrada do teclado. Tudo o que você quer que o computador faça é retransmitido digitando comandos em vez de clicar nos ícones.

Se você nunca usou um CLI, a princípio pode ser um desafio porque requer memorizar comandos e suas opções. No entanto, uma CLI fornece controle mais preciso, maior velocidade e a capacidade de automatizar facilmente tarefas por meio de scripts. Embora o Linux tenha muitos ambientes GUI, você poderá controlar o Linux de maneira muito mais eficiente usando a Interface da Linha de Comandos.

Figura 1: Importância da Linha de Comando

2 Interface de Linha de Comando (CLI)

A interface de linha de comando (*Command Line Interface, CLI*) é uma interface baseada em texto para o computador, onde o usuário digita um comando e o computador o executa. O ambiente da CLI é fornecido por um aplicativo no computador conhecido como terminal.

O terminal aceita o que o usuário digita e passa para um shell. O shell interpreta o que o usuário digitou em instruções que podem ser executadas pelo sistema operacional. Se uma saída for produzida pelo comando, este texto será exibido no terminal. Se forem encontrados problemas na execução do comando, uma mensagem de erro será exibida.

3 Acessando um Terminal

Existem muitas maneiras de acessar uma janela de terminal. Alguns sistemas serão inicializados diretamente em um terminal. Esse é frequentemente o caso dos servidores, já que a interface gráfica do usuário (GUI) pode exigir muitos recursos e pode não ser necessária para executar operações baseadas em servidor.

Um bom exemplo de um servidor que não requer necessariamente uma GUI é um servidor Web. Os servidores da Web precisam ser executados o mais rápido possível e uma GUI apenas atrasaria o sistema.

Em sistemas que inicializam em uma GUI, geralmente há duas maneiras de acessar um terminal, um terminal baseado em GUI e um terminal virtual:

- Um terminal GUI é um programa dentro do ambiente GUI que emula uma janela de terminal. Terminais GUI podem ser acessados através do sistema de menus. Por exemplo, em uma máquina do CentOS, você pode clicar em Aplicativos na barra de menus, depois em Ferramentas do Sistema> e, finalmente, Terminal (figura 2).
- Um terminal virtual pode ser executado ao mesmo tempo que uma GUI, mas requer que o usuário faça o login através do terminal virtual antes de poder executar comandos (como fariam antes de acessar a interface GUI). A maioria dos sistemas tem vários terminais virtuais que podem ser acessados pressionando-se uma combinação de teclas, por exemplo: Ctrl-Alt-F1

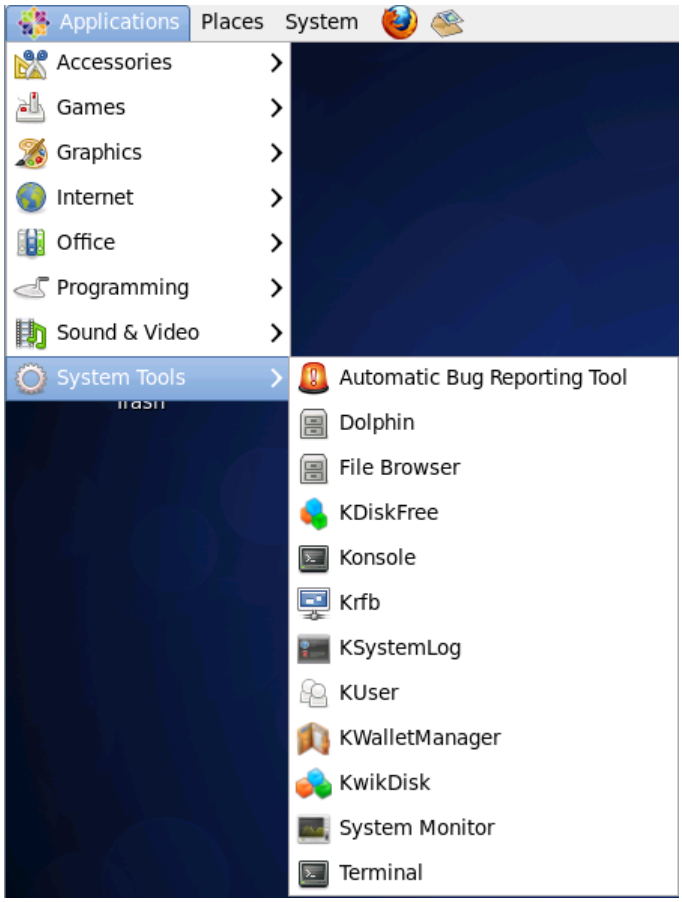

Figura 2: Exemplo de acesso a um terminal GUI no CentOS

Nota: Em máquinas virtuais, terminais virtuais podem não estar disponíveis.

3.1 Prompt

Uma janela de terminal exibe um prompt; o prompt aparece quando nenhum comando está sendo executado e quando toda a saída do comando foi impressa na tela. O prompt é projetado para dizer ao usuário para inserir um comando.

A estrutura do prompt pode variar entre distribuições, mas normalmente contém informações sobre o usuário e o sistema. Abaixo está uma estrutura de prompt comum:

```
sysadmin@localhost:~$
```

O prompt anterior fornece o nome do usuário que efetuou login (`sysadmin`), o nome do sistema (`localhost`) e o diretório atual (`~`). O símbolo `~` é usado como atalho para o diretório pessoal do usuário (geralmente, o diretório inicial do usuário está no diretório `/home` e é nomeado após o nome da conta do usuário, por exemplo: `/home/sysadmin`).

3.2 Shell

Um shell é o interpretador que traduz os comandos inseridos por um usuário em ações a serem executadas pelo sistema operacional. O ambiente Linux oferece muitos tipos diferentes de shells, alguns dos quais existem há muitos anos.

O shell mais usado para distribuições Linux é chamado de shell BASH. É um shell que fornece muitos recursos avançados, como o histórico de comandos, que permite que você reexecute facilmente os comandos executados anteriormente.

O shell BASH também possui outros recursos populares:

- **Scripting**: A habilidade de colocar comandos em um arquivo e executar o arquivo, resultando em todos os comandos sendo executados. Esse recurso também possui alguns recursos de programação, como instruções condicionais e a capacidade de criar funções (AKA, sub-rotinas).
- **Aliases**: A capacidade de criar "apelidos"curtos para comandos mais longos.
- **Variáveis**: Variáveis são usadas para armazenar informações para o shell BASH. Essas variáveis podem ser usadas para modificar como os comandos e recursos funcionam, além de fornecer informações vitais do sistema.

Nota: A lista anterior é apenas um breve resumo de alguns dos muitos recursos fornecidos pelo shell BASH.

3.3 Formatando Comandos

Muitos comandos podem ser usados sozinhos sem mais informações. Alguns comandos requerem entrada adicional para serem executados corretamente. Esta entrada adicional vem em duas formas: opções e argumentos.

O formato típico de um comando é o seguinte:

```
command [options] [arguments]
```

> Tenha em mente que o Linux faz distinção entre maiúsculas e minúsculas. Comandos, opções, argumentos, variáveis e nomes de arquivos devem ser digitados exatamente como mostrado.

O comando **ls** fornecerá exemplos úteis. Por si só, o comando **ls** listará os arquivos e diretórios contidos em seu diretório de trabalho atual:

```
sysadmin@localhost:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
sysadmin@localhost:~$
```

> O comando **ls** será coberto com detalhes completos em um capítulo posterior. O objetivo de introduzir esse comando agora é demonstrar como os argumentos e as opções funcionam. Neste ponto, você não deve se preocupar com a saída do comando, mas sim em entender o que é um argumento e uma opção.

Um argumento também pode ser passado para o comando **ls** para especificar qual diretório deve-se listar o conteúdo. Por exemplo, o comando **ls /etc/ppp** listará o conteúdo do diretório **/etc/ppp** em vez do diretório atual:

```
sysadmin@localhost:~$ ls /etc/ppp
ip-down.d  ip-up.d
sysadmin@localhost:~$
```

Como o comando **ls** aceitará vários argumentos, você poderá listar o conteúdo de vários diretórios de uma só vez digitando o comando **ls /etc/ppp /etc/sssh**:

```
sysadmin@localhost:~$ ls /etc/ppp /etc/ssh
/etc/ppp:
ip-down.d  ip-up.d
/etc/ssh:
moduli                ssh_host_dsa_key.pub     ssh_host_rsa_key      ss
hd_configssh_config
ssh_host_ecdsa_key    ssh_host_rsa_key.pub
ssh_host_dsa_key      ssh_host_ecdsa_key.pub   ssh_import_id
sysadmin@localhost:~$
```

3.4 Trabalhando com opções

Opções podem ser usadas com comandos para expandir ou modificar a maneira como um comando se comporta. As opções geralmente são letras únicas; no entanto, às vezes eles serão "palavras"também. Normalmente, os comandos mais antigos usam letras únicas enquanto os comandos mais novos usam palavras completas para as opções. Opções de letra única são precedidas por um único traço -. Opções de palavras completas são precedidas por dois traços –.

Por exemplo, você pode usar a opção **-l** com o comando **ls** para exibir mais informações sobre os arquivos listados. O comando **ls -l** listará os arquivos contidos no diretório atual e fornecerá informações adicionais, como as permissões, o tamanho do arquivo e outras informações:

```
sysadmin@localhost:~$ ls -l
total 0
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29  2015 Desktop
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29  2015 Documents
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29  2015 Downloads
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29  2015 Music
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29  2015 Pictures
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29  2015 Public
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29  2015 Templates
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29  2015 Videos
sysadmin@localhost:~$
```

Na maioria dos casos, as opções podem ser usadas em conjunto com outras opções. Por exemplo, o comando **ls -l -h** ou **ls -lh** listará os arquivos com detalhes, mas exibirá os tamanhos dos arquivos em formato legível em vez do valor padrão (bytes):

```
sysadmin@localhost:~$ ls -l /usr/bin/perl

-rwxr-xr-x 2 root root 10376 Feb  4  2014 /usr/bin/perl
sysadmin@localhost:~$ ls -lh /usr/bin/perl
-rwxr-xr-x 2 root root 11K Feb  4  2014 /usr/bin/perl
sysadmin@localhost:~$
```

Observe que o exemplo anterior também demonstrou como você pode combinar opções de letra única: **-lh** -lh. A ordem das opções combinadas não é importante.

A opção **-h**-h também possui uma forma de palavra completa: **–human-readable**.

Opções muitas vezes podem ser usadas com um argumento. De fato, algumas opções requerem seus próprios argumentos. Você pode usar opções e argumentos com o comando **ls** para listar o conteúdo de outro diretório executando o comando **ls -l /etc/ppp**:

```
sysadmin@localhost:~$ ls -l /etc/ppp
total 0
drwxr-xr-x 1 root root 10 Jan 29  2015 ip-down.d
drwxr-xr-x 1 root root 10 Jan 29  2015 ip-up.d
sysadmin@localhost:~$
```

4 História de comando

Quando você executa um comando em um terminal, o comando é armazenado em uma "lista de histórico". Isso foi projetado para facilitar a execução do mesmo comando mais tarde, pois você não precisará digitar novamente o comando inteiro.

Para ver a lista de histórico de um terminal, use o comando **history**:

```
sysadmin@localhost:~$ date
Sun Nov  1 00:40:28 UTC 2015
sysadmin@localhost:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates
Videos
sysadmin@localhost:~$ cal 5 2015
    May 2015
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
sysadmin@localhost:~$ history
    1  date
    2  ls
    3  cal 5 2015
    4  history
sysadmin@localhost:~$
```

Presssionando a tecla **seta para cima** ↑ será exibido o comando anterior digitado no prompt. Você pode pressionar essa tecla repetidamente para voltar no histórico de comandos até o comando que deseja executar. Pressionando a tecla **Enter** será executado novamente o comando exibido.

Quando você encontrar o comando que deseja executar, poderá usar as teclas de **seta para a esquerda** ← e **seta para a direita** → para posicionar o cursor para edição. Outras teclas úteis para edição incluem as teclas Home, End, Backspace e Delete.

Se você vir um comando que deseja executar na lista que o comando history gera, você pode executar esse comando digitando um ponto de exclamação e, em seguida, o número ao lado do comando, por exemplo:

```
!3
```

```
sysadmin@localhost:~$ history
    1  date
    2  ls
    3  cal 5 2015
    4  history
sysadmin@localhost:~$ !3
cal 5 2015
     May 2015
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
                  31
sysadmin@localhost:~$
```

Alguns exemplos adicionais de **history**:

Example	Meaning
`history 5`	Show the last five commands from the history list
`!!`	Execute the last command again
`!-5`	Execute the fifth command from the bottom of the history list
`!ls`	Execute the most recent `ls` command

5 Introduzindo variáveis de shell BASH

Uma variável de shell BASH é um recurso que permite que você ou o shell armazenem dados. Esses dados podem ser usados para fornecer informações críticas do sistema ou para alterar o comportamento de como o shell BASH (ou outros comandos) funcionem.

As variáveis recebem nomes e são armazenadas temporariamente na memória. Quando você fecha uma janela de terminal ou shell, todas as variáveis são perdidas. No entanto, o sistema recria automaticamente muitas dessas variáveis quando um novo shell é aberto.

Para exibir o valor de uma variável, você pode usar o comando **echo**. O comando **echo** é usado para exibir a saída no terminal; no exemplo abaixo, o comando exibirá o valor da variável `HISTSIZE`:

```
sysadmin@localhost:~$ echo $HISTSIZE
1000
sysadmin@localhost:~$
```

A variável `HISTSIZE` define quantos comandos anteriores para armazenar na lista de histórico. Para exibir o valor da variável, use um caractere $ do cifrão antes do nome da variável. Para modificar o valor da variável, você não usa o caractere $:

```
sysadmin@localhost:~$ HISTSIZE=500
sysadmin@localhost:~$ echo $HISTSIZE
500
sysadmin@localhost:~$
```

> Existem muitas variáveis de shell disponíveis para o shell BASH, assim como variáveis que afetarão diferentes comandos do Linux. Uma discussão de todas as variáveis do shell está além do escopo desta aula, no entanto, mais variáveis do shell serão abordadas à medida que este curso progrida.

6 Variável PATH

Uma das variáveis mais importantes do shell BASH para entender é a variável `PATH`.

O termo *path* (caminho) refere-se a uma lista que define quais diretórios o shell procurará para encontrar os comandos. Se você digitar um comando e receber um erro "comando não encontrado", é porque o shell BASH não conseguiu localizar um comando com esse nome em nenhum dos diretórios incluídos no *path*. O comando a seguir exibe o caminho do shell atual:

```
sysadmin@localhost:~$ echo $PATH
/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/
sbin:/bin:
/usr/games
sysadmin@localhost:~$
```

Com base na saída procedente, quando você tentar executar um comando, o shell procurará primeiro o comando no diretório `/home/sysadmin/bin`. Se o comando for encontrado nesse diretório, ele será executado. Se não for encontrado, o shell irá procurar no diretório `/usr/local/sbin`.

Se o comando não for encontrado em nenhum diretório listado na variável `PATH`, você receberá um erro de comando não encontrado (`comand not found`):

```
sysadmin@localhost:~$ zed
-bash: zed: command not found
sysadmin@localhost:~$
```

Se um software personalizado estiver instalado em seu sistema, talvez seja necessário modificar o PATH para facilitar a execução desses comandos. Por exemplo, o seguinte incluirá o diretório `/usr/bin/custom` na variável `PATH`:

```
sysadmin@localhost:~$ PATH=/usr/bin/custom:$PATH
sysadmin@localhost:~$ echo $PATH
/usr/bin/custom:/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr
/sbin:/usr/bin:/sbin:/bin:/usr/games
sysadmin@localhost:~$
```

7 Comando export

Existem dois tipos de variáveis usadas no shell BASH, local e ambiente. Variáveis de ambiente, como `PATH` e `HOME`, são usadas pelo BASH ao interpretar comandos e executar tarefas. Variáveis locais são frequentemente associadas a tarefas baseadas em usuários e são minúsculas por convenção. Para criar uma variável local, basta digitar:

```
sysadmin@localhost:~$ variable1='Something'
```

Para ver o conteúdo da variável, consulte-a com um sinal inicial $:

```
sysadmin@localhost:~$ echo $variable1
Something
```

Para visualizar as variáveis de ambiente, use o comando **env** (a pesquisa através da saída usando **grep**, como mostrado aqui, será discutida em aulas posteriores). Nesse caso, a pesquisa por variável1 nas variáveis de ambiente resulta em nenhuma saída:

```
sysadmin@localhost:~$ env | grep variable1
sysadmin@localhost:~$
```

Depois de exportar a `variável1`, agora é uma variável de ambiente. Observe que, desta vez, ele é encontrado na pesquisa por meio das variáveis de ambiente:

```
sysadmin@localhost:~$ export variable1
sysadmin@localhost:~$ env | grep variable1
variable1=Something
```

O comando de **export** também pode ser usado para criar uma variável de ambiente na sua criação:

```
sysadmin@localhost:~$ export variable2='Else'
sysadmin@localhost:~$ env | grep variable2
variable2=Else
```

Para alterar o valor de uma variável de ambiente, simplesmente omita o $ ao fazer referência a ele:

```
sysadmin@localhost:~$ variable1=$variable1' '$variable2
sysadmin@localhost:~$ echo $variable1
Something Else
```

Variáveis exportadas podem ser removidas usando o comando **unset**:

```
sysadmin@localhost:~$ unset variable2
```

8 Comando which

Pode haver situações em que diferentes versões do mesmo comando estejam instaladas em um sistema ou em que os comandos sejam acessíveis a alguns usuários e não a outros. Se um comando não se comportar como esperado ou se um comando não estiver acessível, deve ser benéfico saber onde o shell está localizando o comando ou qual versão ele está usando.

Seria tedioso ter que procurar manualmente em cada diretório listado na variável `PATH`. Em vez disso, você pode usar o comando **which** para exibir o caminho completo para o comando em questão:

```
sysadmin@localhost:~$ which date
/bin/date
sysadmin@localhost:~$ which cal


/usr/bin/cal
sysadmin@localhost:~$
```

O comando **which** pesquisa a localização de um comando pesquisando a variável `PATH`.

9 Comando type

O comando **type** pode ser usado para determinar informações sobre vários comandos. Alguns comandos são originários de um arquivo específico:

```
sysadmin@localhost:~$ type which
which is hashed (/usr/bin/which)
```

Essa saída seria semelhante à saída do comando **which** (como discutido na seção anterior, que exibe o caminho completo do comando):

```
sysadmin@localhost:~$ which which
/usr/bin/which
```

O comando **type** também pode identificar comandos que são construídos no shell bash (ou outro):

```
sysadmin@localhost:~$ type echo
echo is a shell builtin
```

Nesse caso, a saída é significativamente diferente da saída do comando which:

```
sysadmin@localhost:~$ which echo
/bin/echo
```

Usando a opção **-a**, o comando **type** também pode revelar o caminho de outro comando:

```
sysadmin@localhost:~$ type -a echo
echo is a shell builtin
echo is /bin/echo
```

O comando **type** também pode identificar aliases para outros comandos:

```
sysadmin@localhost:~$ type ll
ll is aliased to `ls -alF'
sysadmin@localhost:~$ type ls
ls is aliased to `ls --color=auto'
```

A saída desses comandos indica que **ll** é um alias para **ls -alF** e até **ls** é um alias para **ls –color = auto**. Novamente, a saída é significativamente diferente do comando **which**:

```
sysadmin@localhost:~$ which ll
sysadmin@localhost:~$ which ls
/bin/ls
```

O comando **type** suporta outras opções e pode pesquisar vários comandos simultaneamente. Para exibir apenas uma única palavra descrevendo os comandos **echo**, **ll** e **which**, use a opção **-t**:

```
sysadmin@localhost:~$ type -t echo ll which
builtin
alias
file
```

10 Aliases

Um *alias* pode ser usado para mapear comandos mais longos para sequências de chaves mais curtas. Quando o shell vê um alias sendo executado, ele substitui a sequência mais longa antes de continuar a interpretar os comandos.

Por exemplo, o comando **ls -l** é comumente usado abreviado para um alias como **l** ou **ll**. Como esses comandos menores são mais fáceis de digitar, torna-se mais rápido executar a linha de comando **ls -l**.

Você pode determinar quais aliases estão configurados no seu shell com o comando **alias**:

```
sysadmin@localhost:~$ alias
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
```

Os aliases que você vê nos exemplos anteriores foram criados pelos arquivos de inicialização. Esses arquivos são projetados para tornar o processo de criação automática de aliases e serão discutidos em mais detalhes em uma aula posterior.

Novos aliases podem ser criados digitando-se `alias nome = comando`, em que nome é o nome que você deseja fornecer ao alias e o comando é o comando que você deseja executar ao executar o alias.

Por exemplo, você pode criar um alias para que **lh** exiba uma longa lista de arquivos, classificados por tamanho com um tamanho "human friendly" com o comando **alias lh = 'ls -Shl'**. Digitar **lh** agora deve resultar na mesma saída que digitar o comando **ls -Shl**:

```
sysadmin@localhost:~$ alias lh='ls -Shl'
sysadmin@localhost:~$ lh /etc/ppp
total 0
drwxr-xr-x 1 root root 10 Jan 29  2015 ip-down.d
drwxr-xr-x 1 root root 10 Jan 29  2015 ip-up.d
```

Os alias criados dessa maneira persistirão apenas enquanto o shell estiver aberto. Depois que o shell é fechado, os novos aliases que você criou serão perdidos. Além disso, cada shell tem seus próprios aliases, portanto, se você criar um alias em um shell e depois abrir outro shell, não verá o alias no novo shell.

11 Globbing

Os caracteres Glob são geralmente chamados de ”curingas”. Estes são símbolos que possuem um significado especial para o shell.

Ao contrário dos comandos que o shell executará, ou das opções e argumentos que o shell passará para comandos, os caracteres glob são interpretados pelo próprio shell antes de tentar executar qualquer comando. Isso significa que os caracteres glob podem ser usados com qualquer comando.

Os globs são poderosos porque permitem especificar padrões que correspondem a nomes de arquivos em um diretório, de modo que, em vez de manipular um único arquivo por vez, você pode facilmente executar comandos que afetarão muitos arquivos. Por exemplo, usando caracteres glob é possível manipular todos os arquivos com uma certa extensão ou com um comprimento de nome de arquivo específico.

Tenha em mente que esses globs podem ser usados com qualquer comando, porque é o shell, não o comando, que se expande com globs em nomes de arquivos correspondentes. Os exemplos fornecidos neste capítulo usarão o comando **echo** para demonstração.

11.1 Asterisco (*)

O caractere asterisco é usado para representar zero ou mais de qualquer caractere em um nome de arquivo. Por exemplo, suponha que você queira exibir todos os arquivos no diretório `/etc` que começam com a letra `t`:

```
sysadmin@localhost:~$ echo /etc/t*
/etc/terminfo /etc/timezone
sysadmin@localhost:~$
```

O padrão `t*` significa "corresponde a qualquer arquivo que comece com o caractere `t` e tenha zero ou mais de qualquer caractere após o `t`".

Você pode usar o caractere asterisco em qualquer lugar dentro do padrão de nome de arquivo. Por exemplo, o seguinte corresponderá a qualquer nome de arquivo no diretório `/etc` que termine com `.d`:

```
sysadmin@localhost:~$ echo /etc/*.d
/etc/apparmor.d /etc/bash_completion.d  /etc/cron.d /etc/depmod.d   /e
tc/fstab.d /etc/init.d /etc/insserv.conf.d /etc/ld.so.conf.d /etc/logr
otate.d /etc/modprobe.d /etc/pam.d /etc/profile.d /etc/rc0.d /etc/rc1.
d /etc/rc2.d /etc/rc3.d /etc/rc4.d /etc/rc5.d /etc/rc6.d /etc/rcS.d /e
tc/rsyslog.d /etc/sudoers.d /etc/sysctl.d  /etc/update-motd.d
```

No próximo exemplo, todos os arquivos no diretório `/etc` que começam com a letra `r` e terminam com `.conf` serão exibidos:

```
sysadmin@localhost:~$ echo /etc/r*.conf
/etc/resolv.conf /etc/rsyslog.conf
```

11.2 Ponto de Interrogação (?)

O ponto de interrogação representa qualquer caractere. Cada caractere de ponto de interrogação corresponde exatamente a um caractere, nem mais nem menos.

Suponha que você queira exibir todos os arquivos no diretório `/etc` que começam com a letra `t` e têm exatamente 7 caracteres após o caractere `t`:

```
sysadmin@localhost:~$ echo /etc/t???????
/etc/terminfo /etc/timezone
sysadmin@localhost:~$
```

Os caracteres Glob podem ser usados juntos para encontrar padrões ainda mais complexos. O comando **echo /etc/*????????????????????** irá imprimir apenas arquivos no diretório `/etc` com vinte ou mais caracteres no nome do arquivo:

```
sysadmin@localhost:~$ echo /etc/*????????????????????
/etc/bindresvport.blacklist /etc/ca-certificates.conf
sysadmin@localhost:~$
```

O asterisco e o ponto de interrogação também podem ser usados juntos para procurar arquivos com extensões de três letras executando o comando **echo /etc/*.???**:

```
sysadmin@localhost:~$ echo /etc/*.???
/etc/blkid.tab /etc/issue.net
sysadmin@localhost:~$
```

11.3 Colchetes []

Os colchetes são usados para corresponder a um único caractere, representando um intervalo de caracteres que são possíveis caracteres de correspondência. Por exemplo, **echo /etc/[gu]*** imprimirá qualquer arquivo que comece com um caractere g ou u e contenha zero ou mais caracteres adicionais:

```
sysadmin@localhost:~$ echo /etc/[gu]*
/etc/gai.conf /etc/groff /etc/group /etc/group- /etc/gshadow /etc/gsha
dow- /etc/ucf.conf /etc/udev /etc/ufw /etc/update-motd.d /etc/updatedb
.conf
sysadmin@localhost:~$
```

Os colchetes também podem ser usados para representar um intervalo de caracteres. Por exemplo, o comando **echo /etc/[a-d]*** imprimirá todos os arquivos que começam com qualquer letra entre e incluindo a e d:

```
sysadmin@localhost:~$ echo /etc/[a-d]*
/etc/adduser.conf /etc/adjtime /etc/alternatives /etc/apparmor.d
/etc/apt /etc/bash.bashrc /etc/bash_completion.d /etc/bind /etc/bindre
svport.blacklist /etc/blkid.conf /etc/blkid.tab /etc/ca-certificates /
etc/ca-certificates.conf /etc/calendar /etc/cron.d /etc/cron.daily /et
c/cron.hourly /etc/cron.monthly /etc/cron.weekly /etc/crontab /etc/dbu
s-1 /etc/debconf.conf /etc/debian_version /etc/default
/etc/deluser.conf /etc/depmod.d /etc/dpkg
sysadmin@localhost:~$
```

O comando **echo /etc/*[0-9]*** exibe qualquer arquivo que contenha pelo menos um número:

```
sysadmin@localhost:~$ echo /etc/*[0-9]*
/etc/dbus-1 /etc/iproute2 /etc/mke2fs.conf /etc/python2.7 /etc/rc0.d
/etc/rc1.d /etc/rc2.d /etc/rc3.d /etc/rc4.d /etc/rc5.d /etc/rc6.d
sysadmin@localhost:~$
```

O intervalo é baseado na tabela de texto ASCII. Esta tabela define uma lista de caracteres, organizando-os em uma ordem padrão específica. Se você fornecer um pedido inválido, nenhuma correspondência será feita:

```
sysadmin@localhost:~$ echo /etc/*[9-0]*
/etc/*[9-0]*
sysadmin@localhost:~$
```

11.4 Ponto de exclamação (!)

O ponto de exclamação é usado em conjunto com os colchetes para negar um intervalo. Por exemplo, o comando **echo [!DP]*** exibirá qualquer arquivo que não comece com D ou P.

12 Usando aspas

Existem três tipos de aspas que têm significado especial para o shell Bash: aspas duplas ", aspas simples ' e aspas simples invertida (ou crase) `. Cada conjunto de aspas indica ao shell que ele deve tratar o texto dentro das aspas de maneira diferente do que normalmente ser tratado.

12.1 Aspas duplas

Aspas duplas impedirão que o shell interprete alguns metacaracteres, incluindo caracteres glob. Dentro de aspas duplas, um asterisco é apenas um asterisco, um ponto de interrogação é apenas um ponto de interrogação e assim por diante. Isso significa que quando você usa o segundo **echo** abaixo, o shell BASH não converte o padrão glob em nomes de arquivos que correspondem ao padrão:

```
sysadmin@localhost:~$ echo /etc/[DP]*
/etc/DIR_COLORS /etc/DIR_COLORS.256color /etc/DIR_COLORS.lightbgcolor
/etc/PackageKit
sysadmin@localhost:~$ echo "/etc/[DP]*"
/etc/[DP]*
sysadmin@localhost:~$
```

Isso é útil quando você deseja exibir algo na tela que normalmente é um caractere especial para o shell:

```
sysadmin@localhost:~$ echo "The glob characters are *, ? and [ ]"
The glob characters are *, ? and [ ]
sysadmin@localhost:~$
```

Aspas duplas ainda permitem a substituição de comandos (discutida mais adiante neste capítulo), a substituição de variáveis e permitem alguns outros metacaracteres de shell que ainda não foram discutidos. Por exemplo, na demonstração a seguir, você notará que o valor da variável PATH é exibido:

```
sysadmin@localhost:~$ echo "The path is $PATH"
The path is /usr/bin/custom:/home/sysadmin/bin:/usr/local/sbin:/usr/lo
cal/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
sysadmin@localhost:~$
```

12.2 Aspas Simples

Aspas simples impedem que o shell faça qualquer interpretação de caracteres especiais. Isso inclui globs, variáveis, substituição de comando e outros metacaracteres que ainda não foram discutidos.

Por exemplo, se você quiser que o caractere $ signifique simplesmente um $, em vez de agir como um indicador para o shell para procurar o valor de uma variável, você pode executar o segundo comando exibido abaixo:

```
sysadmin@localhost:~$ echo The car costs $100
The car costs 00
sysadmin@localhost:~$ echo 'The car costs $100'
The car costs $100
sysadmin@localhost:~$
```

12.3 Caráter de barra invertida

Você pode usar uma técnica alternativa para basicamente citar um único caractere. Por exemplo, suponha que você queira imprimir o seguinte: "Os serviços custam $100 e o caminho é $PATH". Se você colocar isso entre aspas duplas, $1 e $PATH serão considerados variáveis. Se você colocar entre aspas simples, $1 e $PATH não são variáveis, mas e se você quiser ter $PATH tratada como uma variável e $1 não?

Se você colocar um caractere de barra invertida \ na frente de outro caractere, ele tratará o outro caractere como um caractere "com aspas simples". O terceiro comando abaixo demonstra o uso do caractere \, enquanto os outros dois demonstram como as variáveis seriam tratadas com aspas duplas e simples:

```
sysadmin@localhost:~$ echo "The service costs $100 and the path is $PA
TH"

The service costs 00 and the path is /usr/bin/custom:/home/sysadmin/bi
n:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/ga
mes

sysadmin@localhost:~$ echo 'The service costs $100 and the path is $PA
TH'

The service costs $100 and the path is $PATH

sysadmin@localhost:~$ echo The service costs \$100 and the path is $PA
TH

The service costs $100 and the path is /usr/bin/custom:/home/sysadmin/
bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/
games

sysadmin@localhost:~$
```

12.4 Aspas simples invertida

Aspas simples invertida (ou acento crase) são usadas para especificar um comando dentro de um comando, um processo chamado substituição de comando. Isso permite um uso muito poderoso e sofisticado de comandos.

Embora possa parecer confuso, um exemplo deve tornar as coisas mais claras. Para começar, observe a saída do comando **date**:

```
sysadmin@localhost:~$ date
Mon Nov  2 03:35:50 UTC 2015
```

Agora observe a saída do comando **echo Today is date**:

```
sysadmin@localhost:~$ echo Today is date
Today is date
sysadmin@localhost:~$
```

No comando anterior, a palavra `date` é tratada como texto regular e o shell simplesmente passa a `date` para o comando **echo**. Mas, você provavelmente quer executar o comando **date** e ter a saída desse comando enviada para o comando **echo**. Para conseguir isso, você executaria o comando **echo Today is `date`**:

```
sysadmin@localhost:~$ echo Today is `date`
Today is Mon Nov 2 03:40:04 UTC 2015
sysadmin@localhost:~$
```

13 Instruções de Controle

As instruções de controle permitem usar vários comandos ao mesmo tempo ou executar comandos adicionais, dependendo do sucesso de um comando anterior. Normalmente, essas instruções de controle são usadas em scripts, mas também podem ser usadas na linha de comando.

13.1 Ponto e vírgula

O ponto-e-vírgula pode ser usado para executar vários comandos, um após o outro. Cada comando é executado de forma independente e consecutiva; não importa o resultado do primeiro comando, o segundo será executado assim que o primeiro for concluído, o terceiro e assim por diante.

Por exemplo, se você quiser imprimir os meses de janeiro, fevereiro e março de 2015, poderá executar **cal 1 2015; cal 2 2015; cal 3 2015** na linha de comando:

```
sysadmin@localhost:~$ cal 1 2015; cal 2 2015; cal 3 2015
    January 2015
Su Mo Tu We Th Fr Sa
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31

    February 2015
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28

     March 2015
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

13.2 Duplo e comercial (&&)

O duplo e comercial `&&` funciona como um "e" lógico se o primeiro comando for bem-sucedido, então o segundo comando (à direita do `&&`) também será executado. Se o primeiro comando falhar, o segundo comando não será executado.

Para entender melhor como isso funciona, considere primeiro o conceito de falha e sucesso para comandos. Os comandos são bem-sucedidos quando funcionam corretamente e falham quando algo dá errado. Por exemplo, considere a linha de comando **ls /etc/xml**. O comando será bem-sucedido se o diretório `/etc/xml` estiver acessível e falhar se não estiver.

Por exemplo, o primeiro comando será bem-sucedito porque o diretório `/etc/xml` existe e está acessível, enquanto o segundo comando falhará porque não há diretório `/junk`:

```
sysadmin@localhost:~$ ls /etc/xml
catalog  catalog.old  xml-core.xml  xml-core.xml.old
sysadmin@localhost:~$ ls /etc/junk
ls: cannot access /etc/junk: No such file or directory
sysadmin@localhost:~$
```

A maneira que você usaria o sucesso ou falha do comando **ls** em conjunto com `&&` seria executar uma linha de comando como a seguinte:

```
sysadmin@localhost:~$ ls /etc/xml && echo success
catalog  catalog.old  xml-core.xml  xml-core.xml.old
success
sysadmin@localhost:~$ ls /etc/junk && echo success
ls: cannot access /etc/junk: No such file or directory
sysadmin@localhost:~$
```

No primeiro exemplo acima, o comando **echo** foi executado porque o comando **ls** foi bem-sucedido. No segundo exemplo, o comando **echo** não foi executado porque o comando **ls** falhou.

13.3 Pipe Duplo (||)

O pipe duplo || é um ”ou” lógico. Ele funciona de maneira semelhante ao &&; dependendo do resultado do primeiro comando, o segundo comando será executado ou será ignorado.

Com o pipe duplo, se o primeiro comando for executado com êxito, o segundo comando será ignorado; se o primeiro comando falhar, o segundo comando será executado. Em outras palavras, você está essencialmente dizendo ao shell: "Execute este primeiro comando ou o segundo".

No exemplo a seguir, o comando **echo** será executado apenas se o comando **ls** falhar:

```
sysadmin@localhost:~$ ls /etc/xml || echo failed
catalog  catalog.old  xml-core.xml  xml-core.xml.old
sysadmin@localhost:~$ ls /etc/junk || echo failed
ls: cannot access /etc/junk: No such file or directory
failed
sysadmin@localhost:~$
```