Manual de Referência do Bash

Documentação de Referência para o Bash
Edição 4.3, para `Bash` Versão 4.3.
Fevereiro de 2014

Chet Ramey, Case Western Reserve University
Brian Fox, Free Software Foundation

Este texto é uma breve descrição das características que estão presentes no shell Bash (versão 4.3, 02 de fevereiro de 2014).

Esta é a Edição 4.3, mais recentemente atualizada em 02 de fevereiro de 2014, do *The GNU Bash Reference Manual*, para `Bash`, Versão 4.3.

Na produção deste documento, buscou-se obter o máximo de qualidade possível, em respeito ao leitor. Entretanto, apesar de perseguida, a perfeição é algo difícil de ser alcançada. Por esse motivo, erros podem ter passado despercebidos. Qualquer ajuda no sentido de identificá-los é muito bem vinda, porém o leitor deveria estar consciente de que este documento é distribuido '`sem qualquer garantia`', implícita e/ou explícita.

Sumário

1 Introdução

1.1 O Que é Bash?

Bash é o shell, ou interpretador de linguagem de comando, para o sistema operacional GNU. O nome é um acrônimo para o '`Bourne-Again SHell`', uma homenagem a Stephen Bourne, o autor do ancestral direto do shell Unix atual `sh`, o qual apareceu na versão do Unix da Sétima Edição do Bell Labs Research.

Bash é largamente compatível com `sh` e incorpora características úteis oriundas do shell Korn `ksh` e do shell C `csh`. Ele é entendido para ser uma implementação conforme da porção das Ferramentas e Shell do IEEE POSIX da especificação IEEE POSIX (Padrão IEEE 1003.1). Ele oferece melhoramentos funcionais sobre o `sh` para ambos uso interativo e programação.

Ao mesmo tempo em que o sistema operacional GNU fornece outros shells, incluindo uma versão do `csh`, Bash é o shell padrão. Como outros softwares GNU, Bash é facilmente portável. Ele atualmente roda em quase todas as versões de Unix e alguns outros sistemas operacionais − existem implementações independentemente suportadas para MS-DOS, OS/2, e plataformas Windows.

1.2 O Que um shell?

Na base, um shell é simplesmente um processador de macro que executa comandos. O termo processador de macro significa funcionalidade onde texto e símbolos são expandidos para criar expressões mais amplas.

Um shell Unix é ambos um interpretador de comando e uma linguagem de programação. Como um interpretador de comando, o shell fornece a interface de usuário para o rico conjunto de utilitários GNU. As características de linguagem de programação permitem que tais utilitários sejam combinados. Arquivos contendo comandos podem ser criados, e se tornarem eles mesmos comandos. Esses novos comandos tem o mesmo status que os comandos de sistema em diretórios como `/bin`, permitindo aos usuários e grupos estabelecerem ambientes personalizados para automatizar suas tarefas cotidianas.

Shells podem ser utilizados interativamente ou não interativamente. No modo interativo, eles aceitam entrada digitada no teclado. Quando da execução não-interativamente, shells executam comandos lidos a partir de um arquivo.

Um shell permite a execução de comandos GNU, ambos síncronamente e assincronamente. O shell aguarda comandos síncronos completarem antes de aceitar mais entrada; comandos assíncronos continuam a executar em paralelo com o shell enquanto ele lê e executa comandos adicionais. As construções *redirection* permitem um controle refinado da entrada e saída de tais comandos. Além disso, o shell permite controle sobre o conteúdo dos ambientes de comandos.

Shells também fornecem um pequeno conjunto de comandos internos (*builtins*) implementando funcionalidade impossível ou inconveniente de se obter via utilitários separados. Por exemplo, `cd`, `break`, `continue`, e `exec` não podem ser implementados do lado de fora do shell, pois eles manipulam diretamente o próprio shell. Os comandos internos `history`, `getopts`, `kill`, ou `pwd`, entre outros, poderiam ser implementados em utilitários separados, mas eles são mais convenientes de se utilizar como comandos internos. Todos os comandos internos do shell são descritos em seções subsequentes.

Ao tempo em que executar comandos é essencial, a maior parte do poder (e complexidade) dos shells é devida às suas linguagens de programação embutidas. Como qualquer linguagem de alto nível, o shell fornece variáveis, construtores de fluxo de controle, citações e funções.

Shells oferecem recursos especificamente voltados para uso interativo ao invés de aumentar a linguagem de programação. Essas características interativas incluem controle de tarefa, edição de linha de comando, histórico de comando e apelidos. Cada uma dessas características está descrita neste manual.

2 Definições

Estas definições são utilizadas em todo o restante deste manual.

`POSIX` Uma família de padrões abertos de sistema baseado no Unix. Bash é concernente primariamente com a porção de Shell e Utilitários do padrão POSIX 1003.1.

`blank` Um carácter tab ou espaço.

`builtin` Um comando que é implementado internamente pelo próprio shell, ao invés de o ser por algum programa executável em algum lugar no sistema de arquivo.

`control operator`
Um `token` que desempenha uma função de controle. Ele é um `newline` ou um dos seguintes: '`||`', '`&&`', '`&`', '`;`', '`;;`', '`|`', '`|&`', '`(`', or '`)`'.

`exit status`
O valor retornado por um comando para quem o executou. O valor é restrito a oito bits, de forma que o valor máximo é 255.

`field` Uma unidade de texto que é o resultado de uma das expansões de shell. Após a expansão, quando da execução de um comando, os campos resultantes são utilizados como o nome do comando e argumentos.

`filename` Uma sequência de caracteres utilizada para identificar um arquivo.

`job` Um conjunto de tarefas compreendendo um canal de comunicação (pipeline), e quaisquer processos que descendam dele, os quais estão todos no mesmo grupo de processos.

`job control`
Um mecanismo pelo qual usuários podem seletivamente parar (suspend) e reiniciar (resume) a execução de processos.

`metacharacter`
Um carácter que, sem aspas, separa palavras. Um meta carácter é um `blank` ou um dos caracteres seguintes: '`|`', '`&`', '`;`', '`(`', '`)`', '`<`', ou '`>`'.

`name` Uma `word` consistindo apenas de letras, números, e sublinhados, e iniciando com uma letra ou sublinhado. `Name`s são utilizados como variáveis de shell e como nomes de função. Também referenciado como um `identifier`.

`operator` Um `control operator` ou um `redirection operator`. Veja-se Seção 3.6 [Redireções], Página 34, para uma lista de operadores de redireção. Operadores contém ao menos um `metacharacter` fora de aspas.

`process group`
Uma coleção de processos relacionados cada qual tendo o mesmo ID de grupo de processo.

`process group ID`
Um identificador único que representa um `process group` durante seu tempo de vida.

`reserved word`
: Uma `word` que tem um significado especial para o shell. A maior parte das palavras reservadas introduz construções de controle de fluxo do shell, tais como `for` e `while`.

`return status`
: Um sinônimo para `exit status`.

`signal`
: Um mecanismo pelo qual um processo pode ser notificado de um evento ocorrente no sistema pelo kernel.

`special builtin`
: Um comando interno de shell o qual foi classificado como especial pelo padrão POSIX.

`token`
: Uma sequência de caracteres considerada como sendo uma unidade simples pelo shell. Essa sequência é ou uma `word` ou um `operator`.

`word`
: Uma sequência de caracteres tratada como uma unidade pelo shell. Palavras não podem incluir `metacharacters` fora de aspas.

3 Características Básicas do Shell

Bash é um acrônimo para '`Bourne-Again SHell`'. O shell Bourne é o shell Unix tradicional originalmente escrito por Stephen Bourne. Todos os comandos internos do shell Bourne estão disponíveis no Bash. As regras para avaliação e encapsulamento dentro de aspas são tomadas da especificação POSIX para o shell Unix 'padrão'.

Este capítulo sumariza brevemente os 'blocos de construção' do shell: comandos, estruturas de controle, funções de shell, *parameters* do shell, expansões do shell, *redirections*, os quais são uma forma de direcionar entrada e saída de e para arquivos nomeados, e como o shell executa comandos.

3.1 Sintaxe do Shell

Quando o shell lê entrada, ele percorre uma sequência de operações. Se a entrada indica o início de um comando, o shell ignora o símbolo de comentário ('`#`'), e o restante daquela linha.

De outro lado, grosseiramente falando, o shell lê sua entrada e divide a entrada em palavras e operadores, empregando as regras de encapsulamento entre aspas para selecionar quais significados atribuir para as várias palavras e caracteres.

O shell então processa esses tokens em comandos e outros construtores, remove o significado especial de certas palavras ou caracteres, expande outros, redireciona entrada e saída conforme necessário, executa o comando especificado, aguarda pelo status de saída do comando, e torna esse status de saída disponível para inspeção posterior ou processamento.

3.1.1 Operação do Shell

O seguinte é uma breve descrição da operação do shell quando ele lê e executa um comando. Basicamente, o shell faz o seguinte:

1. Lê a entrada a partir de um arquivo (Seção 3.8 [Scripts de Shell], Página 42), a partir de uma cadeia de caracteres fornecida como um argumento para a opção de invocação `-c` (Seção 6.1 [Invocando o Bash], Página 87), ou a partir do terminal do usuário.
2. Divide a entrada em palavras e operadores, obedecendo às regras de encapsulamento entre aspas descritas em Seção 3.1.2 [Encapsulamento], Página 6. Esses tokens são separados por `metacharacters`. A expansão de apelidos é feita por esse passo (Seção 6.6 [Apelidos], Página 96).
3. Processa os tokens em comandos simples e comandos compostos (Seção 3.2 [Comandos do Shell], Página 8).
4. Desempenha as várias expansões de shell (Seção 3.5 [Expansões de Shell], Página 22), dividindo os tokens expandidos em listas de nomes de arquivos (Seção 3.5.8 [Expansão de Nome de Arquivo], Página 32) e comandos e argumentos.
5. Desempenha quaisquer redireções necessárias (Seção 3.6 [Redireções], Página 34) e remove os operadores de redireção e seus operandos da lista de argumentos.
6. Executa o comando (Seção 3.7 [Execução de Comandos], Página 38).
7. Opcionalmente aguarda o término da execução do comando e coleta seu status de saída (Seção 3.7.5 [Situação de Saída], Página 41).

3.1.2 Encapsulamento

O encapsulamento entre aspas é utilizado para remover o significado especial de determinados caracteres ou palavras para o shell. O encapsulamento pode ser utilizado para desabilitar o tratamento especial para caracteres especiais, prevenir palavras reservadas de serem reconhecidas como tais, e prevenir a expansão de parâmetro.

Cada um dos meta caracteres do shell (Capítulo 2 [Definições], Página 3) tem um significado especial para o shell e devem ser encapsulados caso devam representar eles mesmos. Quando as facilidades da expansão de histórico de comandos estão sendo utilizadas (Seção 9.3 [History Interaction], Página 147), o carácter de *history expansion*, em regra '`!`', deve ser encapsulado para prevenir a expansão de histórico. Veja-se Seção 9.1 [Facilidades do Histórico de Bash], Página 145, para mais detalhes acerca da expansão de histórico.

Existem três mecanismos de encapsulamento em aspas: o *escape character*, encapsulamento simples, e encapsulamento duplo.

3.1.2.1 Carácter de Escape

Uma barra invertida não encapsulada '\' é o carácter de escape do Bash. Esse carácter preserva o valor literal do próximo carácter que se segue, com exceção do `newline`. Se um par `\newline` aparece, e a própria barra invertida não estiver encapsulada entre aspas, o `\newline` é tratado como uma continuação de linha (isto é, ele é removido do fluxo de entrada e efetivamente ignorado).

3.1.2.2 Aspas Simples

Encapsular caracteres em aspas simples (''') preserva o valor literal de cada carácter dentro do encapsulamento. Um encapsulamento simples não deve ocorrer entre aspas simples, ainda que precedido por uma barra invertida.

3.1.2.3 Aspas Duplas

Encapsular caracteres em aspas duplas ('`"`') preserva o valor literal de todos os caracteres dentro das aspas, com exceção de '`$`', '`‘`', '\', e, quando a expansão de histórico esteja habilitada, '`!`'. Os caracteres '`$`' e '`‘`' conservam o significado especial deles quando dentro de aspas duplas (Seção 3.5 [Expansões de Shell], Página 22). A barra invertida conserva seu significado especial apenas quando seguida por um dos seguintes caracteres: '`$`', '`‘`', '`"`', '\', ou `newline`. Dentro de aspas duplas, barras invertidas que são seguidas por um desses caracteres são removidas. Barras invertidas precedendo caracteres sem um significado especial são deixadas sem modificação. Uma aspa dupla pode ser encapsulada em aspas duplas precedendo-se ela com uma barra invertida. Se habilitada, a expansão de histórico será feita, a menos que um '`!`' que esteja aparecendo em aspas duplas seja encapsulado utilizando-se uma barra invertida. A barra invertida que precede um '`!`' não é removida.

Os parâmetros especiais '`*`' e '`@`' tem significado especial quando estejam entre aspas duplas (Seção 3.5.3 [Expansão de Parâmetro de Shell], Página 25).

3.1.2.4 Encapsulamento ANSI-C

Palavras da forma `$'`*`string`*`'` são tratadas especialmente. A palavra expande para *string*, com caracteres encapsulados por barras invertidas sendo substituídos conforme especificado pelo padrão ANSI C. As sequências de encapsulamento de barra invertida, se presentes, são decodificadas conforme a seguir:

`\a` alerta (sino)

`\b` barra invertida

`\e`
`\E` um carácter de encapsulamento (não ANSI C)

`\f` alimentação de formulário

`\n` nova linha

`\r` retorno de carro

`\t` tab horizontal

`\v` tab vertical

`\\` barra invertida

`\'` aspa simples

`\"` aspa dupla

`\`*nnn* o carácter de oito bits cujo valor é o valor octal *nnn* (de um até três dígitos)

`\x`*HH* o carácter de oito bits cujo valor é o valor hexadecimal *HH* (um ou dois dígitos hexadecimais)

`\u`*HHHH* o carácter Unicode (ISO/IEC 10646) cujo valor é o valor hexadecimal *HHHH* (de um até quatro dígitos hexadecimais)

`\U`*HHHHHHHH*
o carácter Unicode (ISO/IEC 10646) cujo valor é o valor hexadecimal *HHHHHHHH* (de um até oito dígitos hexadecimais)

`\c`*x* um carácter control-*x*

O resultado expandido é encapsulado entre aspas simples, como se o sinal de dólar não estivesse presente.

3.1.2.5 Tradução Específica por Locale

Uma cadeia de caracteres dentro de um encapsulamento em aspas duplas precedido por um sinal de dólar (‘`$`’) fará com que a cadeia de caracteres seja traduzida conforme o locale atual. Se o locale atual é `C` ou `POSIX`, o sinal de dólar é ignorado. Se a cadeira de caracteres for traduzida e substituída, o substituto é encapsulado entre aspas duplas.

Alguns sistemas utilizam o catálogo de mensagem selecionado pela variável de shell `LC_MESSAGES`. Outros criam o nome do catálogo de mensagem a partir do valor da variável de shell `TEXTDOMAIN`, possivelmente adicionando um sufixo ‘`.mo`’. Se você utiliza a variável `TEXTDOMAIN`, você obrigatoriamente precisa configurar a variável `TEXTDOMAINDIR` para a localização dos arquivos de catálogo de mensagem. Outros ainda utilizam ambas as variáveis neste desenho: `TEXTDOMAINDIR`/`LC_MESSAGES`/LC_MESSAGES/`TEXTDOMAIN`.mo.

3.1.3 Comentários

Em um shell não interativo, ou um shell interativo no qual a opção `interactive_comments` para o comando interno `shopt` esteja habilitada (Seção 4.3.2 [O Comando Interno Shopt], Página 68), uma palavra iniciando com '`#`' faz com que essa palavra e todos os caracteres restantes naquela linha sejam ignorados. Um shell interativo sem a opção `interactive_comments` habilitada não permite comentários. A opção `interactive_comments` é habilitada por padrão em shells interativos. Veja-se Seção 6.3 [Shells Interativos], Página 91, para uma descrição do que torna um shell interativo.

3.2 Comandos do Shell

Um comando de shell simples como `echo a b c` consiste do próprio comando seguido por argumentos, separados por espaços.

Comandos de shell mais complexos são compostos de comandos simples arranjados juntos em uma variedade de possibilidades: em um canal de comunicação (pipeline) no qual a saída de um comando se torna a entrada de um segundo, em um loop ou construção condicional, ou em algum outro agrupamento.

3.2.1 Comandos Simples

Um comando simples é o tipo de comando encontrado mais frequentemente. Ele é simplesmente uma sequência de palavras separadas por `blank`s, terminada por um dos operadores de controle do shell (Capítulo 2 [Definições], Página 3). A primeira palavra geralmente especifica um comando a ser executado, com o restante das palavras sendo os argumentos daquele comando.

O status de retorno (Seção 3.7.5 [Situação de Saída], Página 41) de um comando simples é seu status de saída como fornecido pela função `waitpid` definida no padrão POSIX 1003.1, ou 128+*n* se o comando foi terminado pelo sinal *n*.

3.2.2 Canais de Comunicação

Um canal de comunicação (`pipeline`) é uma sequência de comandos simples separada por um dos operadores de controle '`|`' ou '`|&`'.

O formato de um tubo é

```
[time [-p]] [!] command1 [ | or |& command2 ] ...
```

A saída de cada comando dentro do canal de comunicação é conectada via tubo à entrada do próximo comando. Isto é, cada comando lê a saída do comando anterior. Essa conexão é feita antes de quaisquer redireções especificadas pelo comando.

Se '`|&`' for utilizado, o descritor de erro padrão do *command1*, em adição ao seu descritor de saída padrão, é conectado ao descritor de entrada padrão do *command2* por meio do tubo; isso é um atalho para `2>&1 |`. Essa redireção implícita do descritor de erro padrão para o descritor de saída padrão é feita após quaisquer redireções especificadas pelo comando.

A palavra reservada `time` faz com que sejam impressas estatísticas de temporização para o canal de comunicação tão logo ele finalize. As estatísticas atualmente consistem do tempo (wall-clock) decorrido e tempo de usuário e de sistema consumidos pela execução do comando. A opção `-p` modifica o formato da saída para aquele especificado por POSIX. Quando o shell está em modo POSIX (Seção 6.11 [O Modo POSIX de Bash], Página 102),

ele não reconhece `time` como uma palavra reservada se o próximo token inicia com um '`-`'. A variável `TIMEFORMAT` pode ser configurada para um formato de cadeia de caracteres que especifique como a informação de temporização deveria ser exibida. Veja-se Seção 5.2 [Variáveis do Bash], Página 75 para uma descrição dos formatos disponíveis. O uso de `time` como uma palavra reservada permite a temporização de comandos internos do shell, de funções do shell, e de canais de comunicação (pipelines). Um comando `time` externo não pode temporizar esses facilmente.

Quando o shell está em modo POSIX (Seção 6.11 [O Modo POSIX de Bash], Página 102), `time` deve ser seguido por um newline. Nesse caso, o shell exibe o tempo de usuário e de sistema total consumido pelo shell e seus filhos. A variável `TIMEFORMAT` deve ser utilizada para especificar o formato da informação de temporização.

Se o canal de comunicação não estiver sendo executado assincronamente (Seção 3.2.3 [Listas], Página 9), o shell aguarda que todos os comandos no canal de comunicação finalizem.

Cada comando em um canal de comunicação é executado em seu próprio sub shell (Seção 3.7.3 [Ambiente de Execução de Comando], Página 39). O status de saída de um canal de comunicação é o status de saída do último comando no canal de comunicação, a menos que a opção `pipefail` esteja habilitada (Seção 4.3.1 [O Comando Interno Set], Página 63). Se a opção `pipefail` estiver habilitada, o status de retorno do canal de comunicação é o valor do último (mais à direita) comando a sair com um status diferente de zero, ou zero se todos os comandos saírem com sucesso. Se a palavra reservada '`!`' preceder o canal de comunicação, o status de saída é a negação lógica do status de saída conforme descrito acima. O shell aguarda que todos os comandos no canal de comunicação terminem antes de retornar um valor.

3.2.3 Listas de Comandos

Uma `list` é uma sequência de um os mais canais de comunicação separados por um dos operadores '`;`', '`&`', '`&&`', ou '`||`', e opcionalmente finalizados por um '`;`', '`&`', ou um `newline`.

Dessa lista de operadores, '`&&`' e '`||`' tem igual precedência, seguidos por '`;`' e '`&`', os quais tem igual precedência.

Uma sequência de um ou mais newlines podem aparecer em uma `list` para delimitar comandos, equivalente a ponto e vírgula.

Se um comando é finalizado pelo operador de controle '`&`', o shell executa o comando assincronamente em um sub shell. Isso é conhecido como executar o comando no *background*. O shell não aguarda que o comando finalize a execução, e o status de retorno é 0 (verdadeiro). Quando o controle de tarefas não está ativo (Capítulo 7 [Controle de Tarefa], Página 106), a entrada padrão para comandos assíncronos, na ausência de quaisquer redireções explícitas, é redirecionada a partir de `/dev/null`.

Comandos separados por um '`;`' são executados sequencialmente; o shell aguarda que cada comando finalize sua execução. O status de retorno é o status de saída do último comando executado.

As listas AND e OR são sequências de um ou mais canais de comunicação separados por operadores de controle '`&&`' e '`||`', respectivamente. As listas AND e OR são executadas com associatividade à esquerda.

Uma lista AND tem a forma

```
command1 && command2
```

command2 é executado se, e somente se, *command1* retorna um status de saída igual a zero.

Uma lista OR tem a forma

```
command1 || command2
```

command2 é executado se, e somente se, *command1* retorna um status de saída qualquer diferente de zero.

O status de retorno das listas AND e OR é o status de saída do último comando executado na lista.

3.2.4 Comandos Compostos

Comandos de composição são os construtores de programação do shell. Cada construtor inicia com uma palavra reservada ou operador de controle e é finalizado pela palavra reservada ou operador correspondentes. Quaisquer redireções (Seção 3.6 [Redireções], Página 34) associadas com um comando de composição se aplicam a todos os comandos dentro desse comando de composição a menos que explicitamente sobrepostas.

Na maioria dos casos uma lista de comandos em uma descrição de comando de composição pode ser separada do restante do comando por um ou mais newlines, e pode ser seguida por um newline ao invés de um ponto e vírgula.

Bash disponibiliza construtores de loops, comandos condicionais e mecanismos para agrupar comandos e executá-los como uma unidade.

3.2.4.1 Construtores de Ciclos

Bash suporta os seguintes construtores de loops.

Note que sempre que um '`;`' aparecer na descrição de uma sintaxe de comando, ele pode ser substituído por um ou mais newlines.

`until` A sintaxe do comando `until` é:

```
until test-commands; do consequent-commands; done
```

Execute *consequent-commands* tantas vezes quantas *test-commands* tenham um status de saída qualquer diferente de zero. O status de retorno é o status de saída do último comando executado no *consequent-commands*, ou zero se nenhum comando foi executado.

`while` A sintaxe do comando `while` é:

```
while test-commands; do consequent-commands; done
```

Execute *consequent-commands* tantas vezes quantas *test-commands* tenham um status de saída igual a zero. O status de retorno é o status de saída do último comando executado em *consequent-commands*, ou zero se nenhum comando foi executado.

`for` A sintaxe do comando `for` é:

```
for name [ [in [words ...] ] ; ] do commands; done
```

Expanda *words*, e execute *commands* uma vez para cada membro na lista resultante, com *name* apontando para o membro atual. Se '`in words`' não estiver

presente, o comando `for` executa os *commands* uma vez para cada parâmetro posicional que estiver configurado, como se '`in "$@"`' tivesse sido especificado (Seção 3.4.2 [Parâmetros Especiais], Página 21). O status de retorno é o status de saída do último comando que executar. Se não existirem itens na expansão de *words*, nenhum comando é executado, e o status de retorno será igual a zero.

Uma forma alternativa do comando `for` também é suportada:

```
for (( expr1 ; expr2 ; expr3 )) ; do commands ; done
```

Primeiro, a expressão aritmética *expr1* é calculada de acordo com as regras descritas abaixo (Seção 6.5 [Aritmética de Shell], Página 95). A expressão aritmética *expr2* é então calculada repetidamente até que ela seja igual a zero. A cada vez que *expr2* seja igual a um valor qualquer diferente de zero, *commands* são executados e a expressão aritmética *expr3* é calculada. Se qualquer expressão for omitida, ela se comporta como se o resultado dela fosse igual a 1. O valor de retorno é o status de saída do último comando em *commands* que for executado, ou falso se quaisquer das expressões for inválida.

Os comandos internos `break` e `continue` podem ser utilizados para controlar a execução do loop.

3.2.4.2 Construtores Condicionais

`if` A sintaxe do comando `if` é:

```
if test-commands; then
  consequent-commands;
[elif more-test-commands; then
  more-consequents;]
[else alternate-consequents;]
fi
```

A lista *test-commands* é executada, e se o seu status de retorno for igual a zero, a lista *consequent-commands* é executada. Se *test-commands* retornar um status diferente de zero, cada lista `elif` é executada, e se o seu status de saída for igual a zero, os *more-consequents* correspondentes são executados e o comando finaliza. Se '`else alternate-consequents`' estiver presente, e o comando final na cláusula `if` ou `elif` final tiver um status de saída qualquer diferente de zero, então *alternate-consequents* são executados. O status de retorno é o status de saída do último comando executado, ou zero se nenhuma condição testada for verdadeira.

`case` A sintaxe do comando `case` é:

```
case word in [ [(] pattern [| pattern]...) command-list ;;]... esac
```

`case` seletivamente executará a *command-list* correspondente à primeira variável *pattern* que coincidir com a variável *word*. Se a opção de shell `nocasematch` (veja-se a descrição de `shopt` em Seção 4.3.2 [O Comando Interno Shopt], Página 68) estiver habilitada, então a coincidência é testada sem considerar maiúsculas e minúsculas. O carácter '`|`' é utilizado para separar múltiplas variáveis *pattern*, e o operador '`)`' finaliza a lista de variáveis. Uma lista de variáveis *pattern* e uma lista de comandos associada é conhecida como uma *clause*.

Cada cláusula deve necessariamente ser finalizada com um '`;;`', '`;&`', ou um '`;;&`'. A *word* está sujeita a expansão de til, expansão de parâmetro, substituição de comando, expansão aritmética e remoção de aspas antes que a coincidência seja testada. Cada *pattern* está sujeita a expansão de til, expansão de parâmetro, substituição de comando e expansão aritmética.

Pode existir um número arbitrário de cláusulas `case`, cada uma das quais finalizada por um '`;;`', '`;&`', ou um '`;;&`'. O primeiro padrão que coincidir determina a lista de comandos a serem executados. É bastante comum utilizar '`*`' como o padrão final para definir o caso padrão, sabido que tal padrão sempre coincidirá.

Eis um exemplo utilizando `case` em um script que poderia ser utilizado para descrever uma característica interessante de um animal:

```
echo -n "Informe o nome de um animal: "
read ANIMAL
echo -n "O $ANIMAL tem "
case $ANIMAL in
  cavalo | cachorro | gato) echo -n "quatro";;
  homem | canguru ) echo -n "duas";;
  *) echo -n "um número desconhecido de";;
esac
echo " pernas."
```

Se o operador '`;;`' for utilizado, então nenhuma coincidência subsequente é tentada após o primeiro padrão coincidir. A utilização de um '`;&`' no lugar de um '`;;`' faz com que a execução continue com a *command-list* associada com a próxima cláusula, se existir alguma. A utilização de um '`;&`' no lugar de um '`;;`' faz com que o shell teste os padrões na próxima cláusula, caso exista alguma, e execute quaisquer *command-list* associadas em cima de uma coincidência testada com sucesso.

O status de retorno é zero se nenhuma variável *pattern* coincidir. Por outro lado, o status de retorno é o status de saída da *command-list* executada.

`select`

O construtor `select` permite a fácil geração de menus. Ele tem quase a mesma sintaxe que o comando `for`:

`select` *name* `[in` *words* `...]; do` *commands*`; done`

A lista de palavras seguintes a `in` é expandida, gerando uma lista de itens. O conjunto de palavras expandidas é impresso no fluxo de saída padrão de erro, cada uma das quais precedida por um número. Se o '`in` *words*' for omitido, os parâmetros posicionais serão impressos, como se '`in "$@"`' tivesse sido especificado. O prompt `PS3` é então exibido e uma linha é lida a partir da entrada padrão. Se a linha consistir de um número correspondente a uma das palavras exibidas, então o valor de *name* é configurada para aquela palavra. Se a linha estiver vazia, então o comando `select` completa sua execução. Qualquer outro valor lido faz com que *name* seja configurada para nulo. A linha lida é salva na variável `REPLY`.

Os *commands* são executados após cada seleção até que um comando `break` seja executado, ponto no qual o comando `select` completa a sua execução.

Eis um exemplo que permite ao usuário pegar um nome de arquivo a partir do diretório de trabalho atual, e exibir o nome e índice do arquivo selecionado.

```
select fname in *;
do
echo você selecionou $fname \($REPLY\)
break;
done
```

`((...))`

```
(( expression ))
```

A *expression* aritmética é calculada de acordo com as regras descritas abaixo (Seção 6.5 [Aritmética de Shell], Página 95). Se o valor da expressão for diferente de zero, então o status de retorno é 0; do contrário o status de retorno é 1. Isso é exatamente equivalente a

```
let "expression"
```

Veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52, para uma descrição completa acerca do comando interno `let`.

`[[...]]`

```
[[ expression ]]
```

Retorna um status de 0 ou 1 dependendo da avaliação da expressão condicional *expression*. Expressões são compostas de primários descritos abaixo em Seção 6.4 [Expressões Condicionais de Bash], Página 93. A divisão de palavras (word splitting) e a expansão de nome de arquivo não são feitas nas palavras contidas entre o `[[` e o `]]`; expansão de til, expansão de variável e parâmetro, expansão aritmética, substituição de comando, substituição de processo e remoção de aspas são feitas. Operadores condicionais tais como '`-f`' devem necessariamente estar fora de aspas para serem reconhecidos como primários.

Quando utilizados com `[[`, os operadores '`<`' e '`>`' ordenam lexicograficamente utilizando o locale atual.

Quando os operadores '`==`' e '`!=`' são utilizados, a sequência de caracteres à direita do operador é considerada um padrão e uma coincidência é tentada de acordo com as regras descritas abaixo em Seção 3.5.8.1 [Coincidência de Modelo], Página 32, como se a opção de shell `extglob` estivesse habilitada. O operador '`=`' é idêntico ao '`==`'. Se a opção de shell `nocasematch` (veja-se a descrição de `shopt` em Seção 4.3.2 [O Comando Interno Shopt], Página 68) estiver habilitada, então a coincidência é feita sem levar em consideração maiúsculas e minúsculas. O valor de retorno é 0 se a sequência de caracteres coincidir ('`==`') ou não coincidir ('`!=`') com o padrão, e 1 nos outros casos. Qualquer parte do padrão pode ser colocada entre aspas para forçar que tal porção seja testada como uma sequência de caracteres.

Um operador binário adicional, '`=~`', está disponível, com a mesma precedência que '`==`' e '`!=`'. Quando esse operador é utilizado, a sequência de caracteres à direita do operador é considerada como sendo uma expressão regular estendida e uma coincidência é tentada de acordo (como em *regex*3)). O valor de retorno é 0 se a sequência de caracteres coincidir com o padrão, e 1 caso contrário.

Se a expressão regular estiver sintaticamente incorreta, o valor de retorno da expressão condicional é igual a 2. Se a opção de shell `nocasematch` (veja-se a descrição de `shopt` em Seção 4.3.2 [O Comando Interno Shopt], Página 68) estiver habilitada, a coincidência é tentada sem levar em consideração maiúsculas e minúsculas. Qualquer parte do padrão pode ser colocado entre aspas para forçar que essa porção seja tratada como sendo uma sequência de caracteres. As expressões entre parênteses em expressões regulares devem necessariamente serem tratadas cuidadosamente, dado que caracteres normais para indicar aspas perdem seus significados entre parênteses. Se o padrão for armazenado em uma variável de shell, então colocar-se a expansão da variável entre aspas força o padrão inteiro ser tratado como sendo uma sequência de caracteres. As subsequências de caracteres testadas por sub expressões entre parênteses dentro da expressão regular são salvas na variável de vetor `BASH_REMATCH`. O elemento do `BASH_REMATCH` cujo índice é 0 é a porção da sequência de caracteres coincidente com a expressão regular inteira. O elemento do `BASH_REMATCH` cujo índice é *n* é a porção da sequência de caracteres coincidente com a *n*ésima sub expressão entre parênteses.

Por exemplo, o seguinte coincidirá com uma linha (armazenada na variável de shell *line*) Se existir uma sequência de caracteres no valor consistente de qualquer número, incluindo zero; de caracteres espaço; zero ou um instâncias de '`a`', então um '`b`':

```
[[ $line =~ [[:space:]]*(a)?b ]]
```

Isso significa que valores como '`aab`' e ' `aaaaaab`' coincidirão, como também coincidirá uma linha que contenha um '`b`' em qualquer lugar no seu valor.

O armazenamento da expressão regular em uma variável de shell é frequentemente uma maneira útil de se evitar problemas com caracteres que fazem encapsulamento e que são especiais para o shell. As vezes é difícil de se especificar uma expressão regular literalmente sem se utilizar aspas, ou se manter um controle do encapsulamento utilizado por expressões regulares enquanto se presta atenção à remoção de aspas do shell.

A utilização de uma variável de shell para armazenar o padrão reduz tais problemas. Por exemplo, o seguinte é equivalente ao acima:

```
pattern='[[:space:]]*(a)?b'
[[ $line =~ $pattern ]]
```

Se você deseja coincidir um carácter que seja especial para a gramática da expressão regular, então esse carácter tem de ser encapsulado para remover o seu significado especial. Isso significa que no padrão '`xxx.txt`', o '`.`' coincide com qualquer carácter na sequência de caracteres (seu significado usual de expressão regular), porém no padrão '`"xxx.txt"`' ele pode apenas coincidir com um '`.`' literal. Os programadores shell deveriam ter um cuidado especial com as barras invertidas, dado que as barras invertidas são utilizadas tanto pelo shell quanto por expressões regulares para remover o significado especial do carácter seguinte à barra invertida. Os dois conjuntos de comandos seguintes *não* são equivalentes:

```
pattern='\.'
```

```
[[ . =~ $pattern ]]
[[ . =~ \. ]]

[[ . =~ "$pattern" ]]
[[ . =~ '\.' ]]
```

Os primeiros dois terão sucesso, porém os dois últimos não, pois nos dois últimos a barra invertida será parte do padrão a ser testado. Nos primeiros dois exemplos, a barra invertida remove o significado especial do '`.`', de forma que o '`.`' literal coincide. Se a sequência de caracteres nos dois primeiros exemplos fosse qualquer outra coisa que não o '`.`', diga-se '`a`', o padrão não coincidiria, pois o '`.`' encapsulado no padrão perde seu significado especial de coincidir com qualquer carácter único.

As expressões podem ser combinadas utilizando-se os seguintes operadores, listados em ordem decrescente de precedência:

`( `*expression*` )`
: Retorna o valor de *expression*. Isso pode ser utilizado para anular a precedência normal de operadores.

`! `*expression*
: Verdadeiro se *expression* for falsa.

expression1 `&&` *expression2*
: Verdadeiro se ambas *expression1* e *expression2* forem falsas.

expression1 `||` *expression2*
: Verdadeiro se ou *expression1* ou *expression2* for verdadeira.

Os operadores `&&` e `||` não calculam *expression2* se o valor de *expression1* for suficiente para determinar o valor de retorno da expressão condicional inteira.

3.2.4.3 Agrupando Comandos

Bash provê duas maneiras de agrupar uma lista de comandos a serem executados como uma unidade. Quando comandos são agrupados, as redireções podem ser aplicadas à lista de comandos inteira. Por exemplo, a saída de todos os comandos na lista pode ser redirecionada para um fluxo único.

`()`

```
( list )
```

Colocar uma lista de comandos entre parênteses faz com que um ambiente de sub shell seja criado (Seção 3.7.3 [Ambiente de Execução de Comando], Página 39), e cada um dos comandos na *list* sejam executados nesse sub shell. Uma vez que a *list* seja executada em um sub shell, atribuições à variáveis não permanecem em efeito após a finalização do sub shell.

`{}`

```
{ list; }
```

> Colocar uma lista de comandos entre chaves faz com que a lista seja executada no contexto do shell atual. Nenhum sub shell é criado. O ponto e vírgula (ou newline) seguinte a *list* é obrigatório.

Adicionalmente à criação de um sub shell, existe uma diferença sútil entre essas duas construções devida a razões históricas. As chaves são `reserved words`, de forma que elas devem necessariamente serem separadas da *list* por `blank`s ou outros meta caracteres de shell. Os parênteses são `operators`, e são reconhecidos como tokens separados pelo shell mesmo se eles não forem separados da *list* por espaços em branco.

O status de saída de ambas dessas construções é o status de saída de *list*.

3.2.5 Coprocessos

Um `coprocess` é um comando de shell precedido pela palavra reservada `coproc`. Um co-processo é executado assincronamente em um sub shell, como se o comando tivesse sido finalizado com o operador de controle '`&`', com um tubo de mão dupla estabelecido entre o shell em execução e o co-processo.

O formato de um co-processo é:

```
coproc [NAME] command [redirections]
```

Isso cria um co-processo chamado *NAME*. Se *NAME* não for informado, então o nome padrão é *COPROC*. *NAME* deve necessariamente não ser informado se *command* for um comando simples (Seção 3.2.1 [Comandos Simples], Página 8); do contrário, *NAME* é interpretada como sendo a primeira palavra do comando simples.

Quando o co-processo é executado, o shell cria um vetor variável (Seção 6.7 [Vetores], Página 97) chamado `NAME` no contexto do shell em execução. A saída padrão de *command* é conectada via tubo ao arquivo descritor no shell em execução, e esse arquivo descritor é atribuído ao primeiro elemento do vetor que é `NAME`[0]. A entrada padrão de *command* é conectada via tubo ao arquivo descritor no shell em execução, e esse arquivo descritor é atribuído ao segundo elemento do vetor que é `NAME`[1]. Esse tubo é estabelecido antes de quaisquer redireções especificadas pelo comando (Seção 3.6 [Redireções], Página 34). Os descritores de arquivo podem ser utilizados como argumentos para redireções e comandos de shell utilizando-se expansões de palavras padrão. Os descritores de arquivo não estão disponíveis em sub shells.

O Identificador (ID) de processo do shell gerado para executar o co-processo está disponível como sendo o valor da variável `NAME`_PID. O comando interno `wait` pode ser utilizado para aguardar o co-processo finalizar.

Dado que o co-processo é criado como um comando assíncrono, o comando `coproc` sempre retorna sucesso. O status de retorno de um co-processo é o status de saída de *command*.

3.2.6 GNU Parallel

Existem maneiras de executar comandos em paralelo os quais não são internos ao Bash. GNU Parallel é uma ferramenta para fazer justamente isso.

GNU Parallel, como seu próprio nome sugere, pode ser utilizado para construir e executar comandos em paralelo. Você pode executar o mesmo comando com argumentos diferentes, quer sejam nomes de arquivos, nomes de usuários, nomes de máquinas ou linhas lidas de

um arquivo. GNU Parallel fornece referências de atalhos para muitas das mais comuns operações (linhas de entrada, várias porções da linha de entrada, maneiras diferentes de especificar a fonte de entrada e assim por diante). GNU Parallel pode substituir `xargs` ou alimentar comandos a partir de suas fontes de entrada para várias instâncias distintas de Bash.

Para uma descrição completa, veja-se a documentação de GNU Parallel. Alguns exemplos deveriam fornecer uma introdução breve ao seu uso.

Por exemplo, é simples substituir `xargs` para compactar todos os arquivos html com gzip no diretório atual e seus subdiretórios:

```
find . -type f -name '*.html' -print | parallel gzip
```

Se você necessitar proteger caracteres especiais tais como newlines em nomes de arquivos, utilize a opção do find `-print0` e a opção do GNU Parallel `-0`.

Você pode utilizar GNU Parallel para mover arquivos do diretório atual quando o número de arquivos for demasiadamente grande para se processar com uma invocação de `mv`:

```
ls | parallel mv {} destdir
```

Como se pode observar, o {} é substituído por cada linha lida a partir da entrada padrão. Ao tempo em que a utilização de `ls` funcionará na maioria das instâncias, isso não é suficiente para lidar com todos os nomes de arquivos. Caso se necessite acomodar caracteres especiais em nomes de arquivos, pode-se utilizar

```
find . -depth 1 \! -name '.*' -print0 | parallel -0 mv {} destdir
```

conforme mencionado acima.

Isso executará tantos comandos `mv` quantos sejam os arquivos no diretório de trabalho atual. Pode-se simular um `xargs` paralelo adicionando-se a opção `-X`:

```
find . -depth 1 \! -name '.*' -print0 | parallel -0 -X mv {} destdir
```

GNU Parallel pode substituir certos idiomas comuns que operam em linhas lidas a partir de um arquivo (nesse caso, nomes de arquivos listados um por linha):

```
while IFS= read -r x; do
do-something1 "$x" "config-$x"
do-something2 < "$x"
done < file | process-output
```

com uma sintaxe mais compacta reminiscente das anônimas:

```
cat list | parallel "do-something1 {} config-{} ; do-something2 < {}" | process-output
```

GNU Parallel fornece um mecanismo implementado internamente para remover extensões de nomes de arquivos, o qual se presta a transformações de arquivos em lote ou renomeamento:

```
ls *.gz | parallel -j+0 "zcat {} | bzip2 >{.}.bz2 && rm {}"
```

Isso irá recompactar todos os arquivos no diretório de trabalho atual com nomes terminando em .gz utilizando bzip2, executando uma tarefa por CPU (-j+0) em paralelo. (Utiliza-se `ls` para brevidade aqui; utilizando-se `find` como acima é mais robusto em face de nomes de arquivos contendo caracteres inesperados). GNU Parallel pode obter argumentos a partir da linha de comando; o exemplo acima também pode ser escrito como

```
parallel "zcat {} | bzip2 >{.}.bz2 && rm {}" ::: *.gz
```

Se um comando gerar saída, pode-se desejar preservar a ordem de entrada na saída. Por exemplo, o comando seguinte

```
{ echo foss.org.my ; echo debian.org; echo freenetproject.org; } | parallel traceroute
```

Exibirá aquela saída da invocação de traceroute que finalizar primeiro. Adicionando-se a opção `-k`

```
{ echo foss.org.my ; echo debian.org; echo freenetproject.org; } | parallel -k tracero
```

se terá a certeza de que a saída de `traceroute foss.org.my` será exibida primeiro.

Finalmente, GNU Parallel pode ser utilizado para se executar uma sequência de comandos de shell em paralelo, similar a '`cat file | bash`'. Não é incomum tomar-se uma lista de nomes de arquivos, criar-se uma série de comandos de shell para operar em cima deles, e fornecer tal lista de comandos a um shell. GNU Parallel pode acelerar isso. Presumindo-se que `file` contém uma lista de comandos de shell, um por linha,

```
parallel -j 10 < file
```

irá avaliar os comandos utilizando o shell (dado que nenhum comando explícito é fornecido como um argumento), em blocos de dez (10) tarefas de shell por vez.

3.3 Funções de Shell

Funções de shell são uma maneira de agrupar comandos para execução posterior utilizando um nome único para o grupo. Elas são executadas exatamente como um comando "regular". Quando o nome de uma função de shell é utilizada como um nome de comando simples, a lista dos comandos associados com aquele nome de função é executada. Funções de shell são executadas no contexto do shell atual; nenhum processo novo é criado para interpretá-los.

As funções são declaradas utilizando-se esta sintaxe:

name `()` *compound-command* `[` *redirections* `]`

or

`function` *name* `[()]` *compound-command* `[` *redirections* `]`

Isso define uma função de shell chamada *name*. A palavra reservada `function` é opcional. Se a palavra reservada `function` for informada, então os parênteses são opcionais. O *body* da função é o comando composto *compound-command* (Seção 3.2.4 [Comandos Compostos], Página 10). Esse comando é em regra uma *list* encapsulada entre { e }, porém pode ser qualquer comando composto listado acima. *compound-command* é executado sempre que *name* for especificado como o nome do comando. Quando o shell está em modo POSIX (Seção 6.11 [O Modo POSIX de Bash], Página 102), *name* pode não ser o mesmo que um dos comandos internos especiais (Seção 4.4 [Comandos Internos Especiais], Página 74). Quaisquer redireções (Seção 3.6 [Redireções], Página 34) associadas com a função de shell são feitas quando a função é executada.

Uma definição de função pode ser deletada utilizando-se a opção `-f` para o comando interno `unset` (Seção 4.1 [Comandos Internos do Shell Bourne], Página 44).

O status de saída de uma definição de função é zero a menos que um erro de sintaxe ocorra ou uma função somente leitura com o mesmo nome já exista. Quando executada, o status de saída de uma função é o status de saída do último comando executado no corpo dessa função.

Perceba que, por razões históricas, no uso mais comum as chaves que envolvem o corpo de uma função devem necessariamente serem separadas do corpo por caracteres `blank`s ou newlines. Isso é porque as chaves são palavras reservadas e apenas são reconhecidas como tais quando elas estão separadas da lista de comando por espaço em branco ou um outro meta carácter de shell. Também, quando da utilização de chaves, a *list* deve necessariamente ser finalizada por um ponto e vírgula, ou por um '`&`', ou por um newline.

Quando a função é executada, os argumentos para a função se tornam os parâmetros posicionais durante a execução da função (Seção 3.4.1 [Parâmetros Posicionais], Página 21). O parâmetro especial '`#`' que é expandido para o número de parâmetros posicionais é atualizado para refletir a mudança. O parâmetro especial `0` fica imodificado. O primeiro elemento da variável `FUNCNAME` é configurado para o nome da função ao tempo em que a função é executada.

Todos os outros aspectos do ambiente de execução de shell são idênticos entre a função e quem a invoca, com as seguintes exceções: os coletores `DEBUG` e `RETURN` não são herdados, a menos que a função tenha recebido o atributo `trace` utilizando o comando interno `declare` ou a opção `-o functrace` tenha sido habilitada com o comando interno `set`, (caso no qual todas as funções herdam os coletores `DEBUG` e `RETURN`, o coletor `ERR` não é herdado, a menos que a opção de shell `-o errtrace` tenha sido habilitada. Veja-se Seção 4.1 [Comandos Internos do Shell Bourne], Página 44, para a descrição do comando interno `trap`.

A variável `FUNCNEST`, se configurada para um valor numérico maior que zero, define um nível máximo de aninhamento de função. As invocações de função que excederem o limite fazem com que o comando inteiro aborte.

Se o comando inteiro `return` for executado em uma função, então a função completa sua execução e reinicia com o próximo comando após a chamada de função. Qualquer comando associado com o coletor `RETURN` é executado antes que a função reinicie sua execução. Quando uma função completa sua execução, os valores dos parâmetros posicionais e o parâmetro especial '`#`' são restaurados para os valores que eles tinham antes da execução da função. Se um argumento numérico é dado ao `return`, então esse é o status de retorno da função; do contrário o status de retorno da função é o status de saída do último comando executado antes do `return`.

Variáveis locais para a função podem ser declaradas com o comando interno `local`. Essas variáveis são visíveis apenas para a função e aos comandos que a função invocar.

Os nomes de função e definições podem ser listados com a opção `-f` para o comando interno `declare` (`typeset`) (Seção 4.2 [Comandos Internos ao Bash], Página 52). A opção `-F` para `declare` ou `typeset` listará apenas os nomes de funções (e opcionalmente o arquivo fonte e número da linha, se a opção de shell `extdebug` estiver habilitada). As funções podem ser exportadas, de forma que sub shells automaticamente tenham elas definidas com a opção `-f` para o comando interno `export` (Seção 4.1 [Comandos Internos do Shell Bourne], Página 44). Perceba que as funções e variáveis de shell com o mesmo nome podem resultar em entradas múltiplas identicamente nomeadas no ambiente passadas para os filhos do shell. Cuidado deveria ser tomado em casos onde isso pode causar um problema.

As funções podem ser recursivas. A variável `FUNCNEST` pode ser utilizada para limitar a profundidade da pilha de chamada da função e restringir o número de invocações de função. Por padrão, nenhum limite é colocado no número de chamadas recursivas.

3.4 Parâmetros de Shell

Um *parameter* é uma entidade que armazena valores. Pode ser um `name`, um número, ou um dos caracteres especiais listados abaixo. Uma *variable* é um parâmetro denotado por um `name`. Uma variável tem um *value* e zero ou mais *attributes*. Atributos são atribuídos utilizando-se o comando interno `declare` (veja-se a descrição do comando interno `declare` em Seção 4.2 [Comandos Internos ao Bash], Página 52).

Um parâmetro é configurado se a ele foi atribuído um valor. A frase null é um valor válido. Uma vez que uma variável for configurada, ela pode ser desconfigurada apenas utilizando-se o comando interno `unset`.

Pode-se atribuir um valor a uma variável com uma declaração da forma

```
name=[value]
```

Se *value* não for dado, é atribuída para a variável a palavra null. Todos os *value*s estão sujeitos à expansão de til, expansão de parâmetro e variável, substituição de comando, expansão aritmética e remoção de aspas (detalhadas abaixo). Se a variável tiver seu atributo `integer` configurado, então *value* é calculado como uma expressão aritmética mesmo se a expansão `$((...))` não for utilizada (veja-se Seção 3.5.5 [Expansão Aritmética], Página 31). Divisão de palavra não é feita, com a exceção de `"$@"` conforme abaixo exposto. Expansão de nome de arquivo não é feita. Declarações de atribuição também podem aparecer como argumentos para os comandos internos `alias`, `declare`, `typeset`, `export`, `readonly`, e `local`. Quando em modo POSIX (veja-se Seção 6.11 [O Modo POSIX de Bash], Página 102), esse comandos internos podem aparecer em um comando após uma ou mais instâncias do comando interno `command` e conservar essas propriedades de declaração de atribuição.

No contexto onde uma declaração de atribuição está atribuindo um valor para uma variável de shell ou índice de vetor (veja-se Seção 6.7 [Vetores], Página 97), o operador '`+=`' pode ser utilizado ou para acrescentar ou para somar ao valor prévio da variável. Quando '`+=`' é aplicado a uma variável para a qual o atributo *integer* tenha sido configurado, *value* é calculado como uma expressão aritmética e adicionado ao valor atual da variável, o qual também é calculado. Quando '`+=`' é aplicado a uma variável de vetor utilizando-se atribuição composta (veja-se Seção 6.7 [Vetores], Página 97), o valor da variável não é desconfigurado (como o é ao se utilizar '`=`'), e novos valores são acrescentados ao vetor iniciando naquele valor maior que o índice máximo do vetor (para vetores indexados), ou somados como pares de valor-chave adicionais em um vetor associativo. Quando aplicado a uma variável de valor de sequência de caracteres, *value* é expandido e acrescentado ao valor da variável.

O atributo *nameref* de uma variável pode ser atribuído utilizando-se a opção `-n` para os comandos internos \fBdeclare\fP ou \fBlocal\fP (veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52) para criar uma *nameref*, ou uma referência a uma outra variável. Isso permite que variáveis sejam manipuladas indiretamente. Sempre que a variável nameref é referenciada ou é atribuída, a operação é atualmente feita na variável especificada pelo valor da variável nameref. Uma variável nameref é usualmente utilizada dentro de funções de shell para referenciar uma variável cujo nome é passado como um argumento para a função. Por exemplo, se um nome de variável é passado para uma função de shell como o seu primeiro argumento, executar

```
declare -n ref=$1
```

dentro da função cria uma variável nameref *ref* cujo valor é o nome da variável passado como o primeiro argumento. Referências e atribuições a *ref* são tratadas como referências e atribuições à variável cujo nome foi passado como `$1`.

Se a variável de controle em um loop `for` tem o atributo nameref, a lista de palavras pode ser uma lista de variáveis de shell, e uma referência de nome será estabelecida para cada palavra na lista, em sequência, quando o loop é executado. Às variáveis de vetores não pode ser atribuído o atributo `-n`. Entretanto, variáveis nameref podem referenciar variáveis de vetor e variáveis de vetor subscritas. Os nameref podem ser desconfigurados utilizando-se a opção `-n` para o comando interno `unset` (veja-se Seção 4.1 [Comandos Internos do Shell Bourne], Página 44). Do contrário, se `unset` for executado com o nome de uma variável nameref como um argumento, a variável referenciada pela variável nameref será desconfigurada.

3.4.1 Parâmetros Posicionais

Um *positional parameter* é um parâmetro denotado por um ou mais dígitos, qualquer outro diferente do dígito único `0`. Parâmetros posicionais são atribuídos via argumentos de shell quando esse é invocado, e podem ser reatribuídos utilizando-se o comando interno `set`. O parâmetro posicional `N` pode ser referenciado como `${N}`, ou como `$N` quando `N` consiste de um dígito único. Parâmetros posicionais não podem ser atribuídos com declarações de atribuição. Os comandos internos `set` e `shift` são utilizados para configurar e desconfigurar os parâmetros posicionais (veja-se Capítulo 4 [Comandos Internos ao Shell], Página 44). Os parâmetros posicionais são substituídos temporariamente quando uma função de shell é executada (veja-se Seção 3.3 [Funções de Shell], Página 18).

Quando um parâmetro posicional consistindo de mais que um dígito único é expandido, ele deve necessariamente estar contido entre chaves.

3.4.2 Parâmetros Especiais

O shell trata diversos parâmetros especialmente. Tais parâmetros apenas podem ser referenciados; atribuição a eles não é permitida.

`*` ($*) Expande para os parâmetros posicionais, iniciando de um. Quando a expansão não está dentro de aspas duplas, cada parâmetro posicional expande para uma palavra separada. Em contextos nos quais tal expansão é feita, as mencionadas palavras são objeto de separação mais ampla de palavras e de expansão de nome de caminho. Quando a expansão acontece dentro de aspas duplas, ela expande para uma palavra única com o valor de cada parâmetro separado pelo primeiro carácter da variável especial `IFS`. Isto é, `"$*"` é equivalente a `"$1`*c*`$2`*c*`..."`, onde *c* é o primeiro carácter do valor da variável `IFS`. Se `IFS` estiver desconfigurada, os parâmetros são separados por espaços. Se `IFS` for null, os parâmetros são unidos sem separadores intervenientes.

`@` ($@) Expande para os parâmetros posicionais, iniciando em um. Quando a expansão acontece dentro de aspas duplas, cada parâmetro expande para uma palavra separada. Isto é, `"$@"` é equivalente a `"$1" "$2" ...`. Se a expansão de aspas duplas ocorre dentro de uma palavra, a expansão do primeiro parâmetro é unido com a parte inicial da palavra original, e a expansão do último parâmetro é unido com a última parte da palavra original. Quando não existem parâmetros

posicionais, `"$@"` e `$@` expandem para nada (ou seja, os parâmetros posicionais são removidos).

`#` ($#) Expande para o número de parâmetros posicionais em decimal.

`?` ($?) Expande para o status de saída do canal de comunicação executado em primeiro plano mais recentemente.

`-` ($-, um hífen.) Expande para as flags de opção atuais conforme especificadas ao tempo da invocação, pelo comando interno `set`, ou aquelas configuradas pelo próprio shell (tal como a opção `-i`).

`$` ($$) Expande para o ID de processo do shell. Dentro de um sub shell `()`, ele expande para o ID de processo do shell que invoca, não o do sub shell.

`!` ($!) Expande para o ID de processo da tarefa colocada mais recentemente em segundo plano, seja executada como um comando assíncrono ou utilizando o comando interno `bg` (veja-se Seção 7.2 [Comandos Internos do Controle de Tarefa], Página 107).

`0` ($0) Expande para o nome do shell ou o do script de shell. Isso é configurado na inicialização do shell. Se Bash for invocado com um arquivo de comandos (veja-se Seção 3.8 [Scripts de Shell], Página 42), `$0` é configurado para o nome desse arquivo. Se Bash for iniciado com a opção `-c` (veja-se Seção 6.1 [Invocando o Bash], Página 87), então `$0` é configurado para o primeiro argumento após a sequência de caracteres ser executada, caso uma esteja presente. Do contrário, é configurado para o nome do arquivo utilizado para invocar Bash, conforme dado pelo argumento zero.

`_` ($_, um sublinhado.) Na inicialização do shell, configurado para o nome de caminho absoluto utilizado para invocar o shell ou script de shell sendo executado conforme passado no ambiente ou na lista de argumento. Subsequentemente, expande para o último argumento ao comando prévio, após expansão. Também configurado para o nome de caminho completo utilizado para invocar cada comando executado e colocado no ambiente exportado para esse comando. Quando da verificação de mensagem, esse parâmetro mantém o nome do arquivo de mensagem.

3.5 Expansões de Shell

A expansão é realizada na linha de comando após ela ter sido dividida em `token`s. Existem sete tipos de expansão realizadas:

- expansão de chave
- expansão de til
- expansão de parâmetro e variável
- substituição de comando
- expansão aritmética
- divisão de palavra
- expansão de nome de arquivo

A ordem das expansões é: expansão de chave; expansão de til, expansão de parâmetro e variável, expansão aritmética, e substituição de comando (feita da esquerda para a direita); divisão de palavra; e expansão de nome de arquivo.

Em sistemas que podem suportar, existe uma expansão adicional disponível: *process substitution*. Essa é realizada ao mesmo tempo que til, parâmetro, variável, e expansão aritmética e substituição de comando.

Apenas a expansão de chave, divisão de palavra e expansão de nome de arquivo podem mudar o número de palavras da expansão; outras expansões expandem a palavra única para uma palavra única. As únicas exceções a isso são as expansões de `"$@"` (veja-se Seção 3.4.2 [Parâmetros Especiais], Página 21) e `"${name[@]}"` (veja-se Seção 6.7 [Vetores], Página 97).

Após todas as expansões `quote removal` (veja-se Seção 3.5.9 [Remoção de Aspas], Página 34) é realizada.

3.5.1 Expansão de Chave

Expansão de chave é um mecanismo pelo qual sequências de caracteres arbitrárias podem ser geradas. Esse mecanismo é similar a *filename expansion* (veja-se Seção 3.5.8 [Expansão de Nome de Arquivo], Página 32), porém os nomes de arquivo gerados não podem existir. Padrões para serem expandidos de chave assumem a forma de um *preamble* opcional, seguido de ou uma série de sequências de caracteres separadas por vírgulas ou uma expressão sequencial entre um par de chaves, seguida por um *postscript* opcional. O preâmbulo é prefixado para cada sequência de caracteres contida nas chaves, e o postscript é então adicionado a cada sequência de caracteres resultante, expandindo da esquerda para a direita.

Expansões de chave podem ser aninhadas. Os resultados de cada sequência de caracteres expandida não são ordenados; a ordem da esquerda para a direita é preservada. Por exemplo,

```
bash$ echo a{d,c,b}e
ade ace abe
```

Uma expressão sequencial assume a forma `{x..y[..incr]}`, onde *x* e *y* são ou inteiros ou caracteres simples, e *incr*, um incremento opcional, é um inteiro. Quando inteiros são fornecidos, a expressão expande para cada número entre *x* e *y*, inclusive. Inteiros informados podem ser prefixados com '`0`' para forçar cada termo a ter o mesmo comprimento. Quando ou *x* ou *y* começam com um zero, o shell tenta forçar todos os termos gerados a conterem o mesmo número de dígitos, completando com zeros onde necessário. Quando caracteres são informados, a expressão expande para cada carácter lexicograficamente entre *x* e *y*, inclusive, utilizando o locale C padrão. Note que ambas *x* e *y* devem necessariamente ser do mesmo tipo. Quando o incremento é informado, ele é utilizado como a diferença entre cada termo. O incremento padrão é um (1) ou -1 conforme for apropriado.

Expansão de chave é realizada antes de quaisquer outras expansões, e quaisquer caracteres especiais para outras expansões são preservados no resultado. É estritamente textual. O Bash não aplica qualquer interpretação sintática ao contexto da expansão ou ao texto entre as chaves. Para evitar conflitos com a expansão de parâmetro, a sequência '`${`' não é considerada apta para expansão de chave.

Uma expansão de chave corretamente formada deve necessariamente conter chaves que abrem e fecham fora de aspas, e ao menos uma vírgula fora de aspas ou uma expressão sequencial válida. Qualquer expansão de chave incorreta é deixada não modificada.

Um { ou uma ‘,’ podem ser encapsuladas com uma barra invertida para prevenir que sejam considerados parte de uma expressão de chave. Para evitar conflitos com a expansão de parâmetro, a sequência ‘${’ não é considerada apta para expansão de chave.

Esta construção é tipicamente utilizada como abreviação quando o prefixo comum das sequências de caracteres a serem geradas é maior que no exemplo acima:

```
mkdir /usr/local/src/bash/{old,new,dist,bugs}
```

ou

```
chown root /usr/{ucb/{ex,edit},lib/{ex?.?*,how_ex}}
```

3.5.2 Expansão de Til

Se uma palavra se inicia com um carácter til fora de aspas (‘~’), todos os caracteres até a primeira barra fora de aspas (ou todos os caracteres, se não existir barra fora de aspas) são considerados como sendo um *tilde-prefix*. Se nenhum dos caracteres dentro do tilde-prefix estiver dentro de aspas, os caracteres dentro do tilde-prefix seguinte ao til são tratados como sendo um possível *login name*. Se esse nome de login for a sequência de caracteres `"null"`, então o til é substituído pelo valor da variável de shell `HOME`. Se `HOME` estiver desconfigurada, então o diretório home do usuário que executa o shell é substituído. Do contrário, o tilde-prefix é substituído pelo diretório home associado com o nome de login especificado.

Se o prefix-tilde for ‘`~+`’, então o valor da variável de shell `PWD` substitui o tilde-prefix. Se o tilde-prefix for ‘`~-`’, então o valor da variável de shell `OLDPWD`, se ele estiver configurado, é substituído.

Se os caracteres seguintes ao til dentro do tilde-prefix consistem de um número *N*, opcionalmente prefixado por um ‘`+`’ ou um ‘`-`’, então o tilde-prefix é substituído com o elemento correspondente originado da pilha de diretório, como se ele pudesse ser exibido pelo comando interno `dirs` invocado com os caracteres seguindo til no tilde-prefix como um argumento (veja-se Seção 6.8 [A Pilha de Diretório], Página 98). Se o tilde-prefix, sem o til, consiste de um número sem um ‘`+`’ ou ‘`-`’ iniciais, então ‘`+`’ é presumido.

Se o nome de login for inválido, ou a expansão de til falhar, a palavra é deixada imodificada.

Cada atribuição a variável é testada para tilde-prefixes fora de aspas imediatamente seguinte a ‘`:`’ ou o primeiro ‘`=`’. Nesse casos, a expansão de til também é feita. Consequentemente, pode-se utilizar nomes de arquivos com tios em atribuições a `PATH`, `MAILPATH`, e `CDPATH`, e o shell atribui o valor expandido.

A tabela seguinte mostra como Bash trata tilde-prefixes fora de aspas:

`~` O valor de `$HOME`

`~/foo` `$HOME/foo`

`~fred/foo`
O subdiretório `foo` do diretório home do usuário `fred`

`~+/foo` `$PWD/foo`

`~-/foo` `${OLDPWD-'~-'}/foo`

`~`*N* A sequência de caracteres que poderia ser exibida por ‘`dirs +`*N*’

`~+`*N* A sequência de caracteres que poderia ser exibida por ‘`dirs +`*N*’

`~-`*N* A sequência de caracteres que poderia ser exibida por ‘`dirs -`*N*’

3.5.3 Expansão de Parâmetro de Shell

O carácter '$' introduz a expansão de parâmetro, substituição de comando, ou expansão aritmética. O nome de parâmetro ou símbolo a ser expandido pode ser encapsulado entre chaves, as quais são opcionais, porém servem para proteger a variável a ser expandida dos caracteres imediatamente seguintes a ela, os quais poderiam ser interpretados como sendo parte do nome.

Quando chaves são utilizadas, a chave que finaliza e confere é o primeiro '}' não encapsulado por uma barra invertida ou contido em uma sequência de caracteres encapsulada por aspas, e não contida em uma expansão aritmética embutida, substituição de comando, ou expansão de parâmetro.

A forma básica da expansão de parâmetro é ${*parameter*}. O valor de *parameter* é substituído. O *parameter* é um parâmetro de shell conforme descrito acima (veja-se Seção 3.4 [Parâmetros de Shell], Página 20) ou uma referência de vetor (veja-se Seção 6.7 [Vetores], Página 97). As chaves são obrigatórias quando *parameter* é um parâmetro posicional com mais que um dígito, ou quando *parameter* for seguido por um carácter que não é para ser interpretado como sendo parte de seu nome.

Se o primeiro carácter de *parameter* for um ponto de exclamação (!), então isso introduz um nível de indireção de variável. Bash utiliza o valor da variável formada a partir do resto de *parameter* como sendo o nome da variável; essa variável é então expandida e aquele valor é utilizado no restante da substituição, em vez do valor do próprio *parameter*. Isso é conhecido como `indirect expansion`. As exceções a isso são as expansões de ${!*prefix**} e ${!*name*[@]} descritas abaixo. O ponto de exclamação deve necessariamente imediatamente seguir o abre chaves com o objetivo de introduzir a expansão indireta.

Em cada um dos casos abaixo, *word* é objeto de expansão de til, expansão de parâmetro, substituição de comando, e expansão aritmética.

Quando não for realizada expansão de substring, utilizando a forma descrita abaixo (por exemplo, ':-'), Bash testa se um parâmetro está desconfigurado ou é nulo. A omissão dos dois pontos resulta em um teste apenas para um parâmetro que está desconfigurado. Colocado de outra maneira, se os dois pontos forem incluídos, o operador verifica a existência de ambos os *parameter*s e se o seu valor não é nulo; se os dois pontos são omitidos, então o operador testa apenas a existência.

`${parameter:-word}`
: Se *parameter* estiver desconfigurado ou for nulo, a expansão de *word* é substituída. Do contrário, o valor de *parameter* é substituído.

`${parameter:=word}`
: Se *parameter* estiver desconfigurado ou for nulo, a expansão de *word* é atribuída a *parameter*. O valor de *parameter* é então substituído. Parâmetros posicionais e parâmetros especiais não podem ser atribuídos dessa maneira.

`${parameter:?word}`
: Se *parameter* for nulo ou estiver desconfigurado, a expansão de *word* (ou uma mensagem para esse efeito se *word* não estiver presente) é escrita para o descritor de erro padrão e o shell, se não for interativo, sai. Do contrário, o valor de *parameter* é substituído.

`${parameter:+word}`
: Se *parameter* for nulo ou estiver desconfigurado, nada é substituído, do contrário a expansão de *word* é substituída.

`${parameter:offset}`
`${parameter:offset:length}`
: Isso é referenciado como Expansão de Substring. Ela expande até *length* caracteres do valor de *parameter* iniciando no carácter especificado por *offset*. Se *parameter* for '`@`', um vetor indexado subscrito por '`@`' ou '`*`', ou um nome de vetor associativo, os resultados diferem conforme descritos abaixo. Se *length* for omitido, expande para a substring do valor de *parameter* iniciando no carácter especificado por *offset* e se estendo até o fim do valor. *length* e *offset* são expressões aritméticas (veja-se Seção 6.5 [Aritmética de Shell], Página 95).

  Se o valor de *offset* for calculado para um número menor que zero, o valor é utilizado como um offset em caracteres a partir do fim do valor de *parameter*. Se o valor de *length* for calculado para um número menor que zero, ela é interpretada como sendo um offset em caracteres a partir do fim do valor de *parameter* em vez de um número de caracteres, e a expansão são os caracteres entre *offset* e aquele resultado. Perceba-se que um offset negativo deve necessariamente ser separado dos dois pontos por pelo menos um espaço para evitar que seja confundido com a expansão de '`:-`'.

  Aqui estão alguns exemplos ilustrando a Expansão de Substring sobre parâmetros e vetores subscritos:

```
$ string=01234567890abcdefgh
$ echo ${string:7}
7890abcdefgh
$ echo ${string:7:0}

$ echo ${string:7:2}
78
$ echo ${string:7:-2}
7890abcdef
$ echo ${string: -7}
bcdefgh
$ echo ${string: -7:0}

$ echo ${string: -7:2}
bc
$ echo ${string: -7:-2}
bcdef
$ set -- 01234567890abcdefgh
$ echo ${1:7}
7890abcdefgh
$ echo ${1:7:0}

$ echo ${1:7:2}
78
```

```
$ echo ${1:7:-2}
7890abcdef
$ echo ${1: -7}
bcdefgh
$ echo ${1: -7:0}

$ echo ${1: -7:2}
bc
$ echo ${1: -7:-2}
bcdef
$ array[0]=01234567890abcdefgh
$ echo ${array[0]:7}
7890abcdefgh
$ echo ${array[0]:7:0}

$ echo ${array[0]:7:2}
78
$ echo ${array[0]:7:-2}
7890abcdef
$ echo ${array[0]: -7}
bcdefgh
$ echo ${array[0]: -7:0}

$ echo ${array[0]: -7:2}
bc
$ echo ${array[0]: -7:-2}
bcdef
```

Se *parameter* for '`@`', o resultado é *length* parâmetros posicionais iniciando em *offset*. Um *offset* negativo é tomado relativo a um maior que o maior parâmetro posicional, de maneira que um offset de -1 tem por resultado o último parâmetro posicional. É um erro de expansão se *length* tiver por resultado um número menor que zero.

Os exemplos seguintes ilustram Expansão de Substring utilizando parâmetros posicionais:

```
$ set -- 1 2 3 4 5 6 7 8 9 0 a b c d e f g h
$ echo ${@:7}
7 8 9 0 a b c d e f g h
$ echo ${@:7:0}

$ echo ${@:7:2}
7 8
$ echo ${@:7:-2}
bash: -2: substring expression < 0
$ echo ${@: -7:2}
b c
$ echo ${@:0}
```

```
./bash 1 2 3 4 5 6 7 8 9 0 a b c d e f g h
$ echo ${@:0:2}
./bash 1
$ echo ${@: -7:0}
```

Se *parameter* for um nome de vetor indexado subscrito por '`@`' ou '`*`', o resultado é o *length* de membros do vetor iniciando com `${parameter[offset]}`. Um *offset* negativo é tomado relativo ao um maior que o índice máximo do vetor especificado. É um erro de expansão se *length* for calculado para um número menor que zero.

Estes exemplos mostram como se pode utilizar Expansão de Substring com vetores indexados:

```
$ array=(0 1 2 3 4 5 6 7 8 9 0 a b c d e f g h)
$ echo ${array[@]:7}
7 8 9 0 a b c d e f g h
$ echo ${array[@]:7:2}
7 8
$ echo ${array[@]: -7:2}
b c
$ echo ${array[@]: -7:-2}
bash: -2: substring expression < 0
$ echo ${array[@]:0}
0 1 2 3 4 5 6 7 8 9 0 a b c d e f g h
$ echo ${array[@]:0:2}
0 1
$ echo ${array[@]: -7:0}
```

Expansão de Substring aplicada a um vetor associativo produz resultados indefinidos.

A indexação de substring é baseada em zero, a menos que os parâmetros posicionais sejam utilizados, caso no qual a indexação começa no um (1) por padrão. Se *offset* for zero (0), e os parâmetros posicionais forem utilizados, `$@` é prefixado à lista.

`${!`*prefix*`*}`
`${!`*prefix*`@}`
: Expande para os nomes das variáveis cujos nomes iniciam com *prefix*, separados pelo primeiro carácter da variável especial `IFS`. Quando '`@`' for utilizada e a expansão aparece dentro de aspas duplas, cada nome de variável expande para uma palavra separada.

`${!`*name*`[@]}`
`${!`*name*`[*]}`
: Se *name* for uma variável de vetor, expande para a lista de índices de vetor (chaves) atribuídos em *name*. Se *name* não for um vetor, expande para zero (0) se *name* estiver configurada e nulo caso contrário. Quando '`@`' for utilizada

e a expansão aparecer dentro de aspas duplas, cada chave expande para uma palavra separada.

`${#parameter}`
O tamanho em caracteres do valor expandido de *parameter* é substituído. Se *parameter* for '`*`' ou '`@`', o valor substituído é o número de parâmetros posicionais. Se *parameter* for um nome de vetor subscrito por '`*`' ou '`@`', o valor substituído é o número de elementos no vetor. Se *parameter* for um nome de vetor indexado subscrito por um número negativo, esse número é interpretado como relativo a um maior que o índice máximo de *parameter*, de maneira que índices negativos contam de volta do fim do vetor, e um índice de -1 referencia o último elemento.

`${parameter#word}`
`${parameter##word}`
A *word* é expandida para produzir um padrão exatamente como em expansão de nome de arquivo (veja-se Seção 3.5.8 [Expansão de Nome de Arquivo], Página 32). Se o padrão coincidir com o inicio do valor expandido de *parameter*, então o resultado da expansão é o valor expandido de *parameter* com o padrão mais curto que coincidir (o caso do '`#`') ou o padrão mais longo que coincidir (o caso do '`##`') deletado. Se *parameter* for '`@`' ou '`*`', a operação de remoção do padrão é aplicada a cada parâmetro posicional em sequência, e a expansão é a lista resultante. Se *parameter* for uma variável de vetor subscrito com '`@`' ou '`*`', a operação de remoção de padrão é aplicada a cada membro do vetor em sequência, e a expansão é a lista resultante.

`${parameter%word}`
`${parameter%%word}`
A *word* é expandida para produzir um padrão exatamente como na expansão de nome de arquivo. Se o padrão coincidir com uma porção ao final do valor expandido de *parameter*, então o resultado da expansão é o valor de *parameter* com o padrão mais curto que coincidir (o caso do '`%`') ou o padrão mais longo que coincidir (o caso do '`%%`') deletado. Se *parameter* for '`@`' ou '`*`', a operação de remoção de padrão é aplicada a cada parâmetro posicional em sequência, e a expansão é a lista resultante. Se *parameter* for uma variável de vetor subscrito com '`@`' ou '`*`', a operação de remoção de padrão é aplicada a cada membro do vetor em sequência, e a expansão é a lista resultante.

`${parameter/pattern/string}`
O *pattern* é expandido para produzir um padrão exatamente como em expansão de nome de arquivo. *Parameter* é expandido e a coincidência mais longa de *pattern* contra seu valor é substituída com *string*. Se *pattern* iniciar com uma '`/`', todas as coincidências de *pattern* são substituídas com *string*. Normalmente apenas a primeira coincidência é substituída. Se *pattern* se iniciar com '`#`', é necessário coincidir no início do valor expandido de *parameter*. Se *pattern* se iniciar com '`%`', é necessário coincidir no final do valor expandido de *parameter*. Se *string* for nula, as coincidências de *pattern* são deletadas e a `/` que se seguir a *pattern* pode ser omitida. Se *parameter* for '`@`' ou '`*`', a operação de substituição é aplicada a cada parâmetro posicional em sequência, e a expansão é a lista

resultante. Se *parameter* for uma variável de vetor subscrito com '`@`' ou '`*`', a operação de substituição é aplicada a cada parâmetro posicional em sequência, e a expansão é a lista resultante. Se *parameter* for uma variável de vetor subscrito com '`@`' ou '`*`', a operação de substituição é aplicada a cada membro do vetor em sequência, e a expansão é a lista resultante.

`${`*parameter*`^`*pattern*`}`
`${`*parameter*`^^`*pattern*`}`
`${`*parameter*`,`*pattern*`}`
`${`*parameter*`,,`*pattern*`}`

Esta expansão modifica o tipo dos caracteres alfabéticos em *parameter*. O *parameter* é expandido para produzir um padrão exatamente como na expansão de nome de arquivo. Cada carácter no valor expandido de *parameter* é testado contra *pattern*, e, se coincidir com o padrão, acontece a conversão de caso. O padrão não deveria tentar coincidir mais que um carácter. O operador '`^`' converte letras minúsculas coincidentes com *pattern* para maiúsculas; o operador '`,`' converte letras maiúsculas coincidentes com *pattern* para minúsculas. As expansões '`^^`' e '`,,`' convertem cada carácter coincidido no valor expandido; as expansões '`^`' e '`,`' coincidem e convertem apenas o primeiro carácter no valor expandido. Se *pattern* for omitido, ele é tratado como um '`?`', o qual coincide com cada carácter. Se *parameter* for '`@`' ou '`*`', a operação de modificação de caso é aplicada a cada parâmetro posicional em sequência, e a expansão é a lista resultante. Se *parameter* for uma variável de vetor subscrito com '`@`' ou '`*`', a operação de modificação de caso é aplicada a cada membro do vetor em sequência, e a expansão é a lista resultante.

3.5.4 Substituição de Comando

A substituição de comando permite que a saída de um comando substitua o próprio comando. A substituição de comando ocorre quando um comando é encapsulado conforme segue:

`$(`*command*`)`

ou

`` `command` ``

Bash realiza a expansão executando *command* e substituindo a substituição de comando com a saída padrão do comando, com cada marcador de nova linha sendo deletado. Os marcadores de nova linha embutidos não são deletados, porém eles podem ser removidos durante a divisão de palavra. A substituição de comando `$(cat` *file*`)` pode ser substituída pelo equivalente, mas mais rápido `$(<` *file*`)`.

Quando a forma de substituição estilo antigo de aspa invertida é utilizada, a barra invertida mantém o seu significado literal exceto quando seguido por '`$`', '`` ` ``', ou '`\`'. A primeira aspa invertida não precedida por uma barra invertida termina a substituição de comando. Quando da utilização da forma `$(`*command*`)`, todos os caracteres entre os parênteses compõem o comando; nenhum é tratado especialmente.

As substituições de comando podem ser aninhadas. Para aninhar quando da utilização da forma de aspa invertida, encapsule as aspas invertidas internas com barras invertidas.

Se a substituição aparece dentro de aspas duplas, divisão de palavra e expansão de nome de arquivo não são realizadas sobre os resultados.

3.5.5 Expansão Aritmética

A expansão aritmética permite o cálculo de uma expressão aritmética e a substituição do resultado. O formato para expansão aritmética é:

```
$(( expression ))
```

A expressão é tratada como se ela estivesse contida entre aspas duplas, porém uma aspa dupla dentro de parênteses não é tratada especialmente. Todos os tokens na expressão estão sujeitos a expansão de parâmetro e variável, substituição de comando, e remoção de aspas. O resultado é tratado como a expressão aritmética a ser calculada. Expansões aritméticas podem ser aninhadas.

O cálculo é realizado de acordo com as regras listadas abaixo (veja-se Seção 6.5 [Aritmética de Shell], Página 95). Se a expressão for inválida, Bash imprime uma mensagem indicando a falha para o descritor de erro padrão e não ocorre substituição.

3.5.6 Substituição de Processo

A substituição de processo é suportada em sistemas que suportam tubos nomeados (FIFOs) ou o método `/dev/fd` de nomear arquivos abertos. A substituição de processo assume a forma de

```
<(list)
```

ou

```
>(list)
```

A *list* de processo é executada com a entrada ou saída dela conectada ao FIFO ou algum arquivo em `/dev/fd`. O nome desse arquivo é passado como um argumento para o comando atual como resultado da expansão. Se a forma `>(list)` for utilizada, então a escrita para o arquivo fornecerá entrada para *list*. Se a forma `<(list)` for utilizada, o arquivo passado como um argumento deveria ser lido para obter a saída de *list*. Perceba-se que nenhum espaço pode aparecer entre o `<` ou `>` e o abre parênteses, do contrário a construção seria interpretada como sendo uma redireção.

Quando disponível, a substituição de processo é realizada simultaneamente com a expansão de parâmetro e variável, substituição de comando, e expansão aritmética.

3.5.7 Divisão de Palavra

O shell escaneia os resultados da expansão de parâmetro, substituição de comando, e expansão aritmética que não ocorreu dentro de aspas duplas para divisão de palavra.

O shell trata cada carácter de `$IFS` como sendo um delimitador, e reparte os resultado das outras expansões em palavras utilizando esses caracteres como terminadores de campo. Se `IFS` estiver desconfigurada, ou o valor dela for exatamente `<space><tab><newline>`, o padrão, então as sequências de `<space>`, `<tab>`, e `<newline>` no início e final dos resultados das expansões prévias são ignoradas, e quaisquer sequências de `IFS` caracteres que não estejam no início ou no final servem para delimitar palavras. Se `IFS` tiver um outro valor qualquer diferente do padrão, então as sequências de caracteres de espaço em branco `space` e `tab` são ignoradas no início e no final da palavra, até onde o carácter espaço em branco

esteja no valor de `IFS` (um carácter de espaço em branco `IFS`). Qualquer carácter em `IFS` que não um espaço em branco `IFS`, junto com quaisquer caracteres espaço em branco `IFS` adjacentes, delimitam o campo. Uma sequência de caracteres espaço em branco `IFS` também é tratada como um delimitador. Se o valor de `IFS` for nulo, não ocorre a divisão de palavra.

Os argumentos "null" explícitos (`""` or `''`) são mantidos. Os argumentos "null" implícitos fora de aspas, resultantes da expansão de parâmetros que não tem valores, são removidos. Se um parâmetro sem um valor for expandido dentro de aspas duplas, um argumento "null" resulta e é mantido.

Note-se que se não ocorrer expansão, divisão não é realizada.

3.5.8 Expansão de Nome de Arquivo

Após a divisão de palavra, a menos que a opção `-f` tenha sido configurada (veja-se Seção 4.3.1 [O Comando Interno Set], Página 63), Bash escaneia cada palavra a procura dos caracteres '`*`', '`?`', e '`[`'. Se um desses caracteres aparece, então a palavra é considerada como sendo um *pattern*, e substituída com uma lista de nomes de arquivos ordenada alfabeticamente coincidindo com o padrão (veja-se Seção 3.5.8.1 [Coincidência de Modelo], Página 32). Se nomes de arquivo que coincidam não forem encontrados, e a opção de shell `nullglob` estiver desabilitada, então a palavra é deixada imodificada. Se a opção `nullglob` estiver configurada, e nenhuma coincidência for encontrada, então a palavra é removida. Se a opção de shell `failglob` estiver configurada, e nenhuma coincidência for encontrada, uma mensagem de erro é impressa e o comando não é executado. Se a opção de shell `nocaseglob` estiver habilitada, o teste de coincidência é tentado sem considerar maiúsculas e minúsculas.

Quando um padrão é utilizado para expansão de nome de arquivo, o carácter '`.`' no início de um nome de arquivo ou imediatamente seguinte a uma barra deve necessariamente ser coincidido explicitamente, a menos que a opção de shell `dotglob` esteja configurada. Quando da coincidência de um nome de arquivo, o carácter barra deve necessariamente sempre ser coincidido explicitamente. Em outros casos, o carácter '`.`' não é tratado especialmente.

Veja-se a descrição de `shopt` na Seção 4.3.2 [O Comando Interno Shopt], Página 68, para uma descrição das opções `nocaseglob`, `nullglob`, `failglob`, e `dotglob`.

A variável de shell `GLOBIGNORE` pode ser utilizada para restringir o conjunto de nomes de arquivo coincidentes com um padrão. Se `GLOBIGNORE` estiver configurada, cada nome de arquivo coincidente que também coincidir com um dos padrões em `GLOBIGNORE` é removido da lista de coincidências. Os nomes de arquivos `.` e `..` sempre são ignorados quando `GLOBIGNORE` estiver configurada e não for nula. Entretanto, a configuração de `GLOBIGNORE` para um valor não nulo tem o efeito de habilitar a opção de shell `dotglob`, de maneira que todos os outros nomes de arquivos iniciando com um '`.`' coincidirão. Para se ter o comportamento antigo de ignorar nomes de arquivos iniciando com um '`.`', torne-se '`.*`' um dos padrões em `GLOBIGNORE`. A opção `dotglob` é desabilitada quando `GLOBIGNORE` estiver desconfigurada.

3.5.8.1 Coincidência de Modelo

Qualquer carácter que apareça em um padrão, que não os caracteres especiais de padrão descritos abaixo, coincidem com eles próprios. O carácter NULL pode não ocorrer em um

padrão. Uma barra invertida encapsula o carácter seguinte a ela; a barra invertida de encapsulamento é descartada quando da coincidência. Os caracteres especiais de padrão devem necessariamente serem encapsulados em aspas se eles são para serem coincididos literalmente.

Os caracteres especiais de padrão tem os significados seguintes:

`*` Coincide com qualquer sequência de caracteres, incluindo a sequência de caracteres "null". Quando a opção de shell `globstar` está habilitada, e '`*`' for utilizado em um contexto de expansão de nome de arquivo, dois '`*`'s adjacentes utilizados como um padrão único coincidirá com todos os arquivos e zero ou mais diretórios e subdiretórios. Se seguido por uma '`/`', dois '`*`'s adjacentes coincidirão apenas com diretórios e subdiretórios.

`?` Coincide com qualquer carácter singular.

`[...]` Coincide com qualquer um dos caracteres encapsulados. Um par de caracteres separados por um hífen denota uma *range expression*; qualquer carácter que cair entre esses dois caracteres, inclusive, utilizando o conjunto de caracteres e sequência de recolha do locale atual, é coincidida. Se o primeiro carácter seguinte a '`[`' for um '`!`' ou um '`^`', então qualquer carácter não encapsulado é coincidido. Um '−' pode ser coincidido incluindo-se ele como o primeiro ou último carácter no conjunto. Um '`]`' pode ser coincidido incluindo-se ele como o primeiro carácter no conjunto. A ordem de arranjo dos caracteres em expressões de intervalo é determinada pelo locale atual e os valores das variáveis de shell `LC_COLLATE` e `LC_ALL`, se configuradas.

Por exemplo, no locale C padrão, '`[a-dx-z]`' é equivalente a '`[abcdxyz]`'. Muitos locales ordenam caracteres na ordem do dicionário, e em tais locales '`[a-dx-z]`' é tipicamente não equivalente a '`[abcdxyz]`'; poderia ser equivalente a '`[aBbCcDdxXyYz]`', por exemplo. Para se obter a interpretação de intervalos tradicional em expressões de parêntese, pode-se forçar o uso do locale C configurando-se a variável de ambiente `LC_COLLATE` ou `LC_ALL` para o valor '`C`', ou habilitar a opção de shell `globasciiranges`. Dentro de '`[`' e '`]`', *character classes* podem ser especificadas utilizando-se a sintaxe `[:`*class*`:]`, onde *class* é uma das classes seguintes definidas no padrão POSIX:

```
alnum   alpha   ascii   blank   cntrl   digit   graph   lower
print   punct   space   upper   word    xdigit
```

Uma classe de caracteres coincide com qualquer carácter pertencente àquela classe. A classe de carácter `word` coincide com letras, dígitos, e com o carácter '`_`'.

Dentro de '`[`' e '`]`', uma *equivalence class* pode ser especificada utilizando-se a sintaxe `[=`*c*`=]`, a qual coincide com todos os caracteres com o mesmo peso de recolha (conforme definido pelo locale atual) como o carácter *c*.

Dentro de '`[`' e '`]`', a sintaxe `[.`*symbol*`.]` coincide com o símbolo de recolha *symbol*.

Se a opção de shell `extglob` for habilitada utilizando o comando interno `shopt`, então vários operadores estendidos de padrões de coincidência são reconhecidos. Na descrição

seguinte, uma *pattern-list* é uma lista de um ou mais padrões separados por um '|'. Padrões de composição podem serem formados utilizando-se um ou mais dos sub-padrões seguintes:

`?(`*pattern-list*`)`
: Coincide com zero ou uma ocorrência dos padrões dados.

`*(`*pattern-list*`)`
: Coincide com zero ou mais ocorrências dos padrões dados.

`+(`*pattern-list*`)`
: Coincide com uma ou mais ocorrências dos padrões dados.

`@(`*pattern-list*`)`
: Coincide com um dos padrões dados.

`!(`*pattern-list*`)`
: Coincide com qualquer coisa, exceto um dos padrões dados.

3.5.9 Remoção de Aspas

Após as expansões precedentes, todas as ocorrências não aspeadas dos caracteres '\', '', e '"' que não resultaram de uma das expansões acima são removidos.

3.6 Redireções

Antes que um comando seja executado, a entrada e a saída dele podem ser *redirected* utilizando-se uma notação especial interpretada pelo shell. A redireção permite que manipuladores de arquivo de comandos sejam duplicados, abertos, fechados, construídos para referenciar arquivos diferentes, e podem modificar os arquivos a partir dos quais o comando lê ou para os quais escreve. A redireção também pode ser utilizada para modificar manipuladores de arquivos no ambiente de execução do shell atual. Os seguintes operadores de redireção podem preceder ou aparecer em qualquer lugar dentro de um comando simples ou podem se seguir a um comando. As redireções são processadas na ordem em que elas aparecem, desde a esquerda até a direita.

Cada redireção que pode ser precedida por um número de descritor de arquivo pode ao invés ser precedida por uma palavra da forma {*varname*}. Nesse caso, para cada operador de redireção, exceto >&- e <&-, o shell alocará um descritor de arquivo maior que dez (10) e o atribuirá a {*varname*}. Se >&- ou <&- forem precedidos por {*varname*}, então o valor de *varname* define o descritor de arquivo a fechar.

Nas descrições seguintes, se o número do descritor de arquivo for omitido, e o primeiro carácter do operador de redireção for '<', então a redireção se refere à entrada padrão (descritor de arquivo 0). Se o primeiro carácter do operador de redireção for '>', então a redireção se refere à saída padrão (descritor de arquivo 1).

A palavra que se segue ao operador de redireção nas descrições seguintes, a menos que apontado de outra maneira, está sujeita à expansão de chave, expansão de til, expansão de parâmetro, substituição de comando, expansão aritmética, remoção de aspas, expansão de nome de arquivo e divisão de palavra. Se ela expandir para mais que uma palavra, então Bash reportará um erro.

Perceba que a ordem das redireções é importante. Por exemplo, o comando

```
ls > dirlist 2>&1
```

direciona ambos saída padrão (descritor de arquivo 1) e erro padrão (descritor de arquivo 2) para o arquivo *dirlist*, ao passo que o comando

```
ls 2>&1 > dirlist
```

direciona apenas a saída padrão para o arquivo *dirlist*, pois o erro padrão foi feito como que uma cópia da saída padrão antes que a saída padrão fosse redirecionada para *dirlist*.

Bash manipula diversos nomes de arquivos de forma especial quando tais arquivos são utilizados em redireções, conforme descrito na tabela seguinte:

`/dev/fd/`*fd*
: Se *fd* for um número inteiro válido, o descritor de arquivo *fd* é duplicado.

`/dev/stdin`
: O descritor de arquivo zero (0) é duplicado.

`/dev/stdout`
: O descritor de arquivo um (1) é duplicado.

`/dev/stderr`
: O descritor de arquivo dois (2) é duplicado.

`/dev/tcp/`*host*`/`*port*
: Se *host* for um nome de máquina válido ou um endereço de Internet, e *port* for um número inteiro de porta ou um nome de serviço, então Bash tenta abrir o soquete TCP correspondente.

`/dev/udp/`*host*`/`*port*
: Se *host* for um nome de máquina válido ou um endereço Internet, e *port* for um número inteiro de porta ou um nome de serviço, então Bash tenta abrir o soquete UDP correspondente.

Uma falha em abrir ou em criar um arquivo faz com que a redireção falhe.

Redireções que utilizem descritores de arquivos maiores que nove (9) deveriam ser utilizados com cuidado, uma vez que eles podem conflitar com descritores de arquivo que o shell utiliza internamente.

3.6.1 Redirecionando Entrada

A redireção de entrada faz com que o arquivo cujo nome resulte da expansão de *word* seja aberto para leitura no descritor de arquivo `n`, ou na entrada padrão (descritor de arquivo zero (0)) se `n` não for especificado.

O formato geral para redirecionar entrada é:

```
[n]<word
```

3.6.2 Redirecionando Saída

A redireção de saída faz com que o arquivo cujo nome resulte da expansão de *word* seja aberto para escrita no descritor de arquivo *n*, ou na saída padrão (descritor de arquivo um (1)) se *n* não for especificado. Se o arquivo não existir, ele é criado; se ele já existir, ele é truncado para tamanho zero.

O formato geral para redirecionar saída é:

`[n]>[|]word`

Se o operadores de redirecionamento for '`>`', e a opção `noclobber` para o comando interno `set` tiver sido habilitada, então a redireção falhará se o arquivo cujo nome resulte da expansão de *word* existir e for um arquivo regular. Se o operador de redireção for '`>|`', ou o operador de redireção for '`>`' e a opção `noclobber` não estiver habilitada, então a redireção é tentada, mesmo se o arquivo nomeado por *word* existir.

3.6.3 Adicionando Saída Redirecionada

A redireção da saída nesta maneira faz com que o arquivo cujo nome resulte da expansão de *word* seja aberto, para acrescentar conteúdo, no descritor de arquivo *n*, ou na saída padrão (descritor de arquivo um (1)) caso *n* não seja especificado. Se o arquivo não existir, então ele é criado.

O formato geral para acrescentar conteúdo de saída é:

`[n]>>word`

3.6.4 Redirecionando a Saída Padrão e o Erro Padrão

Este construtor permite que ambas, a saída padrão (descritor de arquivo um (1)) e a saída de erro padrão (descritor de arquivo dois (2)), sejam redirecionadas para o arquivo cujo nome é a expansão de *word*.

Existem dois formatos para o redirecionamento da saída padrão e do erro padrão:

`&>word`

e

`>&word`

Das duas formas, a primeira é a preferida. Isso é semanticamente equivalente a:

`>word 2>&1`

Quando se utilizar a segunda forma, *word* pode não expandir para um número ou '`-`'. Se expandir, outros operadores de redireção se aplicam (veja-se Duplicando Descritores de Arquivo abaixo) por razões de compatibilidade.

3.6.5 Adicionando a Saída Padrão e o Erro Padrão

Este construtor permite que ambas, a saída padrão (descritor de arquivo um (1)) e a saída de erro padrão (descritor de arquivo dois (2)), sejam acrescentadas ao arquivo cujo nome é a expansão de *word*.

O formato para acrescentar a saída padrão e o erro padrão é:

`&>>word`

Isso é semanticamente equivalente a:

`>>word 2>&1`

(veja-se Duplicando Descritores de Arquivo abaixo).

3.6.6 Documentos Here

Este tipo de redireção instrui o shell a ler entrada vinda da fonte atual até que uma linha contendo apenas *word* (sem espaços em branco seguintes) seja vista. Todas as linhas lidas até tal ponto são então utilizadas como entrada padrão para um comando.

O formato de here-documents é:

```
<<[-]word
        here-document
delimiter
```

Nenhuma expansão de variável e de parâmetro, substituição de comando, expansão aritmética ou expansão de nome de arquivo é efetuada sobre *word*. Se quaisquer caracteres em *word* estiverem entre aspas, o *delimiter* é o resultado da remoção das aspas sobre *word*, e as linhas no documento-aqui não são expandidas. Se *word* estiver fora de aspas, todas as linhas do documento-aqui ficam sujeitas a expansão de parâmetro, substituição de comando, e expansão aritmética, a sequência de carácter `\newline` é ignorada, e '\' deve necessariamente ser utilizado para encapsular os caracteres '\', '$', e '`'.

Se o operador de redireção for '`<<-`', então todos os caracteres tab iniciais são retirados das linhas de entrada e da linha que contém o *delimiter*. Isso permite que documentos-aqui dentro de shell scripts sejam endentados em uma forma natural.

3.6.7 Sequências de Caracteres Here

Uma variante dos documentos aqui, o formato é:

```
<<< word
```

A *word* está sujeita a expansão de chave, expansão de til, expansão de variável e de parâmetro, substituição de comando, expansão aritmética, e remoção de aspas. A expansão de nome de caminho e a divisão de palavra não são efetuadas. O resultado é fornecido como uma sequência de caracteres única para o comando na entrada padrão dele.

3.6.8 Duplicando Descritores de Arquivos

O operador de redireção

```
[n]<&word
```

é utilizado para duplicar descritores de arquivo de entrada. Se *word* expandir para um ou mais dígitos, o descritor de arquivo denotado por *n* é feito como que uma cópia desse descritor de arquivo. Se os dígitos em *word* não especificarem um descritor de arquivo aberto para entrada, um erro de redireção ocorre. Se *word* resultar em '`-`', então o descritor de arquivo *n* é fechado. Se *n* não for especificado, então a entrada padrão (descritor de arquivo zero (0)) é utilizado.

O operador

```
[n]>&word
```

é utilizado similarmente para duplicar descritores de arquivo de saída. Se *n* não for especificado, então a saída padrão (descritor de arquivo um (1)) é utilizado. Se os dígitos em *word* não especificarem um descritor de arquivo aberto para saída, então um erro de redireção ocorre. Se *word* resultar em '`-`', então o descritor de arquivo *n* é fechado. Como um caso especial, se *n* for omitido, e *word* não expandir para um ou mais dígitos ou '`-`', então a saída padrão e o erro padrão são redirecionados conforme descrito previamente.

3.6.9 Movendo Descritores de Arquivos

O operador de redireção

```
[n]<&digit-
```

move o descritor de arquivo *digit* para o descritor de arquivo *n*, ou a entrada padrão (descritor de arquivo zero (0)), se *n* não for especificado. *digit* é fechado após ser duplicado em *n*.

Similarmente, o operador de redireção

```
[n]>&digit-
```

move o descritor de arquivo *digit* para descritor de arquivo *n*, ou a saída padrão (descritor de arquivo um (1)) se *n* não for especificado.

3.6.10 Abrindo Descritores de Arquivos para Leitura e Escrita

O operador de redireção

```
[n]<>word
```

faz com que o arquivo cujo nome é a expansão de *word* seja aberto tanto para leitura quanto para escrita no descritor de arquivo *n*, ou no descritor de arquivo zero (0) se *n* não for especificado. Se o arquivo não existir, ele é criado.

3.7 Execução de Comandos

3.7.1 Expansão de Comando Simples

Quando um comando simples é executado, o shell executa as seguintes expansões, atribuições, e redireções, da esquerda para a direita.

1. As palavras que o interpretador já marcou como atribuições de variáveis (aquelas precedendo o nome do comando) e redireções são salvas para processamento posterior.
2. As palavras que não são atribuições de variável ou redireções são expandidas (Seção 3.5 [Expansões de Shell], Página 22). Se quaisquer palavras permanecerem após a expansão, a primeira palavra é considerada como sendo o nome do comando e as palavras restantes são os argumentos.
3. As redireções são implementadas conforme descrito acima (Seção 3.6 [Redireções], Página 34).
4. O texto após o sinal '=' em cada atribuição de variável está submetido a expansão de til, expansão de parâmetro, substituição de comando, expansão aritmética e remoção de aspas antes de ser atribuída para a variável.

Se não houver um nome de comando como resultado, as atribuições de variável afetam o ambiente de shell atual. Do contrário, as variáveis são adicionadas ao ambiente do comando executado e não afetam o ambiente de shell atual. Se quaisquer das atribuições tenta atribuir um valor para uma variável configurada como apenas leitura, um erro ocorre e o comando sai com um código de saída diferente de zero.

Se não houver um nome de comando como resultado, as redireções são implementadas, porém não afetam o ambiente shell atual. Um erro de redireção faz com que o comando saia com um código de saída diferente de zero.

Se existir um nome de comando deixado após a expansão, a execução procede conforme descrito abaixo. Do contrário, o comando sai. Se uma das expansões continha uma substituição de comando, o código de saída do comando é o código de saída da última substituição implementada. Se não houve substituições de comandos, o comando sai com um código de saída igual a zero.

3.7.2 Busca de Comando e Execução

Após um comando ter sido dividido em palavras, se ele resultar em um comando simples e uma lista opcional de argumentos, as seguintes ações são tomadas.

1. Se o nome do comando não contiver barras, então o shell tenta localizá-lo. Se existir uma função de shell para aquele nome, então aquela função é invocada conforme descrito em Seção 3.3 [Funções de Shell], Página 18.
2. Se o nome não coincidir com o de uma função, então o shell busca por ele na lista de comando internos ao shell. Se uma coincidência for encontrada, então aquele comando interno é invocado.
3. Se o nome não for nem uma função de shell, nem um comando interno, e nem contiver barras, então Bash pesquisa em cada elemento de `$PATH` procurando por um diretório que contenha um arquivo executável para aquele nome. O Bash utiliza uma tabela de hash para lembrar os nomes completos de caminhos dos arquivos executáveis para evitar múltiplas buscas no `$PATH` (veja-se a descrição de `hash` em Seção 4.1 [Comandos Internos do Shell Bourne], Página 44). Uma pesquisa completa dos diretórios em `$PATH` é implementada apenas se o comando não for encontrado na tabela hash. Se a busca não tiver sucesso, então o shell busca uma função de shell definida chamada `command_not_found_handle`. Se tal função existir, ela é invocada com o comando original e os argumentos do comando original como os argumentos dela própria, e o código de saída da função se torna o código de saída do shell. Se tal função não estiver definida, então o shell imprime uma mensagem de erro e retorna um código de saída igual a 127.
4. Se a busca tiver sucesso, ou se o nome do comando contém uma ou mais barras, então o shell executa o aplicativo nomeado em um ambiente de execução separado. O argumento 0 é configurado para o nome dado, e os argumentos restantes para o comando são configurados para os argumentos fornecidos, se existir algum.
5. Se tal execução falhar pelo fato de o arquivo não estar em formato executável, e o arquivo não for um diretório, então é presumido ser um *shell script* e o shell o executa conforme descrito em Seção 3.8 [Scripts de Shell], Página 42.
6. Se o comando não foi iniciado assincronamente, então o shell aguarda que o comando termine sua execução e coleta o código de saída dele.

3.7.3 Ambiente de Execução de Comando

O shell tem um *execution environment*, o qual consiste do seguinte:

- arquivos abertos herdados pelo shell quando da invocação, conforme modificado pelas redireções fornecidas ao comando interno `exec`
- o diretório atual de trabalho conforme configurado por `cd`, `pushd`, ou `popd`, ou herdado pelo shell quando da invocação
- a máscara de modo de criação de arquivo conforme configurado por `umask` ou herdado do pai do shell
- coletores atuais configurados por `trap`
- parâmetros de shell que são configurados por atribuição de variáveis ou com `set` ou herdados do pai do shell no ambiente
- funções de shell definidas durante a execução ou herdadas do pai do shell no ambiente

- opções habilitadas quando da invocação (ou por padrão ou com argumentos de linha de comando) ou por `set`
- opções habilitadas por `shopt` (veja-se Seção 4.3.2 [O Comando Interno Shopt], Página 68)
- apelidos (aliases) de shell definidos com `alias` (veja-se Seção 6.6 [Apelidos], Página 96)
- vários IDs de processos, incluindo aqueles das tarefas de segundo plano (veja-se Seção 3.2.3 [Listas], Página 9), o valor de `$$`, e o valor de `$PPID`

Quando outro comando simples que não um comando interno ao shell ou uma função de shell está para ser executada, ele (o comando simples) é invocado em um ambiente de execução separado que consiste do seguinte. A menos que anotado de outra forma, os valores são herdados do shell.

- os arquivos abertos do shell, mais quaisquer modificações e adições especificadas por redireções ao comando
- o diretório de trabalho atual
- a máscara de modo de criação de arquivo
- funções e variáveis de shell marcadas para exportar, juntamente com variáveis exportadas pelo comando, passadas no ambiente (veja-se Seção 3.7.4 [Ambiente], Página 40)
- coletores capturados pelo shell são reconfigurados para os valores herdados do pai do shell, e coletores ignorados pelo shell são ignorados

Um comando invocado nesse ambiente separado não pode afetar o ambiente de execução do shell.

Substituição de comando, comandos agrupados entre parênteses, e comandos assíncronos são invocados em um ambiente de sub-shell que é uma cópia do ambiente de shell, exceto que os coletores capturados pelo shell são reconfigurados para os valores que o shell herdou do pai dele quando da invocação. Comandos internos que são invocados como parte de um canal de comunicação (pipeline) são executados também em um ambiente de sub-shell. Mudanças feitas no ambiente de sub-shell não podem afetar o ambiente de execução do shell.

Sub-shell gerados para executar substituições de comandos herdam o valor da opção `-e` do shell pai. Quando não em modo POSIX, Bash limpa a opção `-e` em tais sub-shells.

Se um comando for seguido por um sinal '`&`' e o controle de tarefa não estiver ativo, o arquivo descritor de entrada padronizada padrão para o comando é o arquivo vazio `/dev/null`. Do contrário, o comando invocado herda os descritores de arquivo do shell invocante conforme modificado pelas redireções.

3.7.4 Ambiente

Quando um aplicativo é invocado, um vetor de sequência de caracteres é dado a tal aplicativo, vetor esse chamado de *environment*. Esse vetor é uma lista de pares nome-valor, no formato `name=value`.

O Bash provê várias maneiras de manipular o ambiente. Quando da invocação, o shell escaneia o próprio ambiente dele e cria um parâmetro para cada nome encontrado, automaticamente marcando tais parâmetros para *export* para processos filhos. Os comandos executados herdam o ambiente. Os comandos `export` e '`declare -x`' permitem que parâmetros

e funções sejam adicionados a e deletados do ambiente. Se o valor de um parâmetro no ambiente for modificado, o valor novo se torna parte do ambiente, substituindo o antigo. O ambiente herdado por qualquer comando executado consiste do ambiente inicial do shell, cujos valores podem ser modificados dentro do shell, exceto quaisquer pares removidos pelos comandos `unset` e '`export -n`', mais quaisquer adições via comandos `export` e '`declare -x`'.

O ambiente para qualquer comando simples ou função pode ser aumentado temporariamente prefixando tal ambiente com atribuições de parâmetros, conforme descrito em Seção 3.4 [Parâmetros de Shell], Página 20. Tais declarações de atribuição afetam apenas o ambiente visto pelo comando.

Se a opção `-k` for configurada (veja-se Seção 4.3.1 [O Comando Interno Set], Página 63), então todas as atribuições de parâmetro são colocadas no ambiente para um comando, não apenas aquelas que precedem o nome do comando.

Quando Bash invoca um comando externo, a variável '`$_`' é configurada para o nome de caminho completo do comando e passada para aquele comando dentro do ambiente dele.

3.7.5 Situação de Saída

O código de saída de um comando executado é o valor retornado pela chamada de sistema *waitpid* ou função equivalente. Os códigos de saída estão entre o e 255, embora, conforme explanado abaixo, o shell pode utilizar especialmente valores acima de 125. Os códigos de saída originados de comandos internos ao shell e de comandos compostos também são limitados ao intervalo mencionado. Sob certas circunstâncias, o shell utilizará valores especiais para indicar modos de falha específicos.

Para os propósitos do shell, um comando o qual sai com um código de saída igual a zero teve sucesso. Um código de saída diferente de zero indica uma falha. Esse aparentemente esquema de intuitivo de contagem é utilizado de forma que existe uma maneira bem definida para indicar sucesso e uma variedade de maneiras de se indicar vários modos de falha. Quando um comando termina em um sinal fatal cujo número é N, Bash utiliza o valor 128+N como o código de saída.

Se um comando não for encontrado, então o processo filho criado para executá-lo retorna o código de saída 127. Se um comando é encontrado, porém não é um executável, então 'código de saída é o 126.

Se um comando falha por causa de um erro durante a operação de expansão ou a de redireção, então e código de saída é um maior que zero.

O código de saída é utilizado pelos comandos condicionais de Bash (veja-se Seção 3.2.4.2 [Construtores Condicionais], Página 11) e por alguns dos construtores de lista (veja-se Seção 3.2.3 [Listas], Página 9).

Todos os comandos internos ao Bash retornam um código de saída de zero se forem bem sucedidos e um código de saída diferente de zero na falha, de forma que tais códigos de saída podem ser utilizados pelos construtores de lista e condicional. Todos os comandos internos retornam um código de saída de 2 para indicar utilização incorreta.

3.7.6 Sinais

Quando Bash está em modo interativo, na ausência de qualquer coletor, ele ignora `SIGTERM` (de forma que '`kill 0`' não mata um shell interativo), e `SIGINT` é capturado e manipulado

(de forma que o comando interno `wait` pode ser interrompido). Quando Bash recebe um `SIGINT`, ele se liberta de quaisquer loops em execução. Em todos os casos, Bash ignora `SIGQUIT`. Se o controle de tarefa estiver em efeito (veja-se Capítulo 7 [Controle de Tarefa], Página 106), Bash ignora `SIGTTIN`, `SIGTTOU`, e `SIGTSTP`.

Os comandos não internos iniciados por Bash tem manipuladores de sinal configurados para os valores herdados pelo shell originados dos pais dele. Quando o controle de tarefa não está em efeito, os comandos assíncronos ignoram `SIGINT` e `SIGQUIT` em adição àqueles manipuladores herdados. Os comandos executados como um resultado da substituição de comando ignoram os sinais de controle de tarefa gerados pelo teclado: `SIGTTIN`, `SIGTTOU`, e `SIGTSTP`.

O shell sai por padrão assim que receber um `SIGHUP`. Antes de sair, um shell interativo reenvia o `SIGHUP` para todas as tarefas, em execução ou paradas. Para as tarefas paradas é enviado `SIGCONT` para se assegurar que elas receberam o `SIGHUP`. Para prevenir que o shell envie o sinal `SIGHUP` para uma tarefa em particular, tal tarefa deveria ser removida da tabela com o comando interno `disown` (veja-se Seção 7.2 [Comandos Internos do Controle de Tarefa], Página 107) ou assinaladas para não receber `SIGHUP` utilizando-se `disown -h`.

Se a opção de shell `huponexit` tiver sido configurada com `shopt` (veja-se Seção 4.3.2 [O Comando Interno Shopt], Página 68), então Bash envia um `SIGHUP` a todas as tarefas quando um shell de login interativo sai.

Se Bash estiver aguardando que um comando complete sua execução e recebe um sinal para o qual um coletor tenha sido configurado, o coletor não será executado até que o comando complete sua execução. Quando Bash está aguardando um comando assíncrono, via comando interno `wait`, a recepção de um sinal para o qual um coletor tenha sido configurado fará com que o comando interno `wait` retorne imediatamente com um código de saída maior que 128, imediatamente após o que o coletor é executado.

3.8 Scripts de Shell

Um script de shell é um arquivo de texto contendo comandos de shell. Quando um tal arquivo é utilizado como o primeiro argumento não opção quando da invocação de Bash, e nem a opção `-c` nem a `-s` for fornecida (veja-se Seção 6.1 [Invocando o Bash], Página 87), Bash lê e executa comandos a partir do arquivo, e sai. Tal modo de operação cria um shell não interativo. O shell primeiro procura pelo arquivo no diretório de trabalho atual, e olha nos diretórios em `$PATH` se não for encontrado lá.

Quando Bash executa um script de shell, ele configura o parâmetro especial `0` para o nome do arquivo, ao invés do nome do shell, e os parâmetros posicionais são configurados para o restante dos argumentos, se alguns são dados. Se nenhum argumento adicional forem fornecidos, então os parâmetros posicionais são desconfigurados.

Um script de shell pode ser feito executável utilizando-se o comando `chmod` para se ligar o bit executável. Quando Bash encontra um tal arquivo enquanto procura por um comando no `$PATH`, ele gera um sub-shell para executá-lo. Em outras palavras, executar

```
nome_arquivo arguments
```

é equivalente a executar

```
bash nome_arquivo arguments
```

Se `nome_arquivo` for um script de shell executável. Tal sub-shell reinicializa a si próprio, de maneira que o efeito é como se um shell novo tivesse sido invocado para interpretar o script, com a exceção de que as localizações dos comandos relembradas pelo pai (veja-se a descrição de `hash` em Seção 4.1 [Comandos Internos do Shell Bourne], Página 44) são mantidas pelo filho.

A maioria das versões de Unix faz disso uma parte do mecanismo de execução de comando do sistema operacional. Se a primeira linha de um script inicia com os dois caracteres '`#!`', o restante daquela linha especifica um interpretador para o aplicativo. Assim, você pode especificar para o Bash, `awk`, Perl, ou algum outro interpretador e escrever o resto do arquivo de script naquela linguagem.

Os argumentos para o interpretador consistem de um argumento opcional único seguido do nome do interpretador na primeira linha do arquivo de script, seguido pelo nome do arquivo de script, seguido pelo resto dos argumentos. Bash realizará tal ação em sistemas operacionais que não manipulam isso por eles próprios. Perceba que algumas versões antigas de Unix limitam o nome do interpretador e argumento a um máximo de 32 caracteres.

Os scripts de Bash frequentemente começam com `#! /bin/bash` (presumindo que Bash tenha sido instalado em `/bin`), dado que isso assegura que Bash será utilizado para interpretar o script, mesmo que o script seja executado sob um outro shell.

4 Comandos Internos ao Shell

Comandos internos são contidos no próprio shell. Quando o nome de um comando interno é utilizado como a primeira palavra de um comando simples (veja-se Seção 3.2.1 [Comandos Simples], Página 8), o shell executa o comando diretamente, sem invocar outro programa. Comandos internos são necessários para implementar funcionalidade impossível ou inconveniente de se obter com utilitários separados.

Esta seção descreve brevemente os comandos internos os quais Bash herda do Shell Bourne, bem como comandos internos os quais são únicos ao ou foram estendidos no Bash.

Vários comandos internos são descritos em outros capítulos: comandos internos os quais proveem a interface do Bash para as facilidades de controle de tarefas (veja-se Seção 7.2 [Comandos Internos do Controle de Tarefa], Página 107), a pilha de diretório (veja-se Seção 6.8.1 [Comandos Internos da Pilha de Diretório], Página 99), o histórico de comando (veja-se Seção 9.2 [Comandos Internos ao Histórico de Bash], Página 146), e as facilidades de complementação programável (veja-se Seção 8.7 [Comandos Internos à Complementação Programável], Página 137).

Muitos dos comandos internos foram estendidos por POSIX ou Bash.

A menos que informado do contrário, cada comando interno documentado como que aceita opções precedidas por '`-`' também aceita '`--`' para significar o fim das opções. Os comandos internos `:`, `true`, `false`, e `test` não aceitam opções e não tratam '`--`' especialmente. Os comandos internos `exit`, `logout`, `break`, `continue`, `let`, e `shift` aceitam e processam argumentos começando com '`-`' sem exigir '`--`'. Outros comandos internos que aceitam argumentos, mas não são especificados como aceitantes de opções, interpretam argumentos começando com '`-`' como opções inválidas e exigem '`--`' para prevenir tal interpretação.

4.1 Comandos Internos do Shell Bourne

Os comandos internos de shell seguintes são herdados do Shell Bourne. Estes comandos são implementados conforme especificado pelo padrão POSIX.

`:` (dois pontos)

```
: [argumentos]
```

Não faz nada além de expandir *argumentos* e implementar redireções. O código de retorno é zero.

`.` (um ponto)

```
. nome_arquivo [argumentos]
```

Lê e executa comandos a partir do argumento *nome_arquivo* no contexto do shell atual. Se *nome_arquivo* não contiver uma barra, então a variável `PATH` é utilizada para encontrar *nome_arquivo*. Quando Bash não está em modo POSIX, o diretório atual é pesquisado se *nome_arquivo* não for encontrado em `$PATH`. Se quaisquer *argumentos* forem fornecidos, então eles se tornam os parâmetros posicionais quando *nome_arquivo* for executado. Do contrário, os parâmetros posicionais não são modificados. O código de retorno é o código de saída do último comando executado, ou zero se nenhum dos comandos for executado. Se *nome_arquivo* não for encontrado, ou não puder ser lido, o código de retorno é diferente de zero. Esse comando interno é equivalente a `source`.

`break`

```
break [n]
```

Sai de um loop `for`, `while`, `until`, ou `select`. Se *n* for fornecido, então o *n*ésimo loop envolvente é terminado. *n* deve necessariamente ser maior que ou igual a 1. O código de retorno é zero a menos que *n* não seja maior que ou igual a 1.

`cd`

```
cd [-L|[-P [-e]] [-@] [diretório]
```

Muda o diretório de trabalho atual para *diretório*. Se *diretório* não for fornecido, então o valor da variável de shell `HOME` é utilizado. Quaisquer argumentos adicionais seguintes a *diretório* são ignorados. Se a variável de shell `CDPATH` existir, ela é utilizada como um caminho de busca: cada nome de diretório em `CDPATH` é comparado com *diretório*, com nomes de diretório alternativos separados por um sinal de dois pontos (‘`:`’). Se *diretório* começar com uma barra, então `CDPATH` não é utilizada.

A opção `-P` significa não seguir links simbólicos: links simbólicos são resolvidos enquanto `cd` está atravessando *diretório* e antes de processar uma instância de ‘`..`’ em *diretório*.

Por padrão, ou quando a opção `-L` for fornecida, links simbólicos em *diretório* são resolvidos após `cd` processar uma ocorrência de ‘`..`’ em *diretório*.

Se ‘`..`’ aparecer em *diretório*, então essa ocorrência é processada removendo-se o componente de nome de caminho imediatamente precedente, de volta até uma barra ou o início de *diretório*.

Se a opção `-e` for fornecida com `-P` e o diretório de trabalho atual não puder ser determinado com sucesso após uma mudança de diretório exitosa, então `cd` retornará um código de saída de insucesso.

Em sistemas que suportam isto, a opção `-@` apresenta os atributos estendidos associados com um arquivo como um diretório.

Se *diretório* for ‘`-`’, então ele é convertido para `$OLDPWD` antes que a mudança de diretório seja tentada.

Se um nome de diretório não vazio de `CDPATH` for utilizado, ou se ‘`-`’ for o primeiro argumento, e a mudança de diretório tiver sucesso, então e nome de caminho absoluto do novo diretório de trabalho é escrito na saída padrão.

O código de retorno é zero se o diretório for mudado com sucesso, e diferente de zero do contrário.

`continue`

```
continue [n]
```

Retoma a próxima interação de um loop envolvente `for`, `while`, `until`, ou `select`. Se *n* for fornecido, então a execução do *n*ésimo loop envolvente é retomada. *n* deve necessariamente ser maior que ou igual a 1. O código de retorno é zero a menos que *n* não seja maior que ou igual a 1.

`eval`

`eval [argumentos]`

Os argumentos são concatenados juntos em um único comando, o qual é então lido e executado, e o código de saída dele é retornado como o código de saída de `eval`. Se não existirem argumentos ou apenas argumentos vazios, o código de retorno é zero.

`exec`

`exec [-cl] [-a nome] [comando [argumentos]]`

If *comando* for fornecido, então ele substitui o shell sem criar um processo novo. Se a opção `-l` for fornecida, então o shell coloca um traço no início do argumento zero passado para *comando*. Isso é o que o aplicativo `login` faz. A opção `-c` faz com que *comando* seja executado com um ambiente vazio. Se a opção `-a` for fornecida, então o shell passa *nome* como o argumento zero para *comando*. Se *comando* não puder ser executado por alguma razão, então um shell não-interativo sai, a menos que a opção de shell `execfail` esteja habilitada. Nesse caso, é retornado uma falha. Um shell interativo retorna falha se o arquivo não puder ser executado. Se *comando* não for especificado, então as redireções podem ser utilizadas para afetar o ambiente de shell atual. Se não existirem erros de redireção, então o código de retorno é zero; do contrário, o código de retorno é diferente de zero.

`exit`

`exit [n]`

Sai do shell, retornando um código de saída *n* para o pai do shell. Se *n* for omitida, o código de saída é aquele do último comando executado. Qualquer coletor em `EXIT` é executado antes que o shell termine.

`export`

`export [-fn] [-p] [nome[=valor]]`

Marca cada *nome* a ser passado para processos filho no ambiente. Se a opção `-f` for fornecida, então os *nome*s se referem a funções de shell; do contrário, os nomes se referem a variáveis de shell. A opção `-n` significa que não mais marcar cada *nome* para exportar.

Se nenhum *nome*s são fornecidos, ou se a opção `-p` é dada, então uma lista de nomes de todas as variáveis exportadas é mostrada. A opção `-p` mostra saída em uma forma que pode ser reutilizada como entrada. Se a variável nome for seguida por =*valor*, então o valor da variável é configurado para *valor*.

O código de retorno é zero a menos que uma opção inválida seja fornecida, um dos nomes não seja um nome de variável válido do shell, ou a opção `-f` seja fornecida com um nome que não seja uma função de shell.

`getopts`

`getopts optstring nome [args]`

`getopts` é utilizado por scripts de shell para analisar parâmetros posicionais. *optstring* contém os caracteres opcionais a serem reconhecidos; se um carácter for seguido por um sinal de dois pontos, então é esperado que a opção tenha um argumento, o qual deveria estar separado da opção por um espaço em branco.

O sinal de dois pontos (‘`:`’) e o de interrogação (‘`?`’) podem não serem utilizados como caracteres opcionais. A cada vez que é invocado, `getopts` coloca a próxima opção na variável de shell *nome*, inicializando *nome* caso ela não exista, e o índice do próximo argumento a ser processado na variável `OPTIND`. `OPTIND` é inicializada em 1 a cada vez que o shell ou um script de shell for invocado. Quando uma opção exige um argumento, `getopts` coloca aquele argumento na variável `OPTARG`. O shell não reconfigura `OPTIND` automaticamente; ela deve necessariamente ser reconfigurada manualmente entre múltiplas chamadas a `getopts` dentro da mesma invocação de shell se um conjunto novo de parâmetros está para ser utilizado.

Quando o final das opções é encontrado, `getopts` sai com um código de retorno maior que zero. `OPTIND` é configurada para o índice do primeiro argumento não-opção, e *nome* é configurada para ‘`?`’.

`getopts` normalmente analisa os parâmetros posicionais, porém se mais argumentos são dados em *args*, `getopts` analisa esses argumentos em vez disso.

`getopts` pode relatar erros de duas maneiras. Se o primeiro carácter de *optstring* for um sinal de dois pontos, então o relatador de erro *silent* é utilizado. Em operação normal, mensagens de diagnóstico são impressas quando opções inválidas ou a falta de argumentos de opção são encontrados. Se a variável `OPTERR` for configurada para zero (0), então nenhuma mensagem de erro será mostrada, mesmo que o primeiro carácter de `optstring` não seja o sinal de dois pontos.

Se uma opção inválida é vista, `getopts` coloca ‘`?`’ dentro de *nome* e, se não silêncio, imprime uma mensagem de erro e desconfigura `OPTARG`. Se `getopts` for silente, então o carácter de opção encontrado é colocado em `OPTARG` e nenhuma mensagem de diagnóstico é impressa.

Se um argumento exigido não for encontrado, e `getopts` não for silente, um ponto de interrogação (‘`?`’) é colocado em *nome*, `OPTARG` é desconfigurada, e uma mensagem de diagnóstico é impressa. Se `getopts` for silente, então um sinal de dois pontos (‘`:`’) é colocado em *nome* e `OPTARG` é configurada para o caracteres de opção encontrado.

`hash`

```
hash [-r] [-p nome_arquivo] [-dt] [nome]
```

A cada vez que `hash` é invocado, ele se lembra dos nomes completos de caminhos dos comandos especificados como argumentos *nome*, de maneira que não se precisa procurar nos caminhos em subsequentes invocações. Os comandos são encontrados pesquisando-se em todos os diretórios listados em `$PATH`. Quaisquer nome de caminho previamente relembrados são descartados. A opção `-p` inibe a procura de caminho, e *nome_arquivo* é utilizado como a localização de *nome*. A opção `-r` faz com que o shell se esqueça de todas as localizações relembradas. A opção `-d` faz com que o shell se esqueça da localização relembrada de cada *nome*. Se a opção `-t` for fornecida, então o nome completo de caminho para o qual cada *nome* corresponda é impresso. Se múltiplos argumentos *nome* forem fornecidos com `-t`, o *nome* é impresso antes do nome completo do caminho esmiuçado. A opção `-l` faz com que a saída seja exibida em um formato que

pode ser reutilizado como entrada. Se nenhum argumento é dado, ou se apenas `-l` for fornecida, então informação sobre comandos relembrados é impressa. O código de retorno é zero, a menos que um *nome* não seja encontrado ou uma opção inválida seja fornecida.

`pwd`

```
pwd [-LP]
```

Imprime o nome de caminho absoluto do diretório de trabalho atual. Se a opção `-P` for fornecida, então o nome de caminho impresso pode conter links simbólicos. O código de retorno é zero, a menos que um erro seja encontrado quando da determinação do nome do diretório atual ou uma opção inválida seja fornecida.

`readonly`

```
readonly [-aAf] [-p] [nome[=valor]] ...
```

Marca cada *nome* como apenas leitura. Os valores desses nomes não podem ser modificados por atribuição subsequente. Se a opção `-f` for fornecida, então cada *nome* se refere a uma função de shell. A opção `-a` significa que cada *nome* se refere a uma variável de vetor indexado; a opção `-A` significa que cada *nome* se refere a uma variável de vetor associativo. Se ambas as opções são fornecidas, então `-A` tem precedência. Se nenhum argumento *nome* for dado, ou se a opção `-p` for fornecida, então uma lista de todos os nomes apenas leitura é impressa. As outras opções podem ser utilizadas para restringir a saída a um subconjunto do conjunto de nomes apenas leitura. A opção `-p` faz com que a saída seja exibida em um formato que pode ser reutilizada como entrada. Se um nome de variável for seguido por =*valor*, então o valor da variável é configurada para *valor*. O código de retorno é zero, a menos que uma opção inválida seja fornecida, um dos argumentos *nome* não seja uma variável de shell válida ou um nome de função, ou a opção `-f` seja fornecida com um nome que não seja uma função de shell.

`return`

```
return [n]
```

Faz com que uma função de shell pare a sua execução e retorne o valor *n* a quem a chamou. Se *n* não for fornecida, o código de retorno é o código de saída do último comando executado na função. `return` também pode ser utilizado para finalizar a execução de um script sendo executado com o comando interno `.` (`source`), retornando ou *n* ou o código de saída do último comando executado dentro do script como o código de saída do script. Se *n* for fornecida, então o valor do código de retorno são os últimos 8 bits significantes da variável. Qualquer comando associado com o coletor `RETURN` é executado antes da retomada da execução após a função ou script. O código de retorno é diferente de zero se a `return` for fornecido um argumento não numérico ou for utilizado do lado de fora da função e não durante a execução do script por `.` ou `source`.

`shift`

```
shift [n]
```

Desloca os parâmetros posicionais *n* para a esquerda. Os parâmetros posicionais a partir de *n*+1 ... `$#` são renomeados para `$1` ... `$#`-*n*. Os parâmetros representados pelos números `$#` até `$#`-*n*+1 são desconfigurados. *n* deve necessariamente ser um número não-negativo menor que ou igual a `$#`. Se *n* for zero ou maior que `$#`, então os parâmetros posicionais não são modificados. Se *n* não for fornecido, então presume-se que seja igual a 1. O código de retorno é zero, a menos que *n* seja maior que `$#` ou menor que zero, e diferente de zero do contrário.

`test`
`[`

`test` *expr*

Avalia uma expressão condicional *expr* e retorna um código 0 (verdadeiro) ou 1 (falso). Cada operador e operando deve necessariamente ser um argumento separado. Expressões são compostas dos primários descritos abaixo em Seção 6.4 [Expressões Condicionais de Bash], Página 93. `test` não aceita quaisquer opções, nem aceita e ignora um argumento de `--` como significando o final das opções.

Quando a forma `[` é utilizada, o último argumento para o comando deve necessariamente ser um `]`.

Expressões podem ser combinadas utilizando-se os seguintes operadores, listados em ordem decrescente de precedência. A avaliação depende do número de argumentos; veja-se abaixo. A precedência de operador é utilizada quando existem cinco ou mais argumentos.

`!` *expr* Verdadeiro se *expr* for falso.

`(` *expr* `)` Retorna o valor de *expr*. Isso pode ser utilizado para substituir a precedência normal de operadores.

expr1 `-a` *expr2*
Verdadeiro se ambas *expr1* e *expr2* forem verdadeiras.

expr1 `-o` *expr2*
Verdadeiro se ou *expr1* ou *expr2* for verdadeira.

Os comandos internos `test` e `[` avaliam expressões condicionais utilizando um conjunto de regras baseadas no número de argumentos.

0 argumentos
A expressão é falsa.

1 argumento
A expressão é verdadeira se e somente se o argumento não nulo.

2 argumentos
Se o primeiro argumento for '`!`', então a expressão é verdadeira se e somente se o segundo argumento for nulo. Se o primeiro argumento for um dos operadores condicionais unários (veja-se Seção 6.4 [Expressões Condicionais de Bash], Página 93), então a expressão

é verdadeira se o teste unário for verdadeiro. Se o primeiro argumento não for um operador unário válido, então a expressão é falsa.

3 argumentos
: As seguintes condições são aplicadas na ordem listada. Se o segundo argumento for um dos operadores condicionais binários (veja-se Seção 6.4 [Expressões Condicionais de Bash], Página 93), então o resultado da expressão é o resultado do teste binário utilizando-se o primeiro e o terceiro argumentos como operandos. Os operadores '`-a`' e '`-o`' são considerados operadores binários quando existem três argumentos. Se o primeiro argumento for '`!`', então o valor é a negação do teste de dois argumentos utilizando-se o segundo e o terceiro argumentos. Se o primeiro argumento for exatamente '`(`' e o terceiro argumento for exatamente '`)`', então o resultado é o teste de um argumento do segundo argumento. Do contrário, a expressão é falsa.

4 argumentos
: Se o primeiro argumento for '`!`', então o resultado é a negação da expressão de três argumentos, composta dos restantes argumentos. Do contrário, a expressão é analisada e avaliada de acordo com a precedência, utilizando-se as regras listadas acima.

5 ou mais argumentos
: A expressão é analisada e avaliada de acordo com a precedência, utilizando-se as regras listadas acima.

Quando utilizado com `test` ou com '`[`', os operadores '`<`' e '`>`' ordenam lexicograficamente, utilizando ordenamento ASCII.

`times`

```
times
```

Imprime os tempos de sistema e de usuário, utilizados pelo shell e seus filhos. O código de retorno é zero.

`trap`

```
trap [-lp] [arg] [sigspec ...]
```

Os comandos em *arg* são para ser executados quando o shell recebe sinal *sigspec*. Se *arg* estiver ausente (e existir um único *sigspec*) ou for igual a '`-`', então cada disposição de sinal especificada é reconfigurada para o valor que ele tinha quando o shell foi iniciado. Se *arg* for a sequência de caracteres "null", então o sinal especificado por cada *sigspec* é ignorado pelo shell e comandos que ele (o shell) invoca. Se *arg* não estiver presente e a opção `-p` tiver sido fornecida, então o shell exibe os comandos de coletor associados com cada *sigspec*. Se nenhum argumento for fornecido, ou apenas a opção `-p` for dada, então `trap` imprime a lista de comandos associados com cada número de sinal em uma forma que pode ser reutilizada como entrada de shell. A opção `-l` faz com que o shell imprima uma lista de nomes de sinal e os números correspondentes

deles. Cada *sigspec* é ou um nome de sinal ou um número de sinal. Nos nomes de sinal tanto faz maiúsculas como minúsculas, e o prefixo `SIG` é opcional.

Se um *sigspec* for `0` ou `EXIT`, então *arg* é executado quando o shell sair. Se um *sigspec* for `DEBUG`, então o comando *arg* é executado antes de cada comando simples, comando `for`, comando `case`, comando `select`, cada comando `for` aritmético, e antes que o primeiro comando execute em uma função de shell. Consulte-se a descrição da opção `extdebug` para o comando interno `shopt` (veja-se Seção 4.3.2 [O Comando Interno Shopt], Página 68) para detalhes de seu efeito sobre o coletor `DEBUG`. Se um *sigspec* for `RETURN`, então o comando *arg* é executado a cada vez que uma função de shell ou um script executado com os comandos internos `.` ou `source` finalizar a execução. Se um *sigspec* for `ERR`, então o comando *arg* é executado sempre que um canal de comunicação (o qual pode consistir de um único comando simples), uma lista, ou um comando composto retornar um código de saída diferente de zero, objeto das seguintes condições. O coletor `ERR` não é executado se o comando falho for parte da lista de comando imediatamente seguinte a uma palavra-chave `until` ou `while`, parte do teste seguinte às palavras reservadas `if` ou `elif`, parte de um comando executado em uma lista `&&` ou `||`, exceto o comando que segue o `&&` ou `||` final, qualquer comando em um canal de comunicação, mas o último, ou se o código de retorno do comando estiver sendo invertido utilizando `!`. Essas são as mesmas condições obedecidas pela opção `errexit` (`-e`).

Sinais são ignorados após as entradas ao shell não puderem ser coletadas ou reconfiguradas. Sinais coletados que não estão sendo ignorados são reconfigurados para os valores originais deles em um sub-shell ou ambiente de sub-shell quando um for criado.

O código de retorno é zero, a menos que um *sigspec* não especifique um sinal válido.

`umask`

```
umask [-p] [-S] [modo]
```

Configura a máscara de criação de arquivo do processo de shell para *modo*. Se *modo* iniciar com um dígito, então ele é interpretado como sendo um número octal; se não, é interpretado como sendo uma máscara de modo simbólico, similar àquela aceita pelo comando `chmod`. Se *modo* for omitido, então o valor atual da máscara é impresso. Se a opção `-S` for fornecida sem um argumento *modo*, então a máscara é impressa em um formato simbólico. Se a opção `-p` for fornecida, e *modo* for omitido, então a saída fica em uma forma que pode ser reutilizada como entrada. O código de retorno é zero se o modo for mudado com sucesso ou se nenhum argumento *modo* for fornecido, e diferente de zero do contrário.

Perceba-se que quando o modo for interpretado como sendo um número octal, cada número da máscara é subtraído de `7`. Assim, um umask de `022` resulta em permissões de `755`.

`unset`

```
unset [-fnv] [nome]
```

Remove cada variável ou função *nome*. Se a opção `-v` for dada, cada *nome* se referirá a uma variável de shell e essa variável é removida. Se a opção `-f` for dada, então os *nome*s se referirão a funções de shell, e a definição de função é removida. Se a opção `-n` for fornecida, e *nome* for uma variável com o atributo *nameref*, então *nome* será desconfigurada ao invés da variável que ele referencia. `-n` não tem efeito se a opção `-f` for fornecida. Se nenhuma opção for fornecida, então cada *nome* se referirá a uma variável; se não existir variáveis por aquele nome, então qualquer função com aquele nome é desconfigurada. Variáveis apenas leitura e funções não podem ser desconfiguradas. O código de retorno é zero, a menos que um argumento *nome* seja apenas leitura.

4.2 Comandos Internos ao Bash

Esta seção descreve comandos internos ao shell os quais são únicos para ou foram ampliados no Bash. Alguns desses comandos estão especificados no padrão POSIX.

`alias`

```
alias [-p] [nome[=valor] ...]
```

Sem argumentos ou com a opção `-p`, `alias` imprime a lista de apelidos na saída padrão em uma forma que se possa utilizá-la como entrada. Se argumentos forem fornecidos, um apelido é definido para cada *nome* cujo *valor* for dado. Se nenhum *valor* for dado, então o nome e valor do apelido é impresso. Os apelidos são descritos em Seção 6.6 [Apelidos], Página 96.

`bind`

```
bind [-m keymap] [-lpsvPSVX]
bind [-m keymap] [-q function] [-u function] [-r keyseq]
bind [-m keymap] -f filename
bind [-m keymap] -x keyseq:shell-command
bind [-m keymap] keyseq:function-name
bind readline-command
```

Exibir os vínculos Readline de função e tecla (veja-se Capítulo 8 [Edição de Linha de Comando], Página 110), vincula uma sequência de tecla a uma função Readline ou macro, ou configura uma variável Readline. Cada argumento que não seja uma opção é um comando conforme tal comando apareceria em um arquivo de inicialização Readline (veja-se Seção 8.3 [Arquivo Init de Readline], Página 113), porém cada vinculação ou comando deve necessariamente ser passado como um argumento separado; por exemplo, '`"\C-x\C-r":re-read-init-file`'. As opções, se fornecidas, tem os seguintes significados:

`-m` *keymap* Utilize *keymap* como sendo o mapa de teclas a ser afetado pelas vinculações subsequentes. Os nomes de *keymap* aceitáveis são `emacs`, `emacs-standard`, `emacs-meta`, `emacs-ctlx`, `vi`, `vi-move`, `vi-command`, e `vi-insert`.

`vi` é equivalente a `vi-command`; `emacs` é equivalente a `emacs-standard`.

`-l` Lista os nomes de todas as funções Readline.

`-p` Exibe nomes de função Readline e vinculações em uma tal maneira que se pode utilizá-los como entrada ou em um arquivo de inicialização Readline.

`-P` Lista nomes de função Readline e vinculações atuais.

`-v` Exibe nomes de variáveis Readline e valores em uma tal maneira que se pode utilizá-los como entrada ou em um arquivo de inicialização Readline.

`-V` Lista nomes de variáveis Readline e valores atuais.

`-s` Exibe sequências de tecla Readline vinculadas à macros e as sequências de caracteres que tais sequências de tecla entregam como saída em uma tal maneira que tais sequências de caracteres possam ser utilizadas como entrada ou em um arquivo de inicialização Readline.

`-S` Exibe sequências de tecla Readline vinculadas à macros e as sequências de caracteres que tais sequências de tecla entregam como saída.

`-f` *filename*
Lê vínculos de tecla a partir de *filename*.

`-q` *function*
Consulta sobre quais teclas invocam a *function* nominada.

`-u` *function*
Desvincula todas as teclas vinculadas à *function* nominada.

`-r` *keyseq* Remove qualquer vinculação atual para a sequência de tecla *keyseq*.

`-x` *keyseq:shell-command*
Faz com que *shell-command* seja executado sempre que a sequência de tecla *keyseq* seja fornecida. Quando o comando *shell-command* é executado, o shell configura a variável `READLINE_LINE` para que essa contenha o conteúdo do buffer de linha do Readline e a variável `READLINE_POINT` para apontar para a localização atual do ponto de inserção. Se o comando executado modifica o valor de `READLINE_LINE` ou o de `READLINE_POINT`, então os novos valores dessas variáveis são refletidos no estado de edição.

`-X` Lista todas as sequências de tecla vinculadas à comandos de shell e os comandos associados em uma forma que possa ser reutilizado como entrada.

O código de retorno é zero, a menos que uma opção inválida seja fornecida ou um erro ocorra.

`builtin`

`builtin [`*shell-builtin* `[`*args*`]]`

Executa um comando interno ao shell, passando os argumentos *args* para o comando, e retorna o seu código de saída. Isso é útil quando da definição de

uma função de shell com o mesmo nome que um comando interno ao shell, restringindo a funcionalidade do comando interno ao âmbito da função. O código de retorno é diferente de zero se *shell-builtin* não for um comando interno ao shell.

`caller`

```
caller [expr]
```

Retorna o contexto de qualquer chamada de sub-rotina ativa (uma função de shell ou um script executado com os comandos internos `.` ou `source`).

Sem o parâmetro *expr*, `caller` exibe o número da linha e nome do arquivo fonte da chamada de sub rotina atual. Se um número inteiro não negativo for fornecido como *expr*, então `caller` exibe o número da linha, nome da sub-rotina, e arquivo fonte correspondente àquela posição na pilha de chamada de execução atual. Essa informação extra talvez possa ser utilizada, por exemplo, para imprimir um rastreamento da pilha. O quadro atual é o frame 0.

O código de retorno é 0 a menos que o shell não esteja executando uma chamada de sub-rotina ou *expr* não corresponda a uma posição válida na pilha de chamada.

`command`

```
command [-pVv] command [arguments ...]
```

Executa o comando *command* com os argumentos *arguments*, ignorando qualquer função de shell chamada *command*. Apenas comandos internos ao shell ou comandos encontrados pela busca no caminho `PATH` são executados. Se existir uma função de shell chamada `ls`, a execução de '`command ls`' no âmbito da função executará o comando estorno `ls` ao invés de chamar a função recursivamente. A opção `-p` significa utilizar um valor padrão para `PATH` que garante encontrar todos os utilitários padrão. O código de retorno nesse caso é 127 se *command* não puder ser encontrado ou um erro ocorrer, e o código de saída do comando *command* do contrário.

Se ou a opção `-V` ou a opção `-v` for fornecida, então uma descrição de *command* é impressa. A opção `-v` faz com que seja exibida uma palavra única indicando o comando ou nome de arquivo utilizado para invocar *command*; a opção `-V` produz uma descrição mais detalhada. Nesse caso, o código de retorno é zero se o comando *command* for encontrado, e diferente de zero se não for.

`declare`

```
declare [-aAfFgilnrtux] [-p] [nome[=valor] ...]
```

Declara variáveis e dá atributos a elas. Se nenhum *nome* for dado, então exibe os valores das variáveis.

A opção `-p` exibirá os atributos e os valores de cada *nome*. Quando a opção `-p` é utilizada com argumentos *nome*, opções adicionais, outras que não `-f` e `-F`, são ignoradas.

Quando a opção `-p` é fornecida sem argumentos *nome*, `declare` exibirá os atributos e valores de todas as variáveis que tenham os atributos especificados por opções adicionais. Se nenhuma outra opção for fornecida com `-p`, então

`declare` exibirá os atributos e valores de todas as variáveis de shell. A opção `-f` restringirá a exibição a funções de shell.

A opção `-F` inibe a exibição de definições de função; apenas o nome da função e atributos são impressos. Se a opção de shell `extdebug` estiver habilitada utilizando-se o comando interno ao shell `shopt` (veja-se Seção 4.3.2 [O Comando Interno Shopt], Página 68), então o nome do arquivo fonte e número da linha onde a função está definida são exibidos também. `-F` implica `-f`.

A opção `-g` força que variáveis sejam criadas ou modificadas no escopo global, mesmo quando `declare` é executado em uma função de shell. Essa opção é ignorada em todos os outros casos.

As opções seguintes podem ser utilizadas para restringir saída a variáveis com os atributos especificados ou para dar atributos a variáveis:

`-a` Cada *nome* é uma variável vetor indexada (veja-se Seção 6.7 [Vetores], Página 97).

`-A` Cada *nome* é uma variável vetor associativa (veja-se Seção 6.7 [Vetores], Página 97).

`-f` Utilize apenas nomes de função.

`-i` A variável é para ser tratada como sendo um número inteiro; avaliação aritmética (veja-se Seção 6.5 [Aritmética de Shell], Página 95) é implementada quando for atribuído um valor para a variável.

`-l` Quando um valor é atribuído para a variável, todos os caracteres maiúsculos são convertidos em minúsculos. O atributo maiúsculo é desabilitado.

`-n` Dá para cada *nome* o atributo *nameref*, fazendo com que ele seja uma referência de nome para uma outra variável. Essa outra variável é definida pelo valor de *nome*. Todas as referências e atribuições a *nome*, exceto para modificar o próprio atributo `-n`, são implementadas na variável referenciada pelo valor de *nome*. O atributo `-n` não pode ser aplicado a variáveis vetor.

`-r` Torna *nome*s apenas leitura. Não pode então ser atribuídos valores via declarações de atribuição subsequentes ou desconfigurados.

`-t` Dá a cada *nome* o atributo `trace`. As funções rastreadas herdam os coletores `DEBUG` e `RETURN` do shell que as chamam. O atributo de rastreamento não tem significado especial para variáveis.

`-u` Quando um valor é atribuído a uma variável, todos os caracteres minúsculos são convertidos para maiúsculos. O atributo minúsculo é desabilitado.

`-x` Marca cada *nome* para exportação a comandos subsequentes via o ambiente.

Ao se utilizar '`+`' ao invés de '`-`' desliga o atributo, com as exceções de que '`+a`' pode não ser utilizado para destruir uma variável vetor e '`+r`' não removerá

o atributo apenas leitura. Quando utilizado em uma função, `declare` torna cada *nome* local, como com o comando `local`, a menos que a opção `-g` seja utilizada. Se um nome de variável for seguido por =*value*, então o valor da variável é configurado para *valor*.

Quando se utiliza `-a` ou `-A` e a sintaxe de atribuição composta para criar variáveis vetor, atributos adicionais não tem efeito até atribuições subsequentes.

O código de retorno é zero a menos que uma opção inválida seja encontrada; uma tentativa for feita para definir a função utilizando '`-f foo=bar`'; uma tentativa for feita para atribuir um valor a uma variável apenas leitura; uma tentativa for feita de atribuir um valor a uma variável vetor sem utilizar a sintaxe de atribuição composta (veja-se Seção 6.7 [Vetores], Página 97); um dos *nomes* não for um nome de variável de shell válido; uma tentativa for feita de desligar o estado de somente leitura para uma variável apenas leitura; uma tentativa for feita de desligar o estado de vetor para uma variável vetor; ou uma tentativa for feita de exibir uma função não existente com `-f`.

`echo`

```
echo [-neE] [arg ...]
```

Entrega como saída os argumentos *arg*, separados por espaços, terminados com um carácter newline. O código de retorno é 0 a menos que um erro de escrita ocorra. Se `-n` for especificada, então o newline ao final é suprimido. Se a opção `-e` for dada, então é habilitada a interpretação dos seguintes caracteres encapsulados com barra invertida. A opção `-E` desabilita a interpretação destes caracteres de escape, mesmo em sistemas onde eles são interpretados por padrão. A opção de shell `xpg_echo` pode ser utilizada para determinar dinamicamente quando ou não `echo` expande estes caracteres de escape por padrão. `echo` não interpreta `--` como sendo o final das opções.

`echo` interpreta as seguintes sequências de escape:

`\a`	alerta (sino)
`\b`	backspace
`\c`	suprime saída
`\e` `\E`	escape
`\f`	alimentação de formulário
`\n`	nova linha
`\r`	retorno de carro
`\t`	tab horizontal
`\v`	tab vertical
`\\`	barra invertida
`\0`*nnn*	o carácter de oito-bits cujo valor é o valor octal *nnn* (zero a três dígitos octais)

`\x`*HH* o carácter de oito-bits cujo valor é o valor hexadecimal *HH* (um ou dois dígitos hexadecimais)

`\u`*HHHH* o carácter Unicode (ISO/IEC 10646) cujo valor é o valor hexadecimal *HHHH* (um até quatro dígitos hexadecimais)

`\U`*HHHHHHHH*
o carácter Unicode (ISO/IEC 10646) cujo valor é o valor hexadecimal *HHHHHHHH* (um até oito dígitos hexadecimais)

`enable`

```
enable [-a] [-dnps] [-f nome_arquivo] [nome ...]
```

Habilita e desabilita comandos internos do shell. Desabilitar um comando interno permite que um comando de disco o qual tenha o mesmo nome que o comando interno ao shell seja executado sem se especificar um nome de caminho completo, ainda que o shell normalmente procure por comandos internos antes que comandos de disco. Se a opção `-n` for utilizada, então os *nome*s se tornam desabilitados. Do contrário os *nome*s são habilitados. Por exemplo, para se utilizar o binário `test` encontrado via `$PATH` ao invés da versão interna do shell, digite '`enable -n test`'.

Se a opção `-p` for fornecida, ou nenhum argumento *nome* aparecer, uma lista dos comandos internos ao shell é impressa. Sem nenhum outro argumento, a lista consiste de todos os comandos internos ao shell habilitados. A opção `-a` significa listar cada comando interno com uma indicação de se ou não o comando esteja habilitado.

A opção `-f` significa carregar o novo comando interno *nome* a partir do objeto compartilhado *nome_arquivo*, em sistemas que suportem carregamento dinâmico. A opção `-d` deletará um comando interno carregado com `-f`.

Se não existirem opções, então uma lista dos comandos internos ao shell é exibida. A opção `-s` restringe `enable` aos comandos internos especiais do padrão POSIX. Se `-s` for utilizada com `-f`,, então o novo comando interno se torna um comando interno especial (veja-se Seção 4.4 [Comandos Internos Especiais], Página 74).

O código de retorno é zero a menos que um *nome* não seja um comando interno ao shell ou exista um erro ao carregar um novo comando interno a partir de um objeto compartilhado.

`help`

```
help [-dms] [pattern]
```

Exibe informação útil acerca de comandos internos. Se *pattern* for especificada, então `help` fornece ajuda detalhada sobre todos os comandos coincidentes com *pattern*, do contrário uma lista de comandos internos é impressa.

Opções, de fornecidas, tem os seguintes significados:

`-d` Exibe uma descrição curta de cada *pattern*

`-m` Exibe a descrição de cada *pattern* em um formato parecido com página de manual.

`-s` Exibe apenas uma sinopse curta de uso para cada *pattern*

O código de retorno é zero a menos que nenhum comando coincida com *pattern*.

`let`

```
let expressão [expressão ...]
```

O comando interno `let` permite que operações aritméticas sejam efetuadas sobre variáveis de shell. Cada *expressão* é avaliada de acordo com as regras dadas abaixo em Seção 6.5 [Aritmética de Shell], Página 95. Se a última *expressão* avaliada resultar em 0, então `let` retornará 1; do contrário 0 é retornado.

`local`

```
local [opção] nome[=valor] ...
```

Para cada argumento, uma variável local chamada *nome* é criada, e *valor* é atribuído a essa variável. A *opção* pode ser quaisquer das opções aceitas por `declare`. `local` apenas pode ser utilizado dentro de uma função; ele faz com que a variável *nome* tenha um escopo visível restrito àquela função e os filhos dela. O código de retorno é zero a menos que `local` seja utilizada fora da função, um *nome* seja fornecido, ou *nome* seja uma variável apenas leitura.

`logout`

```
logout [n]
```

Sai de um shell de login, retornando um código de *n* para o pai do shell.

`mapfile`

```
mapfile [-n count] [-O origin] [-s count] [-t] [-u fd]
    [-C callback] [-c quantum] [array]
```

Lê linhas a partir da entrada padrão armazenando em uma variável vetor indexada *array*, ou a partir do descritor de arquivo *fd* se a opção `-u` for fornecida. A variável `MAPFILE` é o *array* padrão. Opções, se fornecidas, tem os seguintes significados:

`-n` Copia no máximo *count* linhas. Se *count* for 0, então todas as linhas são copiadas.

`-O` Atribui a *array* iniciando no índice *origin*. O índice padrão é 0.

`-s` Descarta as primeiras *count* linhas lidas.

`-t` Remove um marcador newline do final de cada linha lida.

`-u` Lê linhas a partir do descritor de arquivo *fd* ao invés de o fazer a partir da entrada padrão.

`-C` Avalia *callback* a cada vez que *quantum*P linhas são lidas. A opção `-c` especifica *quantum*.

`-c` Especifica o número de linhas lidas entre cada chamada a *callback*.

Se `-C` for especificada sem `-c`, então a quantidade padrão é 5000. Quando *callback* for avaliada, como argumentos adicionais, é fornecido o índice do próximo elemento do vetor a ser atribuído e a linha a ser atribuída àquele elemento.

callback é avaliada após a linha ser lida, mas antes que o elemento do vetor seja atribuído.

Se `mapfile` não for fornecida com uma origem explícita, então `mapfile` esvaziará *array* antes de atribuir-lhe dados.

`mapfile` retorna um código de sucesso a menos que uma opção inválida ou um argumento opcional seja fornecido; *array* seja inválido e inatribuível; ou *array* não seja um vetor indexado.

`printf`

```
printf [-v var] formato [argumentos]
```

Escreve os *argumentos* formatados na saída padrão sob o controle de *formato*. A opção `-v` faz com que a saída seja atribuída para a variável *var* no lugar de ser impressa na saída padrão.

O *formato* é uma sequência de caracteres a qual contém três tipos de objetos: caracteres planos, os quais são simplesmente copiados para a saída padrão; sequências de caracteres de escape, as quais são convertidas e copiadas para a saída padrão; e especificações de formato, cada uma das quais provoca a impressão dos próximos *argumentos* sucessivos. Em adição aos formatos `printf(1)` padrão, `printf` interpreta as seguintes extensões:

`%b` Faz com que `printf` expanda sequências de escape de barra invertida nos correspondentes *argumentos*, exceto que '`\c`' finaliza a saída; barras invertidas dentro de '`\'`', '`\"`', e '`\?`' não são removidas; e escapes octais iniciando com '`\0`' podem conter até quatro dígitos.

`%q` Faz com que `printf` entregue como saída os correspondentes *argumentos* em um formato que pode ser reutilizado como entrada de shell.

`%(datefmt)T`
Faz com que `printf` entregue como saída a sequência de caracteres resultante da utilização de *datefmt* como uma sequência de caracteres de formatação para `strftime`(3). Os *argumentos* correspondentes são um número inteiro representativos do número de segundos desde a época. Dois valores de argumento especiais podem ser utilizados: -1 representa o horário atual; e -2 representa o horário no qual o shell foi invocado. Se nenhum argumento for especificado, então a conversão se comporta como se -1 tivesse sido dado. Isso é uma exceção ao comportamento usual de `printf`.

Os argumentos para os especificadores de formato que não sejam sequências de caracteres são tratados como constantes da linguagem C, exceto que é permitido no início um sinal mais ou um menos, e se o carácter inicial for uma aspa simples ou uma dupla, então o valor é o valor ASCII do carácter seguinte.

O *formato* é reutilizado conforme necessário para consumir todos os *argumentos*. Se o *formato* necessitar de mais *argumentos* que os que forem fornecidos, então as especificações extra de formato se comportam como se um

valor zero ou uma sequência de caracteres nula, conforme apropriado, tivesse sido fornecida. O código de retorno é zero no sucesso, e diferente de zero na falha.

`read`

```
read [-ers] [-a aname] [-d delim] [-i text] [-n nchars]
    [-N nchars] [-p prompt] [-t timeout] [-u fd] [name ...]
```

Uma linha é lida a partir da entrada padrão, ou a partir do descritor de arquivo *fd* fornecido como um argumento para a opção `-u`, e a primeira palavra é atribuída para o primeiro *name*; a segunda palavra para o segundo *name*, e assim por diante, com as palavras que sobrarem e os separadores intervenientes delas atribuídos para o último *name*. Se existirem palavras menores que os nomes, lidas do fluxo de entrada, então aos nomes remanescentes serão atribuídos valores vazios. Os caracteres no valor da variável `IFS` são utilizados para dividir a linha em palavras utilizando-se as mesmas regras que o shell utiliza para expansão (descrita acima em Seção 3.5.7 [Divisão de Palavra], Página 31). O carácter "barra invertida" '\' pode ser utilizado para remover qualquer significado especial do próximo carácter lido e para continuação de linha. Se nenhum nome for fornecido, então a linha lida é atribuída à variável `REPLY`. O código de retorno é zero, a menos que um marcador de fim de arquivo for encontrado; `read` tiver em intervalo de tempo (caso no qual o código de retorno é maior que 128); um erro de atribuição de variável (tal qual a atribuição a uma variável apenas leitura) ocorrer; ou um descritor de arquivo inválido for fornecido como um argumento para `-u`.

Opções, se fornecidas, tem os seguintes significados:

`-a` *aname* As palavras são atribuídas a índices sequenciais da variável vetor *aname*, iniciando no 0. Todos os elementos são removidos do *aname* antes da atribuição. Outros argumentos *aname* são ignorados.

`-d` *delim* O primeiro carácter de *delim* é utilizado para terminar a linha de entrada, ao invés do marcador newline.

`-e` Readline (veja-se Capítulo 8 [Edição de Linha de Comando], Página 110) é utilizado para se obter a linha. Readline utiliza as configurações de edição atuais (ou a padrão, se a edição de linha não foi previamente ativada).

`-i` *text* Se Readline está sendo utilizada para ler a linha, então *text* é colocado na área de edição antes que a edição propriamente dita comece.

`-n` *nchars* `read` retorna após a leitura de *nchars* caracteres ao invés de aguardar por uma linha completa de entrada, porém respeita um delimitador se menos que *nchars* caracteres forem lidos antes do delimitador.

`-N` *nchars* `read` retorna após a leitura de exatamente *nchars* caracteres ao invés de aguardar pela leitura de uma linha completa de entrada, a menos que um marcador EOF seja encontrado ou `read` tenha um intervalo de tempo. Os caracteres delimitadores encontrados na

entrada não são tratados especialmente e não fazem com que `read` retorne até que *nchars* sejam lidos.

`-p` *prompt* Exibe *prompt*, sem um finalizador newline, antes da tentativa de ler qualquer entrada. O prompt é exibido apenas se a entrada estiver vindo de um terminal.

`-r` Se essa opção for dada, a barra invertida não atua como um carácter de escape. A barra invertida é considerada como fazendo parte da linha. Em particular, um par barra invertida + newline não pode ser utilizado como uma continuação de linha.

`-s` Modo silencioso. Se a entrada está vindo de um terminal, então os caracteres não são ecoados.

`-t` *timeout*
Faz com que `read` tenha um intervalo e retorne falha se uma linha completa de entrada (ou um número especificado de caracteres) não for lido dentro de *timeout* segundos. *timeout* pode ser um número decimal com uma porção em forma de fração seguinte ao ponto decimal. Essa opção apenas é efetiva se `read` estiver lendo entrada a partir de um terminal, canal de comunicação, ou outro arquivo especial; a opção não tem efeito quando a leitura é feita a partir de arquivos regulares. Se `read` tiver um intervalo de tempo, então `read` salvará qualquer entrada parcial lida na variável especificada por *name*. Se *timeout* for 0, então `read` retorna imediatamente, sem tentar ler e sem dados. O código de saída é 0 se a entrada estiver disponível no descritor de arquivo especificado, e diferente de zero no caso contrário. O código de saída é maior que 128 se o intervalo de tempo for excedido.

`-u` *fd* Lê entrada a partir do descritor de arquivo *fd*.

`readarray`

```
readarray [-n count] [-O origin] [-s count] [-t] [-u fd]
    [-C callback] [-c quantum] [array]
```

Lê linha a partir da entrada padrão para a variável vetor indexada *array*, ou a partir do descritor de arquivo *fd* se a opção `-u` for fornecida.

É um sinônimo para `mapfile`.

`source`

```
source filename
```

Um sinônimo para `.` (veja-se Seção 4.1 [Comandos Internos do Shell Bourne], Página 44).

`type`

```
type [-afptP] [name ...]
```

Para cada *name*, indica como deveria ser interpretado se utilizado como um nome de comando.

Se a opção `-t` for utilizada, `type` imprime uma palavra única a qual é '`alias`', '`function`', '`builtin`', '`file`' ou '`keyword`', se *name* for um apelido; uma função de shell; um comando interno ao shell; um arquivo de disco; ou uma palavra reservada de shell, respectivamente. Se *name* não for encontrada, então nada é impresso e `type` retorna um código de falha.

Se a opção `-p` for utilizada, então `type` ou retorna o nome do arquivo de disco que deveria ser executado, ou nada caso `-t` não retorne '`file`'.

A opção `-P` força uma busca de caminho para cada *name*, mesmo no caso de `-t` não retornar '`file`'.

Se um comando for dividido em pedaços, então `-p` e `-P` imprimem o valor cortado em pedaços, o qual não é necessariamente o arquivo que aparecer primeiro no `$PATH`.

Se a opção `-a` for utilizada, então `type` retorna todos os locais que contém um executável chamado *file*. Isso inclui apelidos e funções, se e somente se a opção `-p` também não for utilizada.

Se a opção `-f` for utilizada, então `type` não tentará encontrar funções de shell, como com o comando interno `command`.

O código de retorno é zero se todos os *names* forem encontrados, e diferente de zero se qualquer deles não for encontrado.

`typeset`

```
typeset [-afFgrxilnrtux] [-p] [name[=value] ...]
```

O comando `typeset` é fornecido para compatibilidade com o shell Korn. Ele é um sinônimo para o comando interno `declare`.

`ulimit`

```
ulimit [-abcdefilmnpqrstuvxHST] [limit]
```

`ulimit` provê controle sobre os recursos disponíveis aos processos inicializados pelo shell, em sistemas que permitem tal controle. Se uma opção for dada, essa opção é interpretada conforme segue:

`-S`	Modifica e relata o limite leve associado com o recurso.
`-H`	Modifica e relata o limite pesado associado com o recurso.
`-a`	Todos os limites atuais são relatados.
`-b`	O tamanho máximo da memória intermediária de socket.
`-c`	O tamanho máximo dos arquivos centrais criados.
`-d`	O tamanho máximo de um segmento de dados do processo.
`-e`	A prioridade máxima do agendamento ("nice").
`-f`	O tamanho máximo de arquivos escritos pelo shell e filhos dele.
`-i`	O número máximo de sinais pendentes.
`-l`	O tamanho máximo que pode ser travado dentro da memória.
`-m`	O tamanho máximo do conjunto residente (muitos sistemas não respeitam esse limite).

`-n` O número máximo de descritores de arquivo abertos (muitos sistemas não permitem que esse valor seja configurado).

`-p` O tamanho da memória intermediária do canal de comunicação.

`-q` O número máximo de bytes nas consultas de mensagem POSIX.

`-r` A prioridade máxima de agendamento de tempo-real.

`-s` O tamanho máximo da pilha.

`-t` A quantidade máxima de tempo de cpu em segundos.

`-u` O número máximo de processos disponíveis para um usuário único.

`-v` A quantidade máxima de memória virtual disponível para o shell, e, em alguns sistemas, para os filhos dele.

`-x` O número máximo de travas de arquivo.

`-T` O número máximo de camadas.

Se *limit* for dado, então a opção `-a` não é utilizada, *limit* é o valor novo os recurso especificado. Os valores *limit* especiais `hard`, `soft`, e `unlimited` representam o atual limite pesado; o atual limite leve; e sem limite, respectivamente. Um limite pesado não pode incrementado por um usuário sem privilégios administrativos uma vez que for configurado; um limite leve pode ser incrementado até o valor do limite pesado. Do contrário, o valor atual do limite leve para o recurso especificado é impresso, a menos que a opção `-H` seja fornecida. Quando da configuração de novos limites, se nem `-H` nem `-S` forem fornecidas, então ambos os limites, o pesado e o leve, são configurados. Se nenhuma opção for dada, então `-f` é presumida. Os valores são em incrementos de 1024-byte, exceto para `-t`, o qual é em segundos; `-p`, o qual é em unidades de blocos de 512-bytes; e `-T`, `-b`, `-n` e `-u`, os quais são valores sem escala.

O código de retorno é zero a menos que uma opção inválida ou argumento seja fornecido, ou um erro ocorra quando da configuração de um novo limite.

`unalias`

```
unalias [-a] [name ... ]
```

Remove cada *name* da lista de apelidos. Se `-a` for fornecida, então todos os apelidos serão removidos. Os apelidos são descritos em Seção 6.6 [Apelidos], Página 96.

4.3 Modificando o Comportamento do Shell

4.3.1 O Comando Interno Set

Esse comando interno é tão complicado que ele merece sua própria seção. `set` te permite modificar os valores das opções de shell e configurar os parâmetros posicionais, ou exibir os nomes e valores das variáveis de shell.

`set`

```
set [--abefhkmnptuvxBCEHPT] [-o option-name] [argument ...]
set [+abefhkmnptuvxBCEHPT] [+o option-name] [argument ...]
```

Se nenhuma opção ou argumento forem fornecidas, então `set` exibe os nomes e valores de todas as variáveis e funções de shell, ordenados de acordo com a localização atual, em um formato que pode ser reutilizado como entrada para configurar ou reconfigurar as variáveis atualmente configuradas. As variáveis somente leitura não podem ser reconfiguradas. No modo POSIX, somente as variáveis de shell são listadas.

Quando opções são fornecidas, elas configuram ou desconfiguram atributos de shell. As opções, se especificadas, tem os seguintes significados:

`-a` Variáveis e funções de marca são modificados ou criados para exportar para o ambiente de comandos subsequentes.

`-b` Faz com que o estado de tarefas de segundo plano terminadas seja relatado imediatamente, ao invés de imprimir antes do próximo prompt primário.

`-e` Sai imediatamente se um canal de comunicação (veja-se Seção 3.2.2 [Canais de Comunicação], Página 8), o qual pode consistir de um único comando simples (veja-se Seção 3.2.1 [Comandos Simples], Página 8), uma lista (veja-se Seção 3.2.3 [Listas], Página 9), ou um comando composto (veja-se Seção 3.2.4 [Comandos Compostos], Página 10) retorna um código diferente de zero. O shell não sai se o comando que falhar for parte de uma lista de comandos imediatamente seguinte a uma palavra chave `while` ou `until`, parte de um teste em uma declaração `if`, parte de qualquer comando comando executado em uma lista `&&` ou `||`, exceto o comando seguinte ao `&&` ou `||` final, qualquer comando em um canal de comunicação menos o último, ou se o código de retorno do comando estiver sendo invertido com `!`. Se um comando composto outro que não um sub-shell retornar um código diferente de zero por causa que um comando falhou enquanto `-e` estava sendo ignorado, então o shell não sai. Um coletor sobre `ERR`, se configurado, é executado antes que o shell saia.

Essa opção se aplica ao ambiente de shell e a cada ambiente de sub-shell separadamente (veja-se Seção 3.7.3 [Ambiente de Execução de Comando], Página 39), e pode fazer com que sub-shells saiam antes de executar todos os comandos no sub-shell.

Se um comando composto ou uma função de shell executar em um contexto onde `-e` estiver sendo ignorada, então nenhum dos comandos executados dentro do comando composto ou corpo da função S afetados pela configuração `-e`, mesmo se `-e` estiver configurada e um comando retornar um código de falha. Se um comando composto ou função de shell configura `-e` quando da execução em um contexto onde `-e` é ignorada, aquela configuração não terá qualquer efeito até que o comando composto ou o comando contendo a chamada de função complete.

`-f` Desabilita a expansão de nome de arquivo (casamento de padrões).

`-h` Localiza e lembra (hash) comandos conforme eles são procurados para execução. Essa opção é habilitada por padrão.

`-k` Todos os argumentos na forma de declarações de atribuição são colocados no ambiente para um comando, não apenas aqueles que precedem o nome do comando.

`-m` Controle de tarefas é habilitado (veja-se Capítulo 7 [Controle de Tarefa], Página 106). Todos os processos executam em um grupo de processos separado. Quando uma tarefa de segundo plano completa, o shell imprime uma linha contendo o código de saída dela.

`-n` Lê comandos, porém não os executa; isso pode ser utilizado para verificar um script e identificar erros de sintaxe. Essa opção é ignorada pelos shells interativos.

`-o` *option-name*
Configura a opção correspondente a *option-name*:

`allexport`
O mesmo que `-a`.

`braceexpand`
O mesmo que `-B`.

`emacs` Utiliza uma interface de edição de linha de comando estilo `emacs` (veja-se Capítulo 8 [Edição de Linha de Comando], Página 110). Isso também afeta a interface de edição utilizada para `read -e`.

`errexit` O mesmo que `-e`.

`errtrace` O mesmo que `-E`.

`functrace`
O mesmo que `-T`.

`hashall` O mesmo que `-h`.

`histexpand`
O mesmo que `-H`.

`history` Habilita o histórico de comandos, conforme descrito em Seção 9.1 [Facilidades do Histórico de Bash], Página 145. Essa opção está ligada por padrão em shells interativos.

`ignoreeof`
Um shell interativo não sairá até que leia EOF.

`keyword` O mesmo que `-k`.

`monitor` O mesmo que `-m`.

`noclobber`
O mesmo que `-C`.

`noexec` O mesmo que `-n`.

`noglob` O mesmo que `-f`.

`nolog` Atualmente ignorado.

`notify` O mesmo que `-b`.

`nounset` O mesmo que `-u`.

`onecmd` O mesmo que `-t`.

`physical` O mesmo que `-P`.

`pipefail` Se configurado, o valor de retorno de um canal de comunicação é o valor do último (mais à direita) comando a sair com um código diferente de zero, ou zero se todos os comandos no canal de comunicação saírem com sucesso. Essa opção é desabilitada por padrão.

`posix` Modifica o comportamento de Bash onde a operação padrão diferir do padrão POSIX para coincidir com o padrão (veja-se Seção 6.11 [O Modo POSIX de Bash], Página 102). Isso é entendido para fazer com que Bash se comporte como um estrito super conjunto daquele padrão.

`privileged` O mesmo que `-p`.

`verbose` O mesmo que `-v`.

`vi` Utiliza uma interface de linha de edição estilo `vi`. Isso também afeta a interface de edição utilizada por `read -e`.

`xtrace` O mesmo que `-x`.

`-p` Liga o modo privilegiado. Neste modo, os arquivos `$BASH_ENV` e `$ENV` não são processados, funções de shell não são herdadas do ambiente, e as variáveis `SHELLOPTS`, `BASHOPTS`, `CDPATH` e `GLOBIGNORE`, se elas aparecerem no ambiente, são ignoradas. Se o shell for inicializado com o id de usuário (grupo) efetivo diferente do id de usuário (grupo) real, e a opção `-p` não for fornecida, essas ações são tomadas e o id de usuário efetivo é configurado para o id de usuário real. Se a opção `-p` for fornecida na inicialização, então o id de usuário efetivo não é reconfigurado. O desligamento dessa opção faz com que os ids efetivos de usuário e grupo sejam configurados para os ids reais de usuário e grupo.

`-t` Sai após a leitura e execução de um comando.

`-u` Trata variáveis e parâmetros desconfigurados outros que não os parâmetros especiais '`@`' ou '`*`' como um erro quando da realização da expansão de parâmetro. Uma mensagem de erro será escrita para a saída de erro padrão, e um shell não interativo sairá.

`-v` Imprime linhas de entrada de shell conforme elas são lidas.

`-x` Imprime um rastro de comandos simples, comandos `for`, comandos `case`, comandos `select`, e comandos `for` aritméticos e os argumentos deles ou listas de palavras associadas após elas serem expandidas e antes de elas serem executadas. O valor da variável `PS4` é expandido e o valor resultante é impresso antes do comando e dos argumentos expandidos dele.

`-B` O shell realizará uma expansão de chave (veja-se Seção 3.5.1 [Expansão de Chave], Página 23). Essa opção está ligada por padrão.

`-C` Evita a sobrescrita de arquivos existentes quando da utilização da redireção de saída utilizando '`>`', '`>&`', e '`<>`'.

`-E` Se configurado, qualquer coletor sobre `ERR` é herdado pelas funções de shell, substituições de comando, e comandos executados em um ambiente de sub-shell. O coletor `ERR` normalmente não é herdado em tais casos.

`-H` Habilita a substituição de histórico estilo '`!`' (veja-se Seção 9.3 [History Interaction], Página 147). Essa opção está ligada por padrão para shell interativos.

`-P` Se configurado, não resolve ligações simbólicas quando da realização de comandos tais como `cd` o qual modifica o diretório atual. O diretório físico é utilizado ao invés. Por padrão, Bash segue a cadeia lógica de diretórios quando da realização de comandos os quais modificam o diretório atual.

Por exemplo, se `/usr/sys` for uma ligação simbólica para `/usr/local/sys` então:

```
$ cd /usr/sys; echo $PWD
/usr/sys
$ cd ..; pwd
/usr
```

Se `set -P` estiver ligado, então:

```
$ cd /usr/sys; echo $PWD
/usr/local/sys
$ cd ..; pwd
/usr/local
```

`-T` Se configurado, quaisquer coletores sobre `DEBUG` e `RETURN` são herdados pelas funções de shell, substituições de comando, e comandos executados em um ambiente de sub-shell. Os coletores `DEBUG` e `RETURN` normalmente não são herdados em tais casos.

`--` Se nenhum argumento seguir essa opção, então os parâmetros posicionais são desconfigurados. Do contrário, os parâmetros posicionais são configurados para os *arguments*, mesmo que alguns deles comecem com um '`-`'.

`-` Sinaliza o final das opções, faz com todos os *arguments* restantes sejam atribuídos aos parâmetros posicionais. As opções `-x` e `-v` são desligadas. Se não existirem argumentos, então os parâmetros posicionais permanecem não modificados.

A utilização de '`+`' ao invés de '`-`' faz com que essas opções sejam desligadas. As opções também podem ser utilizadas quando da invocação do shell. O conjunto atual de opções pode ser encontrado em `$-`.

Os restantes N *arguments* são parâmetros posicionais e são atribuídos, em ordem, a `$1`, `$2`, ... `$N`. O parâmetro especial `#` é configurado para N.

O código de retorno sempre é zero a menos que uma opção inválida seja fornecida.

4.3.2 O Comando Interno Shopt

Esse comando interno permite modificar o comportamento opcional adicional do shell.

`shopt`

```
shopt [-pqsu] [-o] [optname ...]
```

Permuta os valores das configurações que controlam o comportamento opcional do shell. As configurações podem ser ou aquelas listadas abaixo, ou, se a opção `-o` for utilizada, aquelas disponíveis com a opção `-o` para o comando interno `set` (veja-se Seção 4.3.1 [O Comando Interno Set], Página 63). Sem opções, ou com a opção `-p`, uma lista de todas as opções configuráveis é exibida, com uma indicação de se ou não cada está configurada. A opção `-p` faz com a saída seja exibida em uma forma que pode ser reutilizada como entrada. Outras opções tem os seguintes significados:

`-s` Habilita (configura) cada *optname*.

`-u` Desabilita (desconfigura) cada *optname*.

`-q` Suprime a saída normal; o código de retorno indica quando a *optname* está configurada ou desconfigurada. Se múltiplos argumentos *optname* forem dados com `-q`, então o código de retorno é zero se todos *optnames* estiverem habilitados; diferente de zero do contrário.

`-o` Restringe os valores de *optname* àqueles definidos para a opção `-o` para o comando interno `set` (veja-se Seção 4.3.1 [O Comando Interno Set], Página 63).

Se ou `-s` ou `-u` forem utilizadas sem argumentos *optname*, então `shopt` mostra apenas aquelas opções as quais estão configuradas ou desconfiguradas, respectivamente.

A menos que apontado do contrário, as opções `shopt` estão desabilitadas (desligadas) por padrão.

O código de retorno quando das opções de listagem é zero se todas *optnames* estiverem habilitadas, diferente de zero do contrário. Quando configurando ou desconfigurando opções, o código de retorno é zero a menos que uma *optname* não seja uma opção de shell válida.

A lista das opções `shopt` é:

`autocd` Se configurado, um nome de comando que seja o nome de um diretório é executado como se o nome fosse o argumento para o comando `cd`. Essa opção é utilizada apenas por shells interativos.

`cdable_vars`
Se isso estiver configurado, um argumento para o comando interno `cd` que não seja um diretório é presumido que seja o nome de uma variável cujo valor é o diretório para o qual se acessar.

`cdspell` Se configurado, error menores na grafia de um componente de diretório em um comando `cd` serão corrigidos. Os erros verificados são caracteres transpostos, um carácter em falta, e um carácter demasiado. Se uma correção for encontrada, então o cominho corrigido é impresso, e o comando procede. Essa opção é utilizada apenas por shells interativos.

`checkhash`
Se isso estiver configurado, Bash verifica se um comando encontrado na tabela de hash existe antes de tentar executá-lo. Se um comando gravado na tabela não mais existir, uma procura de caminho normal é realizada.

`checkjobs`
Se configurada, Bash lista o estado de quaisquer tarefas paradas e em execução antes de sair de um shell interativo. Se quaisquer tarefas estão em execução, isso faz com que a saída seja adiada até que uma segunda saída seja tentada sem um comando interveniente (veja-se Capítulo 7 [Controle de Tarefa], Página 106). O shell sempre adia a saída se quaisquer tarefas estiverem paradas.

`checkwinsize`
Se configurada, Bash verifica o tamanho da janela após cada comando e, se necessário, atualiza os valores de `LINES` e `COLUMNS`.

`cmdhist` Se configurado, Bash tenta salvar todas as linhas de um comando de múltiplas linhas na mesma entrada de histórico. Isso permite a fácil reedição de comandos de múltiplas linhas.

`compat31` Se configurado, Bash muda o comportamento dele para aquele da versão 3.1 com respeito aos argumentos entre aspas para o operador '`=~`' do comando condicional e com respeito a comparações de sequências de caracteres específicas do local quando da utilização dos operadores '`<`' e '`>`' do comando condicional `[[`. As versões de Bash anteriores a bash-4.1 utilizam colação ASCII e strcmp(3); bash-4.1 e posterior utilizam a sequência de colação do local atual e strcoll(3).

`compat32` Se configurado, Bash muda o comportamento dele para aquele da versão 3.2 com respeito à comparação de sequência de caracteres específica do local quando da utilização dos operadores '`<`' e '`>`' do comando condicional `[[` (veja-se o item anterior).

`compat40` Se configurado, Bash muda o comportamento dele para aquele da versão 4.0 com respeito à comparação de sequência de caracteres específica do local quando da utilização dos operadores '`<`' e '`>`' do comando condicional `[[` (veja-se a descrição de `compat31`) e o efeito da interrupção de uma lista de comandos. As versões 4.0 e posteriores de Bash interrompem a lista como se o shell recebesse a interrupção; versões anteriores continuam com o próximo comando na lista.

`compat41` Se configurado, Bash, quando em modo POSIX, trata uma aspa simples em uma expansão de parâmetro dentro de aspas duplas como um carácter especial. As aspas simples devem necessariamente coincidirem (um eventual número) e os caracteres entre as aspas simples são consideradas encapsuladas por aspas. Esse é o comportamento do modo POSIX durante a versão 4.1. O comportamento padrão do Bash permanece como nas versões anteriores.

`compat42` Se configurado, Bash não processa a sequência de caracteres de substituição na expansão de palavra de substituição de padrão utilizando remoção de aspas.

`complete_fullquote`
Se configurado, Bash encapsula entre aspas todos os meta-caracteres de shell em nomes de arquivo e em nomes de diretório quando da realização da completação. Se não configurado, Bash remove meta-caracteres tais como o sinal de dólar do conjunto de caracteres que serão encapsulados entre aspas em nomes de arquivos completados quando esses meta-caracteres aparecerem em referências de variáveis de shell em palavras sejam completados. Isso significa que os sinais de dólar em nomes de variáveis que expandam para diretórios não serão encapsulados em aspas; entretanto, quaisquer sinais de dólar que aparecerem em nomes de arquivo não serão encapsulados em aspas, entretanto. Isso está ativo apenas quando bash está utilizando barras invertidas para encapsular em aspas nomes de arquivos completados. Essa variável é configurada por padrão, o qual é o comportamento padrão de Bash em versões durante 4.2.

`direxpand`
Se configurado, Bash substitui os nomes de diretório com os resultados da expansão de palavra quando da realização da completação de nome de arquivo. Isso modifica o conteúdo da área de edição do Readline. Se não configurado, Bash tenta preservar o que o usuário digitou.

`dirspell` Se configurado, Bash tenta a correção ortográfica em nomes de diretórios durante a completação de palavra se o nome de diretório inicialmente fornecido não existir.

`dotglob` Se configurado, Bash inclui nomes de arquivo iniciando com um '.' nos resultados da expansão de nome de arquivo.

`execfail` Se isso estiver configurado, um shell não interativo não sairá se ele não puder executar o arquivo especificado como um argumento para o comando interno `exec`. Um shell interativo não sai se `exec` falha.

`expand_aliases`
Se configurado, apelidos são expandidos conforme descrito abaixo sob Apelidos, Seção 6.6 [Apelidos], Página 96. Essa opção está habilitada por padrão para shells interativos.

`extdebug` Se configurado, o comportamento concebido para uso por depuradores é habilitado:

1. A opção `-F` para o comando interno `declare` (veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52) exibe o nome do arquivo fonte e o número da linha correspondente a cada nome de função fornecida como um argumento.
2. Se o comando executado pelo coletor `DEBUG` retornar um valor diferente de zero, então o próximo comando é ignorado e não executado.
3. Se o comando executado pelo coletor `DEBUG` retornar o valor 2, e o shell estiver executando em uma sub-rotina (uma função de shell ou um script de shell executado pelos comandos internos `.` ou `source`, então uma chamada a `return` é simulada.
4. `BASH_ARGC` e `BASH_ARGV` são atualizados conforme descrito nas descrições deles (veja-se Seção 5.2 [Variáveis do Bash], Página 75).
5. O rastreamento de função é habilitado: substituição de comando, funções de shell, e sub-shells invocados com `( command )` herdam os coletores `DEBUG` e `RETURN`.
6. O rastreamento de erro é habilitado: substituição de comando, funções de shell, e sub-shells invocados com `( command )` herdam o coletor `ERR`.

`extglob`
Se configurado, as características de coincidência estendida de padrões descritas acima (veja-se Seção 3.5.8.1 [Coincidência de Modelo], Página 32) são habilitadas.

`extquote` Se configurado, o encapsulamento de aspas `$'string'` e `$"string"` é realizado dentro de expansões `${parameter}` encapsuladas em aspas duplas. Essa opção é habilitada por padrão.

`failglob` Se configurado. padrões os quais falham em coincidir nomes de arquivo durante a expansão de nome de arquivo resultam em um erro de expansão.

`force_fignore`
Se configurado, os sufixos especificados pela variável de shell `FIGNORE` fazem com que palavras sejam ignoradas quando da

realização de completação de palavra ainda que as palavras ignoradas sejam as únicas completações possíveis. Veja-se Seção 5.2 [Variáveis do Bash], Página 75, para uma descrição de `FIGNORE`. Essa opção está habilitada por padrão.

`globasciiranges`
Se configurado, as expressões de intervalo utilizadas em expressões de suporte de coincidência de padrão (veja-se Seção 3.5.8.1 [Coincidência de Modelo], Página 32) se comportam como se estivesse no local C tradicional quando da realização de comparações. Isto é, a sequência de colação de local atual não é levado em consideração, de forma que '`b`' não cotejará entre '`A`' e '`B`', e caracteres ASCII maiúsculos e minúsculos serão cotejados juntos.

`globstar` Se configurado, o padrão '`**`' utilizado em um contexto de expansão de nome de arquivo coincidirá com todos os arquivos e zero ou mais diretórios e subdiretórios. Se o padrão for seguido por uma '`/`', então apenas diretórios e subdiretórios coincidem.

`gnu_errfmt`
Se configurado, as mensagens de erro do shell serão escritas no formato de mensagem de erro padrão GNU.

`histappend`
Se configurado, a lista de histórico é acrescentada ao arquivo nominado pelo valor da variável `HISTFILE` quando o shell sair, ao invés de sobrescrever o arquivo.

`histreedit`
Se configurado, e Readline estiver sendo utilizada, ao usuário é dada a oportunidade de reeditar uma substituição de histórico que falhou.

`histverify`
Se configurado, e Readline estiver sendo utilizada, então os resultados da substituição de histórico não são imediatamente passadas para o analisador de shell. Ao invés disso, a linha resultante é carregada na área de edição da Readline, permitindo maiores modificações.

`hostcomplete`
Se configurado, e Readline estiver sendo utilizada, então Bash tentará realizar uma completação de nome de máquina quando uma palavra contendo um '`@`' estiver sendo completada (veja-se Seção 8.4.6 [Commands For Completion], Página 130). Essa opção está habilitada por padrão.

`huponexit`
Se configurado, Bash enviará `SIGHUP` a todas as tarefas quando um shell de login interativo sair (veja-se Seção 3.7.6 [Sinais], Página 41).

`interactive_comments`
: Permite uma palavra iniciando com '`#`', fazendo com que essa palavra e todos os caracteres restantes naquela linha sejam ignorados em um shell interativo. Essa opção está habilitada por padrão.

`lastpipe` Se configurado, e o controle de tarefas não estiver ativo, o shell executa o último comando de um canal de comunicação não executado em segundo plano no ambiente de shell atual.

`lithist` Se habilitado, e a opção `cmdhist` estiver habilitada, comandos de múltiplas linhas são salvos no histórico com marcadores de novas linhas embutidos ao invés de utilizar separadores de ponto e vírgula onde possível.

`login_shell`
: O shell configura essa opção se ele (shell) foi inicializado como um shell de login (veja-se Seção 6.1 [Invocando o Bash], Página 87). O valor não pode ser modificado.

`mailwarn` Se configurado, e um arquivo que Bash estiver verificando por correspondência eletrônica tiver sido acessado desde a última vez que ele foi acessado, então a mensagem "`A correspondência em` *`mail-file`* `foi lida`" é exibida.

`no_empty_cmd_completion`
: Se configurado, e Readline estiver sendo utilizada, Bash não tentará procurar no `PATH` por possíveis completações quando a completação for tentada em uma linha vazia.

`nocaseglob`
: Se configurado, Bash coincide nomes de arquivos de forma a ignorar a diferença entre maiúsculas e minúsculas quando da realização da expansão de nome de arquivo.

`nocasematch`
: Se configurado, Bash coincide modelos de forma a ignorar a diferença entre maiúsculas e minúsculas quando da realização da operação de coincidência enquanto estiver executando os comandos condicionais `case` ou `[[`.

`nullglob` Se configurado, Bash permite que modelos de nome de arquivo os quais não coincidem com nenhum arquivo sejam expandidos para uma sequência de caracteres nula, ao invés dos próprios modelos.

`progcomp` Se configurado, as facilidades da completação programável (veja-se Seção 8.6 [Complementação Programável], Página 135) são habilitadas. Essa opção está habilitada por padrão.

`promptvars`
: Se configurado, apresenta sequências de caracteres como prompt sujeitas a expansão de parâmetro, substituição de comando, expansão aritmética e remoção de aspas, após ser expandida con-

forme descrito abaixo (veja-se Seção 6.9 [Controlando o Prompt], Página 100). Essa opção está habilitada por padrão.

`restricted_shell`
O shell configura essa opção se ele for inicializado em modo restrito (veja-se Seção 6.10 [O Shell Restrito], Página 101). O valor não pode ser modificado. Isso não é reconfigurado quando os arquivos de inicialização são executados, permitindo que os arquivos de inicialização descubram se o shell é restrito ou não.

`shift_verbose`
Se isso estiver configurado, então o comando interno `shift` imprime uma mensagem de erro quando a contagem de deslocamento exceder o número de parâmetros posicionais.

`sourcepath`
Se configurado, o comando interno `source` utiliza o valor de `PATH` para encontrar o diretório contendo o arquivo fornecido como um argumento. Essa opção está habilitada por padrão.

`xpg_echo` Se configurado, então o comando interno `echo` expande sequências de escape de barra invertida por padrão.

O código de retorno quando da listagem de opções é zero se todos os *optnames* estiverem habilitados, e diferente de zero do contrário. Quando configurando ou desconfigurando opções, o código de retorno é zero a menos que um *optname* não seja uma opção de shell válida.

4.4 Comandos Internos Especiais

Por razões históricas, o padrão POSIX classificou vários comandos internos como *special*. Quando Bash está executando no modo POSIX, os comandos internos especiais diferem de outros comandos internos em três aspectos:

1. Comandos internos especiais são encontrados antes das funções de shell durante a procura de comando.
2. Se um comando interno especial retornar um código de erro, um shell não interativo sai.
3. Declarações de atribuição precedendo o comando permanecem em efeito no ambiente do shell após o comando completar.

Quando Bash não está executando no modo POSIX, estes comandos internos se comportam sem nenhuma diferença do restante dos comandos internos ao Bash. O modo POSIX do Bash está descrito em Seção 6.11 [O Modo POSIX de Bash], Página 102.

Estes são os comandos internos especiais do POSIX:

```
break : . continue eval exec exit export readonly return set
shift trap unset
```

5 Variáveis do Shell

Este capítulo descreve as variáveis de shell que Bash utiliza. Bash automaticamente atribui valores padrão a um número de variáveis.

5.1 Variáveis do Shell Bourne

Bash utiliza certas variáveis de shell da mesma maneira que o shell Bourne. Em alguns casos, Bash atribui um valor padrão para a variável.

`CDPATH` Uma lista de diretórios separados por vírgula utilizado como um caminho de pesquisa para o comando interno `cd`.

`HOME` O diretório home do usuário atual; o padrão para o comando interno `cd`. O valor dessa variável também é utilizado pela expansão de til (veja-se Seção 3.5.2 [Expansão de Til], Página 24).

`IFS` Uma lista de caracteres que separam campos; utilizada quando o shell divide palavras como parte da expansão.

`MAIL` Se esse parâmetro estiver configurado para um nome de arquivo ou nome de diretório e a variável `MAILPATH` não estiver configurada, Bash informa o usuário da chegada de correspondência no arquivo especificado ou no diretório no formato Maildir.

`MAILPATH` Uma lista de nomes de arquivos, separados por vírgula, nos quais o shell periodicamente verifica a existência de correspondência nova. Cada entrada da lista pode especificar a mensagem que é impressa quando uma correspondência nova chega no arquivo de correspondência, separando-se o nome de arquivo da mensagem com um '`?`'. Quando utilizado no texto da mensagem, `$_` expande para o nome do arquivo de correspondência atual.

`OPTARG` O valor do último argumento opção processado pelo comando interno `getopts`.

`OPTIND` O índice do último argumento opção processado pelo comando interno `getopts`.

`PATH` Uma lista de diretórios separados por vírgula nos quais o shell busca por comandos. Um nome de diretório de tamanho zero (nulo) no valor de `PATH` indica o diretório atual. Um nome de diretório nulo pode aparecer como duas vírgulas adjacentes, ou como uma vírgula inicial ou final.

`PS1` A sequência de caracteres primária de prompt. O valor padrão é '`\s-\v\$ `'. Veja-se Seção 6.9 [Controlando o Prompt], Página 100, para a lista completa de sequências de escape que são expandidas antes que `PS1` seja exibida.

`PS2` A sequência de caracteres secundária de prompt. O valor padrão é '`> `'.

5.2 Variáveis do Bash

Estas variáveis são configuradas ou utilizadas por Bash, porém outros shells normalmente não tratam elas especialmente.

Umas poucas variáveis utilizadas por Bash estão descritas em diferentes capítulos: variáveis para controlar as facilidades de controle de tarefas (veja-se Seção 7.3 [Variáveis do Controle de Tarefa], Página 109).

`BASH` O caminho completo utilizado para executar a instância atual de Bash.

`BASHOPTS` Uma lista separada por vírgula das opções habilitadas de shell. Cada palavra na lista é um argumento válido para a opção `-s` ao comando interno `shopt` (veja-se Seção 4.3.2 [O Comando Interno Shopt], Página 68). As opções que aparecem em `BASHOPTS` são aquelas relatadas como '`on`' por '`shopt`'. Se essa variável estiver no ambiente quando Bash inicializar, então cada opção de shell na lista será habilitada antes da leitura de quaisquer arquivos de inicialização. Essa variável é somente leitura.

`BASHPID` Expande para o ID de processo do processo Bash atual. Isso difere de `$$` sob certas circunstâncias, tais como sub-shells que não exigem que Bash seja reinicializado.

`BASH_ALIASES`
Uma variável de vetor associativo cujos membros correspondem à lista interna de apelidos conforme mantida pelo comando interno `alias`. (veja-se Seção 4.1 [Comandos Internos do Shell Bourne], Página 44). Os elementos adicionados a esse vetor aparecem na lista de apelidos; desconfigurar elementos do vetor faz com que os apelidos sejam removidos da lista de apelidos.

`BASH_ARGC`
Uma variável de vetor cujos valores são o número de parâmetros em cada quadro da pilha de chamada de execução do bash atual. O número de parâmetros para a sub-rotina atual (função de shell ou script executado com `.` ou `source`) está no topo da pilha. Quando uma sub-rotina é executada, o número de parâmetros passado é empurrado em `BASH_ARGC`. O shell configura `BASH_ARGC` somente quando estiver em modo de depuração estendido (veja-se Seção 4.3.2 [O Comando Interno Shopt], Página 68 para uma descrição da opção `extdebug` para o comando interno `shopt`).

`BASH_ARGV`
Um variável de vetor contendo todos os parâmetros na pilha de chamada de execução do bash atual. O parâmetro final da última chamada de sub-rotina está no topo da pilha; o primeiro parâmetro da chamada inicial está embaixo. Quando uma sub-rotina é executada, os parâmetros fornecidos são empurrados em `BASH_ARGV`. O shell configura `BASH_ARGV` somente quando estiver em modo de depuração estendido (veja-se Seção 4.3.2 [O Comando Interno Shopt], Página 68 para uma descrição da opção `extdebug` para o comando interno `shopt`).

`BASH_CMDS`
Uma variável vetor associativo cujos membros correspondem à tabela de comandos do hash interno conforme mantida pelo comando interno `hash` (veja-se Seção 4.1 [Comandos Internos do Shell Bourne], Página 44). Os elementos adicionados a esse vetor aparecem na tabela de hash; desconfigurar elementos do vetor faz com que comandos sejam removidos da tabela de hash.

`BASH_COMMAND`
O comando atualmente sendo executado ou prestes a ser executado, a menos que o shell esteja executando um comando como resultado de um coletor, caso no qual é o comando em execução ao tempo do coletor.

`BASH_COMPAT`
O valor é utilizado para configurar o nível de compatibilidade do shell. Veja-se Seção 4.3.2 [O Comando Interno Shopt], Página 68, para uma descrição dos vários níveis de compatibilidade e os efeitos deles. O valor pode ser um número decimal (por exemplo, 4.2) ou um inteiro (por exemplo, 42) correspondente ao nível de compatibilidade desejado. Se `BASH_COMPAT` for desconfigurado ou configurado para a sequência de caracteres vazia, então o nível de compatibilidade é configurado para o padrão para a versão atual. Se `BASH_COMPAT` for configurada para um valor que não seja um dos níveis de compatibilidade válidos, então o shell imprime uma mensagem de erro e configura o nível de compatibilidade para o padrão para a versão atual. Os níveis de compatibilidade válidos correspondem às opções de compatibilidade aceitas pelo comando interno `shopt` descrito acima (por exemplo, *compat42* significa que 4.2 e 42 são valores válidos). A versão atual também é um valor válido.

`BASH_ENV` Se essa variável for configurada quando Bash for invocado para executar um script de shell, então o valor dela é expandido e utilizado como o nome de um arquivo de inicialização a ser lido antes de executar o script. Veja-se Seção 6.2 [Arquivos de Inicialização do Bash], Página 89.

`BASH_EXECUTION_STRING`
O argumento de comando para a opção de invocação `-c`.

`BASH_LINENO`
Uma variável de vetor cujos membros são os números de linha em arquivos fonte onde cada número correspondente de *FUNCNAME* foi invocado. `${BASH_LINENO[$i]}` é o número de linha no arquivo fonte (`${BASH_SOURCE[$i+1]}`) onde `${FUNCNAME[$i]}` foi chamado (ou `${BASH_LINENO[$i-1]}` se referenciado dentro de uma outra função de shell). Utilize `LINENO` para obter o número de linha atual.

`BASH_REMATCH`
Uma variável vetor cujos membros são atribuídos pelo operador binário '`=~`' para o comando condicional `[[` (veja-se Seção 3.2.4.2 [Construtores Condicionais], Página 11). O elemento com índice 0 é a porção da sequência de caracteres que coincide com a expressão regular inteira. O elemento com índice *n* é a porção da sequência de caracteres que coincide com a *n*ésima sub-expressão nos parênteses. Essa variável é somente leitura.

`BASH_SOURCE`
Uma variável vetor cujos membros são os nomes de arquivos fonte onde os nomes de função de shell correspondentes na variável vetor `FUNCNAME` estão definidos. A função de shell `${FUNCNAME[$i]}` está definida no arquivo `${BASH_SOURCE[$i]}` e chamada a partir de `${BASH_SOURCE[$i+1]}`

`BASH_SUBSHELL`
: Incrementada de um dentro de cada sub-shell ou ambiente de sub-shell quando o shell inicia sua execução dentro daquele ambiente. O valor inicial é 0.

`BASH_VERSINFO`
: Uma variável vetor somente leitura (veja-se Seção 6.7 [Vetores], Página 97) cujos membros mantém informação de versão para essa instância de Bash. Os valores atribuídos aos membros do vetor são conforme a seguir:

  `BASH_VERSINFO[0]`
  : O número de versão maior (o *release*).

  `BASH_VERSINFO[1]`
  : O número de versão menor (o *version*).

  `BASH_VERSINFO[2]`
  : O nível de correção.

  `BASH_VERSINFO[3]`
  : A versão de construção.

  `BASH_VERSINFO[4]`
  : O estado de lançamento (por exemplo, *beta1*).

  `BASH_VERSINFO[5]`
  : O valor de `MACHTYPE`.

`BASH_VERSION`
: O número de versão da instância atual de Bash.

`BASH_XTRACEFD`
: Se configurado para um número inteiro correspondente a um descritor de arquivo válido, então Bash escreverá a saída de rastreamento gerada quando '`set -x`' for habilitada para aquele descritor de arquivo. Isso permite que a saída de rastreamento seja separada das mensagens de diagnóstico e erro. O descritor de arquivo é fechado quando `BASH_XTRACEFD` for desconfigurado ou um novo valor for atribuído. Desconfigurar `BASH_XTRACEFD` ou lhe atribuir a sequência de caracteres vazia faz com que a saída de rastreamento seja enviada para a saída de erro padrão. Note que configurar `BASH_XTRACEFD` para 2 (o descritor de arquivo de erro padrão) e então desconfigurá-lo resultará na saída de erro padrão ser fechada.

`CHILD_MAX`
: Configura o número de valores de estado de processos filhos que saíram para que o shell se lembre. Bash não permitirá que esse valor seja decrementado abaixo de um mínimo do obrigado pelo padrão POSIX, e existe um valor minimo (atualmente 8192) que isso não pode exceder. O valor minimo é dependente de sistema.

`COLUMNS`
: Utilizado pelo comando `select` para determinar a largura do terminal quando da impressão de listas de seleção. Automaticamente configurado se a opção `checkwinsize` for habilitada (veja-se Seção 4.3.2 [O Comando Interno Shopt], Página 68), ou em um shell interativo quando do recebimento de um `SIGWINCH`.

`COMP_CWORD`
: Um índice em `${COMP_WORDS}` de uma palavra contendo a posição atual do cursor. Essa variável está disponível apenas em funções de shell invocadas pelas facilidades de completação programáveis (veja-se Seção 8.6 [Complementação Programável], Página 135).

`COMP_LINE`
: A linha de comando atual. Essa variável está disponível somente em funções de shell e comandos externos invocados pelas facilidades de completação programáveis Seção 8.6 [Complementação Programável], Página 135).

`COMP_POINT`
: O índice da posição atual do cursor relativo ao inicio do comando atual. Se a posição atual do cursor estiver no inicio do comando atual, então o valor dessa variável é igual a `${#COMP_LINE}`. Essa variável está disponível somente em funções de shell e comandos externos invocados pelas facilidades de completação programáveis Seção 8.6 [Complementação Programável], Página 135).

`COMP_TYPE`
: Configurado para um valor inteiro correspondente ao tipo de completação tentada que fez com que uma função de completação fosse chamada: *TAB*, para completação normal, '`?`', para listagem de completações após sucessivos tabs, '`!`', para listar alternativas sobre completação de palavra parcial, '`@`', para listar completações, se a palavra não estiver imodificada, ou '`%`', para completação de menu. Essa variável está disponível somente em funções de shell e comandos externos invocados pelas facilidades de completação programáveis Seção 8.6 [Complementação Programável], Página 135).

`COMP_KEY` A chave (ou chave final de uma sequência de chave) utilizada para invocar a função de completação atual.

`COMP_WORDBREAKS`
: O conjunto de caracteres que a biblioteca Readline trata como separadores de palavra quando da realização de completação de palavra. Se `COMP_WORDBREAKS` estiver desconfigurada, então ela perde suas propriedades especiais, mesmo se ela for reconfigurada subsequentemente.

`COMP_WORDS`
: Uma variável vetor consistente de palavras individuais na linha de comando atual. A linha é dividida em palavras como Readline as dividiria, utilizando `COMP_WORDBREAKS` conforme descrito acima. Essa variável está disponível somente em funções de shell invocadas pelas facilidades de completação programáveis (veja-se Seção 8.6 [Complementação Programável], Página 135).

`COMPREPLY`
: Uma variável vetor a partir de quem Bash lê as completações possíveis geradas por uma função de shell invocada pela facilidade de completação programável (veja-se Seção 8.6 [Complementação Programável], Página 135). Cada elemento do vetor contém uma possível completação.

`COPROC` Uma variável vetor criada para manter os descritores de arquivo para saída de e entrada para um co-processo não nomeado (veja-se Seção 3.2.5 [Coprocessos], Página 16).

`DIRSTACK` Uma variável vetor contendo o conteúdo atual da pilha de diretório. Diretórios aparecem na pilha na ordem que eles são exibidos pelo comando interno `dirs`. A atribuição a membros dessa variável vetor pode ser utilizada para modificar diretórios que já estejam na pilha, porém os comandos internos `pushd` e `popd` devem necessariamente ser utilizados para adicionar e para remover diretórios. A atribuição a essa variável não mudará o diretório atual. Se `DIRSTACK` for desconfigurada, então ela perde suas propriedades especiais, mesmo se reconfigurada subsequentemente.

`EMACS` Se Bash encontra essa variável no ambiente quando o shell inicializa com valor '`t`', então Bash assume que o shell está executando em um buffer de shell Emacs e desabilita a edição de linha.

`ENV` Semelhante a `BASH_ENV`; utilizado quando o shell for invocado no modo POSIX (veja-se Seção 6.11 [O Modo POSIX de Bash], Página 102).

`EUID` O id numérico de usuário efetivo do usuário atual. Essa variável é somente leitura.

`FCEDIT` O editor utilizado como um padrão para a opção `-e` para o comando interno `fc`.

`FIGNORE` Uma lista separada por vírgula de sufixos a serem ignorados quando da realização da completação de nome de arquivo. Um nome de arquivo cujo sufixo coincide com uma das entradas em `FIGNORE` é excluído da lista de nomes de arquivos coincididos. Um valor modelo é '`.o:~`'

`FUNCNAME` Uma variável vetor contendo os nomes de todas as funções de shell atualmente na pilha de chamada de execução. O elemento com índice 0 é o nome de qualquer função de shell atualmente em execução. O elemento mais embaixo (aquele com o índice mais alto) é `"main"`. Essa variável existe somente quando uma função de shell está executando. Atribuições a `FUNCNAME` não tem efeito e retornam um código de erro. Se `FUNCNAME` for desconfigurada, então ela perde suas propriedades especiais, mesmo se for reconfigurada subsequentemente.

Essa variável pode ser utilizada com `BASH_LINENO` e `BASH_SOURCE`. Cada elemento de `FUNCNAME` tem elementos correspondentes em `BASH_LINENO` e `BASH_SOURCE` para descrever a pilha de chamada. Por exemplo, `${FUNCNAME[$i]}` foi chamado a partir do arquivo `${BASH_SOURCE[$i+1]}` na linha de número `${BASH_LINENO[$i]}`. O comando interno `caller` exibe a pilha de chamada atual utilizando essa informação.

`FUNCNEST` Se configurada para um valor numérico maior que 0, define um nível máximo de aninhamento de função. Invocações de função que excederem esse nível de aninhamento farão com que o comando atual aborte.

`GLOBIGNORE`
Uma lista separada por vírgula de padrões definidores do conjunto de nomes de arquivo a serem ignorados pela expansão de nome de arquivo. Se um nome

de arquivo coincidido por um modelo de expansão de nome de arquivo também coincide com um dos padrões em `GLOBIGNORE`, então esse nome de arquivo é removido da lista de coincidências.

`GROUPS` Uma variável vetor contendo a lista de grupos dos quais o usuário atual é um membro. Atribuições a `GROUPS` não tem efeito e retornam um código de erro. Se `GROUPS` for desconfigurada, então ela perde as suas propriedades especiais, mesmo se for reconfigurada subsequentemente.

`histchars`
Até três caracteres os quais controlam a expansão de histórico, substituição rápida, e "tokenização" (veja-se Seção 9.3 [History Interaction], Página 147). O primeiro carácter é o carácter de *history expansion*, isto é, o carácter que significa o início de uma expansão de histórico, normalmente '`!`'. O segundo carácter é o carácter que significa "substituição rápida" quando visto como o primeiro carácter em uma linha, normalmente '`^`'. O terceiro carácter opcional é o carácter que indica que o restante da linha é um comentário quando encontrado como o primeiro carácter de uma palavra, usualmente '`#`'. O carácter de comentário do histórico faz com que a substituição de histórico seja saltada para as palavras restantes na linha. Isso não necessariamente faz com que o analisador do shell trate o resto da linha como um comentário.

`HISTCMD` O número do histórico, ou índice na lista de histórico, do comando atual. Se `HISTCMD` for desconfigurada, então ela perde suas propriedades especiais, mesmo se for reconfigurada subsequentemente.

`HISTCONTROL`
Uma lista de valores separados por vírgula que controlam como os comandos são salvos na lista de histórico. Se a lista de valores inclui '`ignorespace`', então as linhas as quais começam com um carácter espaço não são salvas na lista de histórico. Um valor de '`ignoredups`' faz com que as linhas que coincidam com entradas prévias de histórico não sejam salvas. Um valor de '`ignoreboth`' é um atalho para '`ignorespace`' e '`ignoredups`'. Um valor de '`erasedups`' faz com que todas as linhas prévias que coincidam com a linha atual sejam removidas da lista de histórico antes que aquela linha seja salva. Qualquer valor que não esteja na lista acima é ignorado. Se `HISTCONTROL` for desconfigurada, ou não incluir um valor válido, então todas as linhas lidas pelo analisador do shell são salvas na lista de histórico, sujeito ao valor de `HISTIGNORE`. A segunda e subsequentes linhas de um comando composto multilinha não são testadas, e são adicionadas ao histórico independentemente do valor de `HISTCONTROL`.

`HISTFILE` O nome do arquivo para o qual o histórico de comando é salvo. O valor padrão é `~/.bash_history`.

`HISTFILESIZE`
O número máximo de linhas contidas no arquivo de histórico. Quando a essa variável é atribuído um valor, o arquivo de histórico é truncado, se necessário, para conter não mais que aquele número de linhas, removendo-se as entradas mais antigas. O arquivo de histórico também é truncado para esse tamanho após escrevê-lo quando um shell sai. Se o valor for 0, o arquivo de histórico

é truncado para tamanho zero. Os valores não numéricos e valores numéricos menores que zero inibem a truncagem. O shell configura o valor padrão para o valor de `HISTSIZE` após a leitura de quaisquer arquivos de inicialização.

`HISTIGNORE`
: Uma lista de padrões separados por vírgula utilizado para decidir quais linhas de comandos deveriam ser salvos na lista de histórico. Cada padrão é ancorado no inicio da linha e deve necessariamente coincidir com a linha completa (nenhum '`*`' implícito é adicionado). Cada padrão é testado contra a linha após as checagens especificadas por `HISTCONTROL` serem aplicadas. Em adição aos caracteres de coincidência de padrão normais do shell, '`&`' coincide com a linha prévia de histórico. '`&`' pode ser encapsulada utilizando-se um barra invertida; a barra invertida é removida antes da tentativa de coincidência. A segunda e subsequentes linhas de um comando composto multilinha não são testadas, e são adicionadas ao histórico independentemente do valor de `HISTIGNORE`.

  `HISTIGNORE` inclui a função de `HISTCONTROL`. Um padrão de '`&`' é idêntico a `ignoredups`, e um padrão de '`[ ]*`' é idêntico a `ignorespace`. Combinando-se esses dois padrões, separando-se eles com um vírgula, implementa-se a funcionalidade de `ignoreboth`.

`HISTSIZE` O número máximo de comandos a lembrar na lista de histórico. Se o valor for 0, então os comandos não são salvos na lista de histórico. Valores numéricos menores que zero resultam em cada comando sendo salvo na lista de histórico (não existe limite). O shell configura o valor padrão para 500 após a leitura de quaisquer arquivos de configuração.

`HISTTIMEFORMAT`
: Se essa variável estiver configurada e não for nula, então o valor dela é utilizado como uma sequência de caracteres de formato para *strftime* para imprimir a marca temporal associada com cada entrada de histórico exibida pelo comando interno `history`. Se essa variável for configurada, então as marcas temporais são escritas no arquivo de histórico de forma que elas podem ser preservadas entre sessões de shell. Isso utiliza o carácter de comentário de histórico para distinguir marcas temporais de outras linhas de histórico.

`HOSTFILE` Contém o nome de um arquivo no mesmo formato que `/etc/hosts` que deveria ser lido quando o shell precisasse completar um nome de máquina. A lista de possíveis completações de nomes de máquinas pode ser modificada enquanto o shell está em execução; na próxima vez que a completação de nome de máquina for tentada após o valor ser modificado, Bash adiciona o conteúdo do arquivo novo à lista existente. Se `HOSTFILE` for configurada, porém não tiver valor, ou não nominar um arquivo legível, então Bash tenta ler `/etc/hosts` para obter a lista de possíveis completações de nomes de máquina. Quando `HOSTFILE` for desconfigurada, a lista de nomes de máquina é limpa.

`HOSTNAME` O nome da máquina atual.

`HOSTTYPE` Uma sequência de caracteres descrevendo a máquina onde Bash está em execução.

`IGNOREEOF`
: Controla a ação do shell quando do recebimento de um carácter `EOF` como a entrada única. Se configurada, o valor denota o número de caracteres `EOF` consecutivos que podem ser lidos como o primeiro carácter em uma linha de entrada antes que o shell saia. Se a variável existir, porém não tiver um valor numérico (ou não tiver valor), então o padrão é 10. Se a variável não existir, então `EOF` significa o fim de entrada para o shell. Isso está em efeito apenas para shell interativos.

`INPUTRC`
: O nome do arquivo de inicialização de Readline, prevalecendo sobre o padrão `~/.inputrc`.

`LANG`
: Utilizada para determinar a categoria de local para qualquer categoria não selecionada especialmente com uma variável inicializando com `LC_`.

`LC_ALL`
: Essa variável prevalece sobre o valor de `LANG` e qualquer outra variável `LC_` especificando uma categoria de local.

`LC_COLLATE`
: Essa variável determina a ordem de colação utilizada quando da ordenação dos resultados da expansão de nome de arquivo, e determina o comportamento de expressões de intervalo, classes de equivalência, e sequência de colacionamento dentro de expansão de nome de arquivo e coincidências de padrão (veja-se Seção 3.5.8 [Expansão de Nome de Arquivo], Página 32).

`LC_CTYPE`
: Essa variável determina a interpretação de caracteres e o comportamento de classes de caracteres dentro da expansão de nome de arquivo e coincidências de padrão (veja-se Seção 3.5.8 [Expansão de Nome de Arquivo], Página 32).

`LC_MESSAGES`
: Essa variável determina o local utilizado para traduzir sequências de caracteres encapsuladas entre aspas duplas precedidas por um '`$`' (veja-se Seção 3.1.2.5 [Tradução do Locale], Página 7).

`LC_NUMERIC`
: Essa variável determina a categoria de local utilizada para formatação de números.

`LINENO`
: O número da linha no script ou função de shell atualmente em execução.

`LINES`
: Utilizada pelo comando `select` para determinar a largura da coluna para imprimir listas de seleção. Automaticamente configurada se a opção `checkwinsize` estiver habilitada (veja-se Seção 4.3.2 [O Comando Interno Shopt], Página 68), ou em um shell interativo quando do recebimento de um `SIGWINCH`.

`MACHTYPE`
: Uma sequência de caracteres que descreve completamente o tipo de sistema no qual Bash está em execução, no formato padrão GNU *cpu-company-system*

`MAILCHECK`
: Quão frequentemente (em segundos) que o shell deveria verificar a correspondência nos arquivos especificados nas variáveis `MAILPATH` ou `MAIL`. O padrão é 60 segundos. Quando for tempo da checagem da correspondência, o shell faz isso antes da exibição do prompt primário. Se essa variável for desconfigurada,

ou configurada para um valor que não seja um número maior que ou igual a zero, então o shell desabilita a checagem de correspondência.

`MAPFILE` Uma variável vetor criada para manter o texto lido pelo comando interno `mapfile` quando nenhum nome de variável for fornecido.

`OLDPWD` O diretório de trabalho prévio conforme configurado pelo comando interno `cd`.

`OPTERR` Se configurado para o valor 1, então Bash exibe mensagens de erro geradas pelo comando interno `getopts`.

`OSTYPE` Uma sequência de caracteres descrevendo o sistema operacional no qual Bash está em execução.

`PIPESTATUS`
Uma variável vetor (veja-se Seção 6.7 [Vetores], Página 97) contendo uma lista de valores de códigos de saída originados de processos no canal de comunicação mais recentemente executado em segundo plano (o qual pode conter apenas um comando singular).

`POSIXLY_CORRECT`
Se essa variável estiver no ambiente quando Bash iniciar, então o shell entra no modo POSIX (veja-se Seção 6.11 [O Modo POSIX de Bash], Página 102) antes da leitura dos arquivos de inicialização, como se a opção de invocação `--posix` tivesse sido fornecida. Se essa variável for configurada enquanto o shell estiver em execução, então Bash habilita o modo POSIX, como se o comando

```
set -o posix
```

tivesse sido executado.

`PPID` O ID de processo do processo pai do shell. Essa variável é somente leitura.

`PROMPT_COMMAND`
Se configurada, o valor é interpretado como um comando a executar antes da impressão de cada prompt primário (`$PS1`).

`PROMPT_DIRTRIM`
Se configurada para um número maior que zero, então o valor é utilizado como o número de componentes finais de diretórios a reter quando da expansão dos escapes de sequência de caracteres de prompt `\w` e `\W` (veja-se Seção 6.9 [Controlando o Prompt], Página 100). Os caracteres removidos são substituídos com uma elipse.

`PS3` O valor dessa variável é utilizado como o prompt para o comando `select`. Se essa variável não estiver configurada, então o comando `select` sinaliza com '`#?` '

`PS4` O valor é o prompt impresso antes que a linha de comando seja ecoada quando a opção `-x` for configurada (veja-se Seção 4.3.1 [O Comando Interno Set], Página 63). O primeiro carácter de `PS4` é replicado múltiplas vezes, conforme necessário, para indicar níveis múltiplos de sentido. O padrão é '`+` '.

`PWD` O diretório de trabalho atual conforme configurado pelo comando interno `cd`.

`RANDOM` A cada vez que esse parâmetro é referenciado, um número inteiro entre 0 e 32767 é gerado. A atribuição de um valor a essa variável alimenta o gerador de número aleatórios.

`READLINE_LINE`
O conteúdo da memória intermediária de linha do Readline, para uso com '`bind -x`' (veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52).

`READLINE_POINT`
A posição do ponto de inserção na memória intermediária de linha do Readline, para uso com '`bind -x`' (veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52).

`REPLY` A variável padrão para o comando interno `read`.

`SECONDS` Essa variável expande para o número de segundos desde que o shell foi inicializado. A atribuição a essa variável reconfigura a contagem para o valor atribuído, e o valor expandido se torna o valor atribuído mais o número de segundos desde a atribuição.

`SHELL` O nome de caminho completo para o shell é mantido nessa variável de ambiente. Se ela não for configurada quando o shell iniciar, então Bash atribui para ela o nome de caminho completo do shell de login do usuário atual.

`SHELLOPTS`
Uma lista separada por vírgula das opções habilitadas de shell. Cada palavra na lista é um argumento válido para a opção `-o` para o comando interno `set` (veja-se Seção 4.3.1 [O Comando Interno Set], Página 63). As opções que aparecem em `SHELLOPTS` são aquelas relatadas como '`on`' por '`set -o`'. Se essa variável estiver no ambiente quando Bash inicializar, então cada opção de shell na lista será habilitada antes da leitura de quaisquer arquivos de inicialização. Essa variável é somente leitura.

`SHLVL` Incrementada de um cada vez que uma instância nova de Bash for inicializada. Isso é entendido para ser uma contagem de quão profundos os seus shells Bash estão aninhados.

`TIMEFORMAT`
O valor desse parâmetro é utilizado como uma sequência de caracteres de formato especificando como a informação de temporização para canais de comunicação com a palavra reservada `time` deveria ser exibida. O carácter '`%`' introduz uma sequência de escape que é expandida para um valor de tempo ou outra informação. As sequências de escape e os significados delas são conforme segue; as chaves denotam porções opcionais.

`%%` Um '`%`' literal.

`%[`*p*`][l]R` O tempo decorrido em segundos.

`%[`*p*`][l]U` O número de segundos de CPU gastos em modo de usuário.

`%[`*p*`][l]S` O número de segundos de CPU gastos em modo de sistema.

`%P` A percentagem de CPU, computada como (%U + %S) / %R.

O *p* opcional é um dígito especificando a precisão, o número de dígitos fracionários após um ponto decimal. Um valor de 0 faz com que nenhum ponto decimal ou fração seja exibida. No máximo três posições após o ponto decimal podem ser especificados; valores de *p* maiores que 3 são modificados para 3. Se *p* não for especificado, então o valor 3 é utilizado.

O `l` opcional especifica um formato longo, incluindo minutos, da forma *MM*m*SS.FF*s. O valor de *p* determina quando ou não a fração é incluída.

Se essa variável não for configurada, então Bash atua como se ela tivesse o valor

```
$'\nreal\t%3lR\nuser\t%3lU\nsys\t%3lS'
```

Se o valor é nulo, então nenhuma informação de temporização é exibida. Um marcador de nova linha final é adicionado quando a sequência de caracteres de formato for exibida.

`TMOUT` Se configurada para um valor maior que zero, então `TMOUT` é tratada como o intervalo de tempo padrão para o comando interno `read` (veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52). O comando `select` (veja-se Seção 3.2.4.2 [Construtores Condicionais], Página 11) termina se a entrada não chegar após `TMOUT` segundos, quando a entrada estiver vindo de um terminal.

Em um shell interativo, o valor é interpretado como sendo o número de segundos a se aguardar por uma linha de entrada após a emissão do prompt primário. Bash termina após aguardar por aquele número de segundos se uma linha completa de entrada não chegar.

`TMPDIR` Se configurado, Bash utiliza o seu valor como o nome de um diretório no qual Bash cria arquivos temporários para o uso do shell.

`UID` O id numérico real de usuário do usuário atual. Essa variável é somente leitura.

6 Características de Bash

Este capítulo descreve características únicas do Bash.

6.1 Invocando o Bash

`bash [long-opt] [-ir] [-abefhkmnptuvxdBCDHP] [-o option] [-O shopt_option] [argument ...]`
`bash [long-opt] [-abefhkmnptuvxdBCDHP] [-o option] [-O shopt_option] -c string [argument ...]`
`bash [long-opt] -s [-abefhkmnptuvxdBCDHP] [-o option] [-O shopt_option] [argument ...]`

Todas as opções de carácter simples utilizadas com o comando interno `set` (veja-se Seção 4.3.1 [O Comando Interno Set], Página 63) podem ser utilizadas como opções quando o shell for invocado. Adicionalmente, existem várias opções multi-carácter que você pode utilizar. Essas opções devem necessariamente aparecer na linha de comando antes das opções de carácter simples serem reconhecidas.

`--debugger`
: Organiza para que o perfil de depurador seja executado antes que o shell inicialize. Liga o modo estendido de depuração (veja-se Seção 4.3.2 [O Comando Interno Shopt], Página 68 para uma descrição da opção `extdebug` para o comando interno `shopt`.

`--dump-po-strings`
: Uma lista de todas as sequências de caracteres encapsuladas em aspas duplas precedida por '`$`' é impressa na saída padrão no formato de arquivo PO (objeto portável) do GNU `gettext`. Equivalente a `-D` exceto para o formato de saída.

`--dump-strings`
: Equivalente a `-D`.

`--help`
: Exibe uma mensagem útil na saída padrão e sai com sucesso.

`--init-file` *filename*
`--rcfile` *filename*
: Executa comandos originados de *filename* (ao invés de `~/.bashrc`) em um shell interativo.

`--login`
: Equivalente a `-l`.

`--noediting`
: Não utilizar a biblioteca GNU Readline (veja-se Capítulo 8 [Edição de Linha de Comando], Página 110) para ler linhas de comando quando o shell for interativo.

`--noprofile`
: Não carrega o arquivo de inicialização geral de sistema `/etc/profile` ou quaisquer dos arquivos pessoais de inicialização `~/.bash_profile`, `~/.bash_login`, ou `~/.profile` quando Bash for invocado como um shell de login.

`--norc`
: Não lê o arquivo de inicialização `~/.bashrc` em um shell interativo. Isso está ativado por padrão se o shell for invocado como `sh`.

`--posix` Modifica o comportamento de Bash onde a operação padrão diferir do padrão POSIX para coincidir com esse padrão. Isso é entendido como um modo de fazer com que Bash se comporte como um super conjunto estrito daquele padrão. Veja-se Seção 6.11 [O Modo POSIX de Bash], Página 102, para uma descrição do modo POSIX de Bash.

`--restricted`
Faz do shell um shell restrito (veja-se Seção 6.10 [O Shell Restrito], Página 101).

`--verbose`
Equivalente a `-v`. Imprime as linhas de entrada do shell conforme elas sejam lidas.

`--version`
Mostra informação de versão para esta instância de Bash na saída padrão e sai com sucesso.

Existem várias opções de carácter simples que podem ser fornecidas na invocação as quais não estão disponíveis com o comando interno `set`.

`-c` Lê e executa comandos a partir da primeira não opção *argument* após o processamento das opções, então sai. Quaisquer argumentos remanescentes são atribuídos aos parâmetros posicionais, iniciando com `$0`.

`-i` Força o shell a executar interativamente. Os shell interativos estão descritos em Seção 6.3 [Shells Interativos], Página 91.

`-l` Faz com que este shell atue como se ele tivesse sido invocado diretamente por login. Quando o shell for interativo, isso é equivalente a inicializar um shell de login com '`exec -l bash`'. Quando o shell não for interativo, os arquivos de inicialização de shell de login serão executados. '`exec bash -l`' ou '`exec bash --login`' substituirão o shell atual com um shell de login de Bash. Veja-se Seção 6.2 [Arquivos de Inicialização do Bash], Página 89, para uma descrição do comportamento especial de um shell de login.

`-r` Faz do shell um shell restrito (veja-se Seção 6.10 [O Shell Restrito], Página 101).

`-s` Se essa opção estiver presente, ou se nenhum argumento restar após o processamento de opção, então os comandos são lidos a partir da entrada padrão. Essa opção permite que os parâmetros posicionais sejam configurados quando da invocação de um shell interativo.

`-D` Uma lista de todas as sequências de caracteres encapsuladas em aspas duplas precedida por '`$`' é impressa na saída padrão. Essas são as sequências de caracteres que são objeto de tradução de linguagem quando o local atual não for `C` ou `POSIX` (veja-se Seção 3.1.2.5 [Tradução do Locale], Página 7). Isso implica a opção `-n`; nenhum comando será executado.

`[-+]O [`*shopt_option*`]`
shopt_option é uma das opções de shell aceitas pelo comando interno `shopt` (veja-se Seção 4.3.2 [O Comando Interno Shopt], Página 68). Se *shopt_option* estiver presente, então `-O` configura o valor dessa opção; `+O` desconfigura. Se *shopt_option* não for fornecida, então os nomes e valores das opções de shell

aceitas por `shopt` são impressas na saída padrão. Se a opção de invocação for `+O`, então a saída é exibida em um formato que pode ser reutilizado como entrada.

`--` Um `--` sinaliza o fim das opções e desabilita o processamento adicional de opção. Quaisquer argumentos após o `--` são tratados como nomes de arquivos e argumentos.

Um shell de *login* é um cujo primeiro carácter do argumento zero é '`-`', ou um invocado com a opção `--login`.

Um shell *interactive* é um iniciado sem argumentos não opção, a menos que `-s` seja especificado, sem especificar a opção `-c`, e cuja saída e entrada são ambas conectadas a terminais (conforme determinado por `isatty(3)`), ou um iniciado com a opção `-i`. Veja-se Seção 6.3 [Shells Interativos], Página 91, para mais informação.

Se restarem argumentos após o processamento de opção, e nem a opção `-c` nem a `-s` forem fornecidas, então o primeiro argumento é presumido que seja o nome de um arquivo contendo comandos de shell (veja-se Seção 3.8 [Scripts de Shell], Página 42). Quando Bash for invocado dessa forma, `$0` é configurado para o nome do arquivo, e os parâmetros posicionais são configurados para os argumentos restantes. Bash lê e executa comandos a partir desse arquivo, então sai. O código de saída de Bash é o código de saída do último comando executado no script. Se nenhum comando for executado, o código de saída é 0.

6.2 Arquivos de Inicialização do Bash

Esta seção descreve como Bash executa seus arquivos de inicialização. Se quaisquer dos arquivos existirem, porém não puderem ser lidos, então Bash reporta um erro. Os sinais de til são expandidos em nomes de arquivos conforme descrito acima sob Expansão de til (veja-se Seção 3.5.2 [Expansão de Til], Página 24).

Shell interativos são descritos em Seção 6.3 [Shells Interativos], Página 91.

Invocado como um shell de login interativo, ou com `--login`

Quando Bash for invocado como um shell de login interativo, ou como um shell não interativo com a opção `--login`, ele primeiro lê e executa comandos a partir do arquivo `/etc/profile`, se esse arquivo existir. Após a leitura daquele arquivo, Bash procura por `~/.bash_profile`, `~/.bash_login`, e `~/.profile`, nessa exata ordem, e lê e executa comandos a partir do primeiro que existir e for legível. A opção `--noprofile` pode ser utilizada quando o shell for inicializado para inibir esse comportamento.

Quando um shell de login sai, Bash lê e executa comandos a partir do arquivo `~/.bash_logout`, se ele existir.

Invocado como um shell interativo de não-login

Quando um shell interativo que não é um shell de login é inicializado, Bash lê e executa comandos a partir de `~/.bashrc`, se esse arquivo existir. Isso pode ser inibido utilizando-se a opção `--norc`. A opção `--rcfile` *`file`* forçará Bash a ler e executar comandos a partir de *file* ao invés de `~/.bashrc`.

Assim, tipicamente, o seu `~/.bash_profile` contém a linha

```
if [ -f ~/.bashrc ]; then . ~/.bashrc; fi
```

após (ou antes) quaisquer inicializações específicas de login.

Invocado não-interativamente

Quando Bash for inicializado não-interativamente, para executar um script de shell, por exemplo, ele procura pela variável `BASH_ENV` no ambiente, expande o valor dela se ela aparece lá, e utiliza o valor expandido como o nome de um arquivo a ler e executar. Bash se comporta como se o comando seguinte fosse executado:

```
if [ -n "$BASH_ENV" ]; then . "$BASH_ENV"; fi
```

porém o valor da variável `PATH` não é utilizado para buscar pelo nome de arquivo.

Conforme informado acima, se um shell não-interativo for invocado com a opção `--login`, então Bash tenta ler e executar comandos a partir dos arquivos de inicialização de shell de login.

Invocado com nome `sh`

Se Bash for invocado com o nome `sh`, então ele tenta imitar o comportamento de inicialização de versões históricas de `sh` tão aproximadamente quanto for possível, enquanto se conformando ao padrão POSIX também.

Quando invocado como um shell de login interativo, ou como um shell não-interativo com a opção `--login`, Bash primeiro tenta ler e executar comandos a partir de `/etc/profile` e `~/.profile`, nessa exata ordem. A opção `--noprofile` pode ser utilizada para inibir esse comportamento. Quando invocado como um shell interativo com o nome `sh`, Bash procura pela variável `ENV`, expande o valor dela se estiver definido, e utiliza o valor expandido como o nome de um arquivo a ler e executar. Uma vez que um shell invocado como `sh` não tenta ler e executar comandos a partir de quaisquer outros arquivos de inicialização, a opção `--rcfile` não tem efeito. Um shell não-interativo invocado com o nome `sh` não tenta ler quaisquer outros arquivos de inicialização.

Quando invocado como `sh`, Bash entra no modo POSIX após os arquivos de inicialização serem lidos.

Invocado no modo POSIX

Quando Bash for inicializado no modo POSIX, como com a opção de linha de comando `--posix`, ele segue o padrão POSIX para arquivos de inicialização. Nesse modo, os shells interativos expandem a variável `ENV` e comandos são lidos e executados a partir do arquivo cujo nome for o valor expandido. Nenhum outro arquivo de inicialização é lido.

Invocado por daemon de shell remoto

Bash tenta determinar quando está sendo executado com sua entrada padrão conectada a uma conexão de rede, como quando executado por daemon de shell remoto, usualmente `rshd`, ou o daemon de shell seguro `sshd`. Se Bash determina que está sendo executado dessa forma, então ele lê e executa comandos a partir de `~/.bashrc`, se esse arquivo existir e for legível. Bash não fará isso se invocado como `sh`. A opção `--norc` pode ser utilizada para inibir esse comportamento, e a opção `--rcfile` pode ser utilizada para forçar que outro arquivo seja lido, porém nem `rshd` nem `sshd` geralmente invocam o shell com tais opções ou permitem que sejam especificadas.

Invocado com UID/GIDs efetiva and real diferentes

Se Bash for inicializado com o id de usuário efetivo (grupo) diferente do id de usuário real (grupo), e a opção `-p` não for fornecida, nenhum arquivo de inicialização é lido, funções de shell são herdadas do ambiente, as variáveis `SHELLOPTS`, `BASHOPTS`, `CDPATH`, e `GLOBIGNORE`, se elas aparecerem no ambiente, são ignoradas, e o id de usuário efetivo é configurado para o id de usuário real. Se a opção `-p` for fornecida na invocação, então o comportamento de inicialização é o mesmo, porém o id de usuário efetivo não é reconfigurado.

6.3 Shells Interativos

6.3.1 O Que é um Shell Interativo?

Um shell interativo é aquele inicializado sem argumentos não opção, a menos que `-s` seja especificada, sem especificar a opção `-c`, e cuja entrada e saída de erro estão ambas conectadas a terminais (conforme determinado por `isatty(3)`), ou aquele inicializado com a opção `-i`.

Um shell interativo geralmente lê a partir de e escreve para um terminal de usuário.

A opção de invocação `-s` pode ser utilizada para configurar os parâmetros posicionais quando um shell interativo for inicializado.

6.3.2 Este Shell é Interativo?

Para determinar dentro de um script de inicialização quando ou não Bash está executando interativamente, teste o valor do parâmetro especial '`-`'. Ele contém `i` quando o shell é interativo. Por exemplo:

```
case "$-" in
*i*) echo This shell is interactive ;;
*) echo This shell is not interactive ;;
esac
```

Alternativamente, scripts de inicialização podem examinar a variável `PS1`; ela é desconfigurada em shells não interativos, e configurada em shells interativos. Assim:

```
if [ -z "$PS1" ]; then
        echo This shell is not interactive
else
        echo This shell is interactive
fi
```

6.3.3 Comportamento de Shell Interativo

Quando o shell está executando interativamente, ele modifica o comportamento dele de várias maneiras.

1. Arquivos de inicialização são lidos e executados conforme descrito em Seção 6.2 [Arquivos de Inicialização do Bash], Página 89.
2. O Controle de Tarefas (veja-se Capítulo 7 [Controle de Tarefa], Página 106) é habilitado por padrão. Quando o controle de tarefa está em efeito, Bash ignora os sinais de controle de tarefa gerados via teclado `SIGTTIN`, `SIGTTOU`, e `SIGTSTP`.

3. Bash expande e exibe `PS1` antes de ler a primeira linha de um comando, e expande e exibe `PS2` antes de ler a segunda e subsequentes linhas de um comando multilinha.
4. Bash executa o valor da variável `PROMPT_COMMAND` como um comando antes de imprimir o prompt primário, `$PS1` (veja-se Seção 5.2 [Variáveis do Bash], Página 75).
5. Readline (veja-se Capítulo 8 [Edição de Linha de Comando], Página 110) é utilizada para ler comandos a partir do terminal do usuário.
6. Bash inspeciona o valor da opção `ignoreeof` para `set -o` ao invés de sair imediatamente quando receber um `EOF` na sua entrada padrão quando da leitura de um comando (veja-se Seção 4.3.1 [O Comando Interno Set], Página 63).
7. Histórico de Comandos (veja-se Seção 9.1 [Facilidades do Histórico de Bash], Página 145) e expansão de histórico (veja-se Seção 9.3 [History Interaction], Página 147) são habilitados por padrão. Bash salvará o histórico de comandos no arquivo nomeado por `$HISTFILE` quando um shell com histórico habilitado sai.
8. Expansão de Apelidos (veja-se Seção 6.6 [Apelidos], Página 96) é realizado por padrão.
9. Na ausência de quaisquer coletores, Bash ignora `SIGTERM` (veja-se Seção 3.7.6 [Sinais], Página 41).
10. Na ausência de quaisquer coletores, `SIGINT` é capturada e manuseada (veja-se Seção 3.7.6 [Sinais], Página 41). `SIGINT` interromperá alguns comandos internos do shell.
11. Um shell de login interativo envia um `SIGHUP` a todas as tarefas na saída se a opção de shell `huponexit` tiver sido habilitada (veja-se Seção 3.7.6 [Sinais], Página 41).
12. A opção de invocação `-n` é ignorada, e '`set -n`' não tem efeito (veja-se Seção 4.3.1 [O Comando Interno Set], Página 63).
13. Bash verificará a correspondência periodicamente, dependendo dos valores das variáveis de shell `MAIL`, `MAILPATH`, e `MAILCHECK` (veja-se Seção 5.2 [Variáveis do Bash], Página 75).
14. Os erros de expansão, devido a referências a variáveis de shell não checadas após '`set -u`' tiver sido habilitado, não fazem com que o shell saia (veja-se Seção 4.3.1 [O Comando Interno Set], Página 63).
15. O shell não sairá em erros de expansão causados por *var* sendo desconfigurada ou nula nas expansões `${var:?word}` (veja-se Seção 3.5.3 [Expansão de Parâmetro de Shell], Página 25).
16. Erros de redireção encontrados por comandos internos ao shell não farão com que o shell saia.
17. Quando em execução no modo POSIX, um comando interno especial que retornar um código de erro não fará com que o shell saia (veja-se Seção 6.11 [O Modo POSIX de Bash], Página 102).
18. Um `exec` falho não fará com que o shell saia (veja-se Seção 4.1 [Comandos Internos do Shell Bourne], Página 44).
19. Erros de sintaxe de analisador não fazem com que o shell saia.
20. Correções ortográficas simples para argumentos de diretório para o comando interno `cd` estão habilitados por padrão (veja-se a descrição da opção `cdspell` para o comando interno `shopt` em Seção 4.3.2 [O Comando Interno Shopt], Página 68).

21. O shell verificará o valor da variável `TMOUT` e sai se um comando não for lido dentro do número especificado de segundos após a impressão de `$PS1` (veja-se Seção 5.2 [Variáveis do Bash], Página 75).

6.4 Expressões Condicionais de Bash

Expressões condicionais são utilizadas pelo comando composto `[[` e pelos comandos internos `test` e `[`.

Expressões podem ser unárias ou binárias. Expressões unárias são frequentemente utilizadas para examinar a situação de um arquivo. Existem operadores de sequências de caracteres e operadores de comparação numérica também. Se o argumento *file* para um dos primários for da forma `/dev/fd/`*N*, então o descritor de arquivo *N* é verificado. Se o argumento *file* para um dos primários for um destes: `/dev/stdin`, `/dev/stdout`, ou `/dev/stderr`, então o descritor 0, 1 ou 2, respectivamente, é verificado.

Quando utilizado com `[[`, os operadores '`<`' e '`>`' ordenam lexicograficamente utilizando o local atual. O comando `test` emprega ordenamento ASCII.

menos que especificado de outra maneira, primários que operam sobre arquivos seguem links simbólicos e operam sobre o alvo do link, ao invés do próprio link.

`-a` *file* Verdadeiro se *file* existe.

`-b` *file* Verdadeiro se *file* existe e for um arquivo especial de bloco.

`-c` *file* Verdadeiro se *file* existe e for um arquivo especial de carácter.

`-d` *file* Verdadeiro se *file* existe e for um diretório.

`-e` *file* Verdadeiro se *file* existe.

`-f` *file* Verdadeiro se *file* existe e for um arquivo regular.

`-g` *file* Verdadeiro se *file* existe e seu bit do conjunto de id de grupo estiver configurado.

`-h` *file* Verdadeiro se *file* existe e for um link simbólico.

`-k` *file* Verdadeiro se *file* existe e seu bit "sticky" estiver configurado.

`-p` *file* Verdadeiro se *file* existe e for um canal de comunicação nomeado (FIFO).

`-r` *file* Verdadeiro se *file* existe e for legível.

`-s` *file* Verdadeiro se *file* existe e tiver um tamanho maior que zero.

`-t` *fd* Verdadeiro se o descritor de arquivo *fd* estiver aberto e se refere a um terminal.

`-u` *file* Verdadeiro se *file* existe e seu bit do conjunto de id do usuário estiver configurado.

`-w` *file* Verdadeiro se *file* existe e for possível a escrita.

`-x` *file* Verdadeiro se *file* existe e for executável.

`-G` *file* Verdadeiro se *file* existe e for de propriedade do id efetivo de grupo.

`-L` *file* Verdadeiro se *file* existe e for um link simbólico.

`-N` *file* Verdadeiro se *file* existe e tiver sido modificado desde quando ele foi lido mais recentemente.

`-O` *file* Verdadeiro se *file* existe e for de propriedade do id efetivo de usuário.

`-S` *file* Verdadeiro se *file* existe e for um socket.

file1 `-ef` *file2*
Verdadeiro se *file1* e *file2* se referem aos mesmos números de dispositivo e inode.

file1 `-nt` *file2*
Verdadeiro se *file1* for mais novo (de acordo com a data de modificação) que *file2*, ou se *file1* existe e *file2* não.

file1 `-ot` *file2*
Verdadeiro se *file1* for mais antigo que *file2*, ou se *file2* existe e *file1* não.

`-o` *optname*
Verdadeiro se a opção de shell *optname* estiver habilitada. A lista de opções aparece na descrição da opção `-o` para o comando interno `set` (veja-se Seção 4.3.1 [O Comando Interno Set], Página 63).

`-v` *varname*
Verdadeiro se a variável de shell *varname* estiver configurada (tiver recebido um valor).

`-R` *varname*
Verdadeiro se a variável de shell *varname* estiver configurada e for uma referência de nome.

`-z` *string* Verdadeiro se o comprimento de *string* for zero.

`-n` *string*
string Verdadeiro se o comprimento de *string* for diferente de zero.

string1 `==` *string2*
string1 `=` *string2*
Verdadeiro se as sequências de caracteres forem iguais. Quando utilizado com o comando `[[`, isso realiza a coincidência de padrão conforme descrito acima (veja-se Seção 3.2.4.2 [Construtores Condicionais], Página 11).

'`=`' deveria ser utilizado com o comando `test` para conformação com o padrão POSIX.

string1 `!=` *string2*
Verdadeiro se as sequências de caracteres forem diferentes.

string1 `<` *string2*
Verdadeiro se *string1* figurar antes de *string2* lexicograficamente.

string1 `>` *string2*
Verdadeiro se *string1* figurar após *string2* lexicograficamente.

arg1 `OP` *arg2*
`OP` is one of '`-eq`', '`-ne`', '`-lt`', '`-le`', '`-gt`', ou '`-ge`'. Esses operadores aritméticos binários retornam verdadeiro se *arg1* for igual a, diferente de, menor que, menor que ou igual a, maior que, ou maior que ou igual a *arg2*, respectivamente. *Arg1* and *arg2* podem ser número inteiros positivos ou negativos.

6.5 Aritmética de Shell

O shell permite que expressões aritméticas sejam avaliadas, como uma das expansões de shell ou pelo `let` e a opção `-i` para os comandos internos `declare`.

A avaliação é feita em números inteiros de tamanho fixo sem verificação de excesso, apesar que a divisão por 0 é coletada e rotulada como um erro. Os operadores e suas precedências, associatividade, e valores são os mesmos que na linguagem C. A seguinte lista de operadores está agrupada em níveis de operadores de igual precedência. Os níveis estão listados em ordem de precedência decrescente.

`id++ id--` variável pós-incremento e pós-decremento

`++id --id` variável pré-incremento e pré-decremento

`- +` unário menos e mais

`! ~` lógico e negação bit a bit

`**` exponenciação

`* / %` multiplicação, divisão, restante

`+ -` adição, subtração

`<< >>` deslocamentos bit a bit a esquerda e a direita

`<= >= < >` comparação

`== !=` igualdade e desigualdade

`&` E bit a bit

`^` OU exclusivo bit a bit

`|` OU bit a bit

`&&` E lógico

`||` OU lógico

`expr ? expr : expr`
operador condicional

`= *= /= %= += -= <<= >>= &= ^= |=`
atribuição

`expr1 , expr2`
vírgula

As variável do shell funcionam como operandos; expansão de parâmetro é realizada antes que a expressão seja avaliada. Dentro de uma expressão, variáveis de shell também podem ser referenciadas pelo nome sem se utilizar a sintaxe de expansão de parâmetro. Uma variável de shell que é nula ou desconfigurada dá como resultado 0 quando referenciada pelo nome sem se utilizar a sintaxe de expansão de parâmetro. O valor de uma variável é avaliado como uma expressão aritmética quando ela for referenciada, ou quando uma variável para a qual tenha sido dado o atributo *integer* utilizando '`declare -i`' for atribuído um valor. Um valor nulo resulta em 0. Uma variável de shell precisa não ter o atributo *integer* dela ligado para ser utilizada em uma expressão.

Constantes com um 0 inicial são interpretadas como números octais. Um '`0x`' ou '`0X`' denota um número hexadecimal. Do contrário, os números tomam a forma [*base*`#`]*n*, onde o opcional *base* é um número decimal compreendido entre 2 e 64, representando a base aritmética, e *n* é um número na naquela base. Se *base*`#` for omitida, então a base 10 é utilizada. Quando se especificar *n*, os dígitos maiores que 9 são representados por letras minúsculas, pelas letras maiúsculas, '`@`', e '`_`', nessa exata ordem. Se *base* for menor que ou igual a 36, então as letras minúsculas e maiúsculas podem ser utilizadas indistintamente para representar números entre 10 e 35.

Os operadores são avaliados na ordem de precedência. Sub-expressões entre parênteses são avaliadas primeiro e podem substituir as regras de precedência acima.

6.6 Apelidos

Apelidos *Apelidos* permitem que uma sequência de caracteres seja substituída por uma palavra quando forem utilizados como a primeira palavra de um comando simples.

O shell mantém uma lista dos apelidos, que podem ser configurados ou desconfigurados com os comandos internos `alias` e `unalias`.

A primeira palavra de cada comando simples, se não encapsulados entre aspas, é verificada para ser se ela tem um apelido. Se tiver, essa palavra é substituída pelo texto do apelido. Os caracteres '`/`', '`$`', '`‘`', '`=`' e quaisquer dos meta-caracteres do shell ou caracteres de encapsulamento de aspas listados acima podem não aparecer em um nome de apelido. O texto de substituição pode conter qualquer entrada de shell válida, incluindo meta-caracteres de shell. A primeira palavra do texto de substituição é testado para apelidos, porém uma palavra que for idêntica a um apelido sendo expandido não é expandida uma segunda vez. Isso significa que alguém pode apelidar `ls` para `"ls -F"`, por exemplo, e Bash não tenta recursivamente expandir o texto de substituição. Se o último carácter do valor do apelido for um *blank*, então a próxima palavra de comando seguinte ao apelido também é checada para expansão de apelido.

Os apelidos são criados e listados com o comando `alias`, e removidos com o comando `unalias`.

Não existe um mecanismo para se utilizar argumentos em um texto de substituição, como em `csh`. Se argumentos forem necessários, então uma função de shell deveria ser utilizada (veja-se Seção 3.3 [Funções de Shell], Página 18).

Os apelidos não são expandidos quando o shell não é interativo, a menos que a opção de shell `expand_aliases` seja configurada utilizando `shopt` (veja-se Seção 4.3.2 [O Comando Interno Shopt], Página 68).

As regras relativas à definição e uso de apelidos são de alguma maneira confusas. Bash sempre lê pelo menos uma linha completa de entrada antes de executar quaisquer dos comandos naquela linha. Os apelidos são expandidos quando um comando é lido, não quando ele é executado. Portanto, uma definição de apelido que aparece na mesmo linha como um outro comando não tem efeito até que a próxima linha de entrada seja lida. Os comandos que se seguirem à definição de apelidos naquela linha não são afetados pelo novo apelido. Esse comportamento também é um problema quando funções são executadas. Apelidos são expandidos quando uma definição de função é lida, não quando a função é executada, pois uma definição de função é ela própria um comando composto. Como uma consequência, apelidos definidos em uma função não estão disponíveis até após aquela

função ser executada. Por segurança, sempre coloque definições de apelidos em uma linha separada, e não utilize `alias` em comandos compostos.

Para quase todos os propósitos, as funções de shell são preferidas a apelidos.

6.7 Vetores

Bash provê variáveis vetor indexado unidimensional e associativo. Qualquer variável pode ser utilizada como um vetor indexado; o comando interno `declare` explicitamente declarará um vetor. Não existe limite máximo para o tamanho de um vetor, nem qualquer exigência de que os membros sejam indexados ou atribuídos contiguamente. Os vetores indexados são referenciados utilizando-se números inteiros (incluindo expressões aritméticas (veja-se Seção 6.5 [Aritmética de Shell], Página 95) e são baseados em zero; vetores associativos utilizam sequências de caracteres arbitrárias. A menos que informado do contrário, os índices de vetor indexado devem necessariamente ser números inteiros não negativos.

Um vetor indexado é criado automaticamente se qualquer variável for atribuída a ele utilizando-se a sintaxe

```
name[subscript]=value
```

O *subscript* é tratado como uma expressão aritmética que deve necessariamente resultar em um número. Para explicitamente declarar um vetor, utilize

```
declare -a name
```

A sintaxe

```
declare -a name[subscript]
```

também é aceita; o *subscript* é ignorado.

Vetores associativos são criados utilizando-se

```
declare -A name.
```

Atributos podem ser especificados para uma variável vetor utilizando-se os comandos internos `declare` e `readonly`. Cada atributo se aplica a todos os membros de um vetor.

Vetores recebem valores utilizando-se atribuições compostas da forma

```
name=(value1 value2 ... )
```

onde cada *value* é da forma `[subscript]=`*string*. Atribuições de vetor indexado não exigem nada exceto *string*. Quando da atribuição a vetores indexados, se o sub-script opcional for fornecido, então aquele índice recebe a atribuição; do contrário o índice do elemento atribuído é o último índice atribuído pela declaração mais um. A indexação inicia no zero.

Quando da atribuição a um vetor associativo, o sub-script é exigido.

Essa sintaxe também é aceita pelo comando interno `declare`. Pode-se atribuir valores aos elementos individuais de um vetor utilizando-se a sintaxe `name[subscript]=value` apresentada acima.

Quando da atribuição a um vetor indexado, se *name* for subscrita por um número negativo, então esse número é interpretado como relativo a um maior que o índice máximo de *name*, de maneira que índices negativos contam do final do vetor para o início, e um índice de -1 faz referência ao último elemento.

Qualquer elemento de um vetor pode ser referenciado utilizando-se `${name[subscript]}`. As chaves são exigidas para se evitar conflitos com os

operadores de expansão de nome de arquivo do shell. Se o *subscript* for '`@`' ou '`*`', então a palavra expande para todos os membros do vetor *name*. Esses subscritos diferem somente quando a palavra aparece dentro de aspas duplas. Se a palavra estiver encapsulada em aspas duplas, então `${name[*]}` expande para um palavra singular com o valor de cada membro do vetor separado pelo primeiro carácter da variável `IFS`, e `${name[@]}` expande cada elemento de *name* para uma palavra separada. Quando não existem membros do vetor, `${name[@]}` expande para nada. Se a expansão de aspas duplas ocorre dentro de uma palavra, então a expansão do primeiro parâmetro é anexado com a parte inicial da palavra original, e a expansão do último parâmetro é anexado com a última parte da palavra original. Isso é análogo à expansão dos parâmetros especiais '`@`' e '`*`'. `${#name[subscript]}` expande para o comprimento de `${name[subscript]}`. Se *subscript* for '`@`' ou '`*`', então a expansão é o número de elementos no vetor. O referenciamento a uma variável vetor sem um subscrito é equivalente a referenciar com um subscrito 0. Se o *subscript* utilizado para referenciar um elemento de um vetor indexado resultar em um número menor que zero, então ele é interpretado como relativo a um maior que o índice máximo do vetor, de forma que índices negativos contam do final do vetor para o seu início, e um índice de -1 se refere ao último elemento.

Uma variável vetor é considerada configurada se um subscrito tiver sido atribuído ao valor. A sequência de caracteres nula é um valor válido.

É possível se obter as chaves (índices) de um vetor bem os valores. ${!*name*[@]} e ${!*name*[*]} expandem para os índices atribuídos na variável de vetor *name*. O tratamento quando em aspas duplas é semelhante ao da expansão dos parâmetros especiais '`@`' e '`*`' dentro de aspas duplas.

O comando interno `unset` é utilizado para destruir vetores. `unset name[subscript]` destrói o elemento do vetor no índice *subscript*. Subscritos negativos para vetores indexados são interpretados conforme descrito acima. Deve-se necessariamente tomar cuidado para se evitar efeitos colaterais indesejados causados pela expansão de nome de arquivo. `unset name`, onde *name* é um vetor, remove o vetor inteiro. Um subscrito de '`*`' ou '`@`' também remove o vetor inteiro.

Os comandos internos `declare`, `local`, e `readonly` cada aceitam uma opção `-a` para especificar um vetor indexado e uma opção `-A` para especificar um vetor associativo. Se ambas as opções forem fornecidas, então `-A` tem a precedência. O comando interno `read` aceita uma opção `-a` para se atribuir uma lista de palavras lida a partir da entrada padrão para um vetor, e pode ler valores a partir da entrada padrão para elementos individuais do vetor. Os comandos internos `set` e `declare` exibem valores do vetor de uma forma que permite que se possa reutilizá-los como entrada.

6.8 A Pilha de Diretório

A pilha de diretório é uma lista de diretórios recentemente visitados. O comando interno `pushd` adiciona diretório à pilha conforme ele muda o diretório atual, e o comando interno `popd` remove os diretórios especificados da pilha e muda o diretório atual para o diretório removido. O comando interno `dirs` exibe o conteúdo da pilha de diretório.

O conteúdo da pilha de diretório também é visível como o valor da variável de shell `DIRSTACK`.

6.8.1 Comandos Internos da Pilha de Diretório

`dirs`

```
dirs [-clpv] [+N | -N]
```

Exibe a lista dos diretórios atualmente lembrados. Os diretórios são adicionados à lista com o comando `pushd`; o comando `popd` remove diretórios da lista.

`-c` Limpa a pilha de diretório deletando todos os elementos.

`-l` Produz uma listagem utilizando nomes de caminho completos; o formato padrão de listagem utiliza um til para denotar o diretório home.

`-p` Faz com `dirs` imprima a pilha de diretório com uma entrada por linha.

`-v` Faz com que `dirs` imprima a pilha de diretório com uma entrada por linha, prefixando cada entrada com seu índice na pilha.

`+`*N* Exibe o *N*ésimo diretório (contando-se a partir da esquerda da lista impressa por `dirs` quando invocado sem opções), iniciando com zero.

`-`*N* Exibe o *N*ésimo diretório (contando-se a partir da direita da lista impressa por `dirs` quando invocado sem opções), iniciando com zero.

`popd`

```
popd [-n] [+N | -N]
```

Remove a entrada topo da pilha de diretório, e muda (`cd`) para o novo diretório topo. Quando nenhum argumento é dado, `popd` remove o diretório topo da pilha e realiza um `cd` para o novo diretório topo. Os elementos são numerados a partir do 0 iniciando no primeiro diretório listado com `dirs`; isto é, `popd` é equivalente a `popd +0`.

`-n` Suprime a mudança normal de diretório quando da remoção de diretórios da pilha, de forma que somente a pilha é manipulada.

`+`*N* Remove o *N*ésimo diretório (contando-se a partir da esquerda da lista impressa por `dirs`), iniciando com zero.

`-`*N* Remove o *N*ésimo diretório (contando-se a partir da direita da lista impressa por `dirs`), iniciando com zero.

`pushd`

```
pushd [-n] [+N | -N | dir]
```

Salva o diretório atual o topo da pilha de diretório e então muda (`cd`) para *dir*. Sem argumentos, `pushd` permuta os dois diretórios topo.

`-n` Suprime a mudança normal de diretório quando da adição de diretórios à pilha, de forma que somente a pilha é manipulada.

`+N` Traz o *N*ésimo diretório (contando-se a partir da esquerda da lista impressa por `dirs`, iniciando com zero) para o topo da lista rotacionando a pilha.

`-N` Traz o *N*ésimo diretório (contando-se a partir da esquerda da lista impressa por `dirs`, iniciando com zero) para o topo da lista rotacionando a pilha.

`dir` Faz com que o diretório de trabalho atual seja o topo da pilha, tornando-o o novo diretório atual, como se tal diretório tivesse sido fornecido como um argumento para o comando interno `cd`.

6.9 Controlando o Prompt

O valor da variável `PROMPT_COMMAND` é examinado um pouco antes que Bash imprima cada prompt primário. Se `PROMPT_COMMAND` estiver configurada e tiver um valor não nulo, então o valor é executado exatamente como se tal valor tivesse sido digitado na linha de comando.

Adicionalmente, a tabela seguinte descreve os caracteres especiais os quais podem aparecer nas variáveis de prompt `PS1` até `PS4`:

`\a` Um carácter de sino.

`\d` A data, no formato "Dia da Semana Mês Dia" (por exemplo, "Terça Maio 26").

`\D{`*`format`*`}`
O *format* é passado para `strftime`(3) e o resultado é inserido na sequência de caracteres de prompt; um *format* vazio resulta em uma representação de hora específica do local. As chaves são exigidas.

`\e` Um carácter de escape.

`\h` O nome da máquina, até o primeiro '.'.

`\H` O nome da máquina.

`\j` O número de tarefas atualmente gerenciadas pelo shell.

`\l` O nome de base do nome de dispositivo de terminal do shell.

`\n` Uma linha nova.

`\r` Um retorno de carro.

`\s` O nome do shell, o nome de base de `$0` (a porção seguinte à barra final).

`\t` A hora, no formato 24 horas HH:MM:SS

`\T` A hora, no formato 12 horas HH:MM:SS

`\@` A hora, no formato 12 horas am/pm

`\A` A hora, no formato 24 horas HH:MM

`\u` O nome de usuário do usuário atual.

`\v` A versão de Bash (por exemplo, 2.00)

`\V` A versão de lançamento de Bash, versão `+` nível de correção (por exemplo, 2.00.0)

`\w` O diretório de trabalho atual, com `$HOME` abreviada com um til (utiliza a variável `$PROMPT_DIRTRIM`).

`\W` O nome de base de `$PWD`, com `$HOME` abreviada com um til.

`\!` O número de histórico desse comando.

`\#` O número de comando desse comando.

`\$` Se o uid efetivo for 0, `#`, do contrário `$`.

`\`*nnn* O carácter cujo código ASCII for o valor octal *nnn*.

`\\` Uma barra invertida.

`\[` Inicia uma sequência de caracteres não imprimíveis. Isso poderia ser utilizado para embutir uma sequência de controle de terminal no prompt.

`\]` Finaliza uma sequência de caracteres não imprimíveis.

O número de comando e o número de histórico são usualmente diferentes: o número de histórico de um comando é a posição dele na lista de histórico, a qual pode incluir comandos restaurados a partir do arquivo de histórico (veja-se Seção 9.1 [Facilidades do Histórico de Bash], Página 145), enquanto que o número de comando é a posição na sequência de comandos executada durante a atual sessão do shell.

Após a string ser decodificada, ela é expandida via expansão de parâmetro, substituição de comando, expansão aritmética, e remoção de aspas, sujeito ao valor da opção de shell `promptvars` (veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52).

6.10 O Shell Restrito

Se Bash for inicializado com o nome `rbash`, ou a opção `--restricted` ou `-r` for fornecida na invocação, então o shell se torna restrito. Um shell restrito é utilizado para configurar um ambiente mais controlado que o shell padrão. Um shell restrito se comporta identicamente a `bash` com exceção que o seguinte é proibido ou não realizado:

- Mudança de diretórios com o comando interno `cd`.
- Configurar ou desconfigurar os valor das variáveis `SHELL`, `PATH`, `ENV`, ou `BASH_ENV`.
- Especificação de nomes de comandos contendo barras.
- Especificação de um nome de arquivo contendo uma barra como um argumento para o comando interno `.`
- Especificação de um nome de arquivo contendo uma barra como um argumento para a opção `-p` para o comando interno `hash`.
- Importação de definições de função a partir do ambiente de shell na inicialização.
- Análise do valor de `SHELLOPTS` a partir do ambiente de shell na inicialização.
- Redirecionamento de saída utilizando-se os operadores de redireção '`>`', '`>|`', '`<>`', '`>&`', '`&>`', e '`>>`'.
- Utilização do comando interno `exec` para substituir o shell com um outro comando.
- Adicionar ou deletar comandos internos com as opções `-f` e `-d` para o comando interno `enable`.

- Utilizar o comando interno `enable` para habilitar comandos internos de shell desabilitados.
- Especificar a opção `-p` para o comando interno `command`.
- Desligar o modo restrito com '`set +r`' ou '`set +o restricted`'.

Essas restrições são aplicadas após quaisquer arquivos de inicialização serem lidos.

Quando um comando que for encontrado como sendo um script de shell é executado (veja-se Seção 3.8 [Scripts de Shell], Página 42), `rbash` desliga quaisquer restrições no shell gerado para executar o script.

6.11 O Modo POSIX de Bash

Inicializar Bash com a opção de linha de comando `--posix`, ou executar '`set -o posix`' enquanto Bash estiver em execução, fará com que Bash se conforme mais proximamente ao padrão POSIX, modificando o comportamento para coincidir com aquele especificado por POSIX em áreas onde o padrão Bash difere.

Quando invocado como `sh`, Bash entra no modo POSIX após a leitura dos arquivos de inicialização.

A seguinte lista é aquilo que é modificado quando 'modo POSIX' está em efeito:

1. Quando um comando na tabela hash não existe mais, Bash re-pesquisará `$PATH` para encontrar a localização nova. Isso também está disponível com '`shopt -s checkhash`'.
2. A mensagem impressa pelo código de controle de tarefa e comandos internos quando uma tarefa sai com um código diferente de zero é 'Done(status)'.
3. A mensagem impressa pelo código de controle de tarefa e comandos internos quando uma tarefa é parada é 'Stopped(*signame*)', onde *signame* é, por exemplo, `SIGTSTP`.
4. O comando interno `bg` utiliza o formato exigido para descrever cada tarefa colocada em segundo plano, o qual não inclui uma indicação de quando a tarefa é a tarefa atual ou prévia.
5. As palavras reservadas que aparecerem em um contexto onde palavras reservadas são reconhecidas não se sujeitam a expansão de apelidos.
6. As expansões POSIX `PS1` e `PS2` de '`!`' para o número de histórico e '`!!`' para '`!`' são habilitadas, e a expansão de parâmetro é realizada sobre os valores de `PS1` e `PS2`, independentemente da configuração da opção `promptvars`.
7. Os arquivos de inicialização POSIX são executados (`$ENV`) ao invés dos arquivos normais de Bash.
8. A expansão de til somente é realizada sobre atribuições precedendo um nome de comando, ao invés de sobre todas as declarações de atribuição na linha.
9. O comando interno `command` não previne comandos internos, que tomem declarações de atribuição como argumentos, da expansão deles como declarações de atribuição; quando não em modo POSIX, comandos internos de atribuição perdem as suas propriedades de expansão de declaração de atribuição, quando precedidos por `command`.
10. O arquivo de histórico padrão é `~/.sh_history` (isso é o valor padrão de `$HISTFILE`).
11. A saída de '`kill -l`' imprime todos os nomes de sinal em uma linha única, separados por espaços, sem o prefixo '`SIG`'.

12. O comando interno `kill` não aceita nomes de sinal com o prefixo '`SIG`'.
13. Os shell não interativos saem se *filename* em `.` *filename* não seja encontrado.
14. Os shells não interativos saem se um erro de sintaxe em uma expansão aritmética resultar em uma expressão inválida.
15. Os shells não interativos saem se existir um erro de sintaxe em um script lido com os comandos internos `.` ou `source`, ou em uma sequência de caracteres processada pelo comando interno `eval`.
16. Os operadores de redireção não realizam expansão de nome de arquivo sobre a palavra na redireção a menos que o shell seja interativo.
17. Os operadores de redireção não realizam partição de palavra sobre a palavra na redireção.
18. Os nomes de função devem necessariamente ser `name`s válidos de shell. Isto é, eles não podem conter outros caracteres que não letras, dígitos, e sublinhados, e não podem iniciar com um dígito. Declarar uma função com um nome inválido causa um erro fatal de sintaxe em shells não interativos.
19. Os nomes de função não podem ser os mesmos que os de algum dos comandos internos POSIX especiais.
20. Os comandos internos POSIX especiais são encontrados antes das funções de shell durante a procura de comando.
21. A palavra reservada `time` pode ser utilizada por ela mesma como um comando. Quando utilizada dessa maneira, ela exibe estatísticas de temporização para o shell e seus filhos completados. A variável `TIMEFORMAT` controla o formato da informação de temporização.
22. Quando da análise e expansão de uma expansão ${...} que aparece dentro de aspas duplas, aspas simples não mais são especiais e não podem ser utilizadas para encapsular uma chave em fechamento ou outro carácter especial, a menos que o operador seja um daqueles definidos para realizar a remoção de padrão. Nesse caso, as aspas simples não tem de aparecer como pares coincididos.
23. O analisador não reconhece `time` como uma palavra reservada se o próximo token inicia com um '`-`'.
24. Se um comando interno POSIX especial retorna um código de erro, um shell não interativo sai. Os erros fatais são aqueles listados no padrão POSIX, e incluem coisas como a análise de opções incorretas, erros de redireção, erros de atribuição a variável para atribuições que precedem o nome de comando, e assim por diante.
25. Um shell não interativo sai com um código de erro se um erro de atribuição a variável ocorrer quando nenhum nome de comando seguir as declarações de atribuição. Um erro de atribuição a variável ocorre, por exemplo, quando da tentativa de atribuir um valor para uma variável somente leitura.
26. Um shell não interativo sai com um código de erro se um erro de atribuição a variável ocorrer em uma declaração de atribuição precedendo um comando interno especial, porém não com qualquer outro comando simples.
27. Um shell não interativo sai com um código de erro se a variável de iteração em uma declaração `for` ou a variável de seleção em uma declaração `select` for uma variável somente leitura.

28. A substituição de processo não está disponível.
29. Enquanto a indireção de variável estiver disponível, ela não pode ser aplicada aos parâmetros especiais '`#`' e '`?`'.
30. As declarações de atribuição precedendo comandos internos POSIX especiais persistem no ambiente de shell após o comando interno completar.
31. As declarações de atribuição precedendo as chamadas de função de shell persistem no ambiente de shell após a função retornar, como se um comando interno POSIX especial tivesse sido executado.
32. Os comandos internos `export` e `readonly` exibem suas saídas no formato exigido por POSIX.
33. O comando interno `trap` exibe nomes de sinal sem o prefixo `SIG`.
34. O comando interno `trap` não verifica o primeiro argumento para uma possível especificação de sinal e reverte o manipulador de sinal para a disposição original se estiver, a menos que argumento consista unicamente de dígitos e for um número válido de sinal. Se usuários desejarem reconfigurar o manipulador para um dado sinal para a disposição original, então eles deveriam utilizar '`-`' como o primeiro argumento.
35. Os comandos internos `.` e `source` não pesquisam o diretório atual em busca de argumento de nome de arquivo se o argumento não for encontrado pesquisando-se `PATH`.
36. Os sub-shells criados para executar substituições de comando herdam o valor da opção `-e` do shell pai. Quando não no modo POSIX, Bash limpa a opção `-e` em tais sub-shells.
37. A expansão de apelido sempre está habilitada, mesmo em shells não interativos.
38. Quando o comando interno `alias` exibe definições de apelidos, ele não os exibe com um '`alias` ' no início, a menos que a opção `-p` seja fornecida.
39. Quando o comando interno `set` for invocado sem opções, ele não exibe definições e nomes de função de shell.
40. Quando o comando interno `set` for invocado sem opções, ele exibe valores de variáveis sem aspas, a menos que eles contenham meta-caracteres de shell, mesmo se o resultado contém caracteres não imprimíveis.
41. Quando o comando interno `cd` for invocado no modo *logical*, e o nome de caminho construído a partir de `$PWD`e o nome de diretório fornecido como um argumento não se refere a um diretório existente, `cd` falhará, ao invés de retornar ao modo *physical*.
42. O comando interno `pwd` verifica se o valor que ele imprime é o mesmo que o diretório atual, mesmo se não for pedido a ele para checar o sistema de arquivo com a opção `-P`.
43. Quando da listagem do histórico, o comando interno `fc` não inclui uma indicação de quando ou não uma entrada de histórico foi modificada.
44. O editor padrão utilizado por `fc` é `ed`.
45. Os comandos internos `type` e `command` não reportarão um arquivo não executável como tendo sido encontrado, apesar que o shell tentará executar tal arquivo se ele for o único arquivo assim nominado encontrado no `$PATH`.
46. O modo de edição `vi` invocará o editor `vi` diretamente quando o comando '`v`' for executado, ao invés de verificar `$VISUAL` e `$EDITOR`.
47. Quando a opção `xpg_echo` estiver habilitada, Bash não tenta interpretar quaisquer argumentos para `echo` como opções. Cada argumento é exibido, após os caracteres de escape serem convertidos.

48. O comando interno `ulimit` utiliza um tamanho de bloco de 512 bytes para as opções `-c` e `-f`.
49. A chegada de `SIGCHLD` quando um coletor estiver configurado sobre `SIGCHLD` não interrompe o comando interno `wait` e faz com que ele retorne imediatamente. O comando coletor é executado uma vez para cada filho que sai.
50. O comando interno `read` pode ser interrompido por um sinal para o qual um coletor tenha sido configurado. Se Bash recebe um sinal coletado enquanto executando `read`, então o manipulador de coletor executa e `read`, retorna um código de saída maior que 128.

Existe outro comportamento POSIX que Bash não implementa por padrão mesmo quando no modo POSIX. Especificamente:

1. O comando interno `fc` verifica `$EDITOR` como um programa para editar entradas de histórico se `FCEDIT` estiver desconfigurada, no lugar de diretamente considerar como padrão `ed`. `fc` utiliza `ed` se `EDITOR` estiver desconfigurada.
2. Conforme apontado acima, Bash exige que a opção `xpg_echo` esteja habilitada para que o comando interno `echo` seja completamente conformante.

Bash pode ser configurado para ser conformante com POSIX por padrão, especificando-se `--enable-strict-posix-default` para `configure` quando da construção (veja-se Seção 10.8 [Características Opcionais], Página 154).

7 Controle de Tarefa

Este capítulo discute o que é o controle de tarefas, como ele funciona, e como Bash te permite acessar as facilidades desse controle.

7.1 Fundamentos do Controle de Tarefa

O controle de tarefas se refere à habilidade de parar (suspender) seletivamente a execução de processos e continuar (resumir) a execução deles num ponto mais tarde. Um usuário tipicamente emprega essa facilidade via uma interface interativa fornecida conjuntamente pelo driver de terminal do núcleo do sistema operacional e por Bash.

O shell associa um *job* com cada canal de comunicação. O shell mantém uma tabela das tarefas atualmente em execução, as quais podem ser listadas com o comando `jobs`. Quando Bash inicia uma tarefa assincronamente, ele imprime uma linha que se parece com isto:

```
[1] 25647
```

indicando que essa tarefa é a tarefa número 1 e que o ID de processo do último processo no canal de comunicação associado com essa tarefa é 25647. Bash utiliza a abstração de *job* como uma base para o controle de tarefa.

Para facilitar a implementação da interface de usuário para o controle de tarefa, o sistema operacional mantém a noção de ID de grupo de processo de terminal atual. Os membros desse grupo de processos (processos cujo ID de grupo de processo seja igual ao atual ID de grupo de processo de terminal) recebem sinais gerados por teclado tais como `SIGINT`. Diz-se que esses processos estão no primeiro plano. Os processos de segundo plano são aqueles cujo ID de grupo de processo difere do ID de grupo de processo do terminal; tais processos estão imunes aos sinais gerados por teclado. Apenas aos processos de primeiro plano é permitido ler a partir de ou, se o usuário especifica com `stty tostop`, escrever para o terminal. Aos processos de segundo plano que tentam ler a partir (escrever para quando `stty tostop` está em efeito) do terminal é enviado um sinal `SIGTTIN` (`SIGTTOU`) pelo driver de terminal do kernel, o qual, a menos que capturado, suspende o processo.

Se o sistema operacional no qual Bash está em execução suporta controle de tarefa, então Bash contém facilidades para utilizá-lo. Digitar-se o carácter *suspend* (tipicamente '`^Z`', Control-Z) enquanto um processo está em execução faz com que esse processo seja parado e retorne o controle para Bash. O usuário então manipula o estado dessa tarefa, utilizando o comando `bg` para continuá-lo no primeiro plano, ou o comando `kill` para matá-lo. Um '`^Z`' tem efeito imediatamente, e tem o efeito colateral adicional de fazer com que saída pendente e autocompletar serem descartados.

Existe um número de maneiras de se referir a uma tarefa no shell. O carácter '`%`' introduz uma especificação de tarefa (*jobspec*).

Um número de tarefa `n` pode ser referenciado como '`%n`'. Os símbolos '`%%`' e '`%+`' se referem à noção do shell da tarefa atual, a qual é a última tarefa parada enquanto ela estava no primeiro plano ou inicializada no segundo plano.

Um único '`%`' (sem acompanhamento da especificação de tarefa) também se refere à tarefa atual. A tarefa previa pode ser referenciada utilizando-se '`%-`'. Se existir apenas uma tarefa única, então '`%+`' e '`%-`' podem ambos serem utilizados para se referir àquela tarefa. Na saída relativa à tarefas (por exemplo, a saída do comando `jobs`), a tarefa atual sempre é rotulada com um '`+`', e a tarefa prévia com um '`-`'.

Uma tarefa também pode ser referenciada utilizando-se um prefixo do nome utilizado para inicializá-la, ou utilizando-se uma parte da sequência de caracteres que aparece na sua linha de comando. Por exemplo, '`%ce`' se refere à tarefa `ce` parada. Utilizando-se '`%?ce`', por outro lado, se refere a qualquer tarefa contendo a sequência de caracteres '`ce`' em sua linha de comando. Se o prefixo ou parte da sequencia de caracteres coincide com mais que uma tarefa, então Bash reporta um erro.

Em poucas palavras, uma tarefa pode ser utilizada para se trazer para o primeiro plano: '`%1`' é um sinônimo para '`fg %1`', trazendo a tarefa 1 do segundo plano para o primeiro plano. Similarmente, '`%1 &`' suspende a tarefa 1 no primeiro plano, equivalente a '`bg %1`'.

O shell aprende imediatamente quando uma tarefa muda de estado. Normalmente, Bash aguarda até que esteja prestes a imprimir um prompt antes de reportar modificações em um estado da tarefa, de forma a não interromper qualquer outra entrada. Se a opção `-b` para o comando interno `set` estiver habilitada, então reporta tais modificações imediatamente (veja-se Seção 4.3.1 [O Comando Interno Set], Página 63). Qualquer coletor sobre `SIGCHLD` é executado para cada processo filho que sai.

Se uma tentativa de sair de Bash for feita enquanto tarefas estão paradas, (ou em execução, se a opção `checkjobs` estiver habilitada – veja-se Seção 4.3.2 [O Comando Interno Shopt], Página 68), então o shell imprime uma mensagem de alerta, e se a opção `checkjobs` estiver habilitada, lista as tarefas e suas situações.

O comando `jobs` pode então ser utilizado para inspecionar os estados das tarefas. Se uma segunda tentativa de sair for feita, sem um comando interveniente, então Bash não imprime outro alerta, e quaisquer tarefas paradas são terminadas.

7.2 Comandos Internos do Controle de Tarefa

`bg`

```
bg [jobspec ...]
```

Continua a execução de cada tarefa *jobspec* suspensa no segundo plano, como se ela tivesse sido inicializada com '`&`'. Se *jobspec* não for fornecida, então a tarefa atual é utilizada. O código de retorno é zero, a menos que esse comando interno seja executado quando o controle de tarefa não esteja habilitado, ou, quando executado com o controle de tarefa habilitado, qualquer *jobspec* não foi encontrado ou especifica uma tarefa que foi inicializada sem controle de tarefa.

`fg`

```
fg [jobspec]
```

Continua a execução da tarefa *jobspec* no primeiro plano e a faz a tarefa atual. Se *jobspec* não for fornecida, então a tarefa atual é utilizada. O código de retorno é aquele do comando colocado no primeiro plano, ou diferente de zero se executado quando o controle de tarefa estiver desabilitado ou, quando executado com o controle de tarefa habilitado, *jobspec* não especificar uma tarefa válida ou *jobspec* especifica uma tarefa que foi inicializada sem controle de tarefa.

`jobs`

```
jobs [-lnprs] [jobspec]
jobs -x command [arguments]
```

A primeira forma lista as tarefas ativas. As opções tem os seguintes significados:

`-l` Lista IDs de processos em adição à informação normal.

`-n` Exibe informação somente sobre tarefas que tiveram estado modificado desde quando o usuário foi notificado mais recentemente do estado delas.

`-p` Lista somente o ID de processo do líder de grupo de processo da tarefa.

`-r` Exibe apenas tarefas em execução.

`-s` Exibe apenas tarefas paradas.

Se *jobspec* for dada, então a saída é restrita à informação sobre aquela tarefa. Se *jobspec* não for fornecida, então a situação de todas as tarefas é listada.

Se a opção `-x` for fornecida, então `jobs` substitui qualquer *jobspec* encontrado em *command* ou *arguments* com o correspondente ID de grupo de processo, e executa *command*, passando a ele *argument*s, retornando seu código de saída.

`kill`

```
kill [-s sigspec] [-n signum] [-sigspec] jobspec or pid
kill -l [exit_status]
```

Envia um sinal especificado por *sigspec* ou *signum* para o processo nomeado pela especificação de tarefa *jobspec* ou ID de processo *pid*. *sigspec* é ou um nome de sinal sem distinção entre maiúsculas e minúsculas, tal como `SIGINT` (com ou sem o prefixo `SIG`) ou um número de sinal; *signum* é um número de sinal. Se *sigspec* e *signum* não estiverem presentes, então `SIGTERM` é utilizado. A opção `-l` lista os nomes de sinal. Se quaisquer argumentos forem fornecidos quando `-l` for dada, então os nomes dos sinais correspondentes aos argumentos são listados, e o código de retorno é zero. *exit_status* é um número que especifica um número de sinal ou a situação de saída de um processo terminado por um sinal. O código de retorno é zero se pelo menos um sinal foi enviado com sucesso, ou diferente de zero se um erro ocorre ou uma opção inválida é encontrada.

`wait`

```
wait [-n] [jobspec or pid ...]
```

Aguarda até que o processo filho especificado por cada ID de processo *pid* ou especificação de tarefa *jobspec* saia e retorna a situação de saída do último comando aguardado. Se uma especificação de tarefa for dada, então todos os processos na tarefa são aguardados. Se nenhum argumento for dado, então todos os processos filho atualmente ativos são aguardados, e o código de retorno é zero. Se a opção `-n` for fornecida, então `wait` aguarda qualquer tarefa terminar e retorna sua situação de saída. Se nem *jobspec* nem *pid* especifica um processo filho ativo do shell, então o código de retorno é 127.

`disown`

```
disown [-ar] [-h] [jobspec ...]
```

Sem opções, remove cada *jobspec* da tabela de tarefas ativas. Se a opção `-h` for dada, então a tarefa não é removida da tabela, porém é marcada, de forma que

`SIGHUP` não é enviado para a tarefa se o shell recebe um `SIGHUP`. Se *jobspec* não estiver presente, e nem a opção `-a` nem a `-r` forem fornecidas, então a tarefa atual é utilizada. Se nenhuma *jobspec* for fornecida, então a opção `-a` significa remover ou marcar todas as tarefas; a opção `-r` sem um argumento *jobspec* restringe a operação a tarefas em execução.

`suspend`

```
suspend [-f]
```

Para a execução deste shell até que ele receba um sinal `SIGCONT`. Um shell de login não pode ser parado; a opção `-f` pode ser utilizada para anular isso e forçar a suspensão.

Quando o controle de tarefa não está ativo, os comandos internos `kill` e `wait` não aceitam argumentos *jobspec*. Eles devem necessariamente ser alimentados com IDs de processos.

7.3 Variáveis do Controle de Tarefa

`auto_resume`

Essa variável controla como o shell interage com o usuário e o controle de tarefa. Se essa variável existir, então os comandos simples de palavra única sem redireções são tratados como candidatos para o reinício de uma tarefa existente. Não existe ambiguidade que seja permitida; se existe mais que uma tarefa iniciando com a sequência de caracteres digitada, então a mais recentemente acessada tarefa será selecionada. O nome de uma tarefa parada, nesse contexto, é a linha de comando utilizada para iniciá-la. Se essa variável for configurada para o valor '`exact`', então a sequência de caracteres fornecida deve necessariamente coincidir exatamente com o nome de uma tarefa parada; se configurada para '`substring`', então a sequência de caracteres fornecida precisa coincidir com uma parte da sequência de caracteres do nome de uma tarefa parada. O valor de '`substring`', provê funcionalidade análoga para o ID de tarefa '`%?`' (veja-se Seção 7.1 [Fundamentos do Controle de Tarefa], Página 106). Se configurada para qualquer outro valor, então a sequência de caracteres fornecida deve necessariamente ser um prefixo de um nome da tarefa parada; isso provê funcionalidade análoga para o ID de tarefa '`%`'.

8 Edição de Linha de Comando

Este capítulo descreve as características básicas da interface de edição de linha de comando GNU. A edição de linha de comando é provida pela biblioteca Readline, a qual é utilizada por vários programas diferentes, incluindo Bash. A edição de linha de comando está habilitada por padrão quando da utilização de um shell interativo, a menos que a opção `--noediting` seja fornecida na invocação de shell. A edição de linha também é utilizada quando da utilização da opção `-e` para o comando interno `read` (veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52). Por padrão, os comandos de edição de linha são similares àqueles de Emacs. Uma interface de edição de linha ao estilo do vi também está disponível. A edição de linha pode ser habilitada a qualquer tempo utilizando-se as opções `-o emacs` ou `-o vi` para o comando interno `set` (veja-se Seção 4.3.1 [O Comando Interno Set], Página 63), ou desabilitadas utilizando-se as opções `+o emacs` ou `+o vi` para `set`.

8.1 Introdução à Edição de Linha

Os parágrafos seguintes descrevem a notação utilizada para representar toques de tecla.

O texto *`C-k`* é lido como 'Control-K' e descreve o carácter produzido quando a tecla `k` é teclada enquanto a tecla Control está mantida pressionada.

O texto *`M-k`* é lido como 'Meta-K' e descreve o carácter produzido quando a tecla Meta (se você tiver uma) é mantida pressionada, e a tecla `k` é teclada. A tecla Meta é rotulada como `ALT` em muitos teclados. Em teclados com duas tecladas rotuladas como `ALT` (usualmente em ambos os lados da barra de espaços), o `ALT` no lado esquerdo é geralmente configurado para funcionar como uma tecla Meta. A tecla `ALT` na direita também pode ser configurada para funcionar como uma tecla Meta ou pode ser configurada como algum outro modificador, tal como uma tecla Compose para a digitação de caracteres acentuados.

Se você não tiver uma tecla Meta ou `ALT`, ou outra tecla funcionando como uma tecla Meta, o pressionamento de tecla idêntico pode ser gerado digitando-se `ESC` *primeiro*, e então digitando `k`. Qualquer dos processos é conhecido como *metafying* a tecla `k`.

O texto *`M-C-k`* é lido como 'Meta-Control-k' e descreve o carácter produzido por *metafying* *`C-k`*.

Adicionalmente, várias teclas tem seus próprios nomes. Especificamente, `DEL`, `ESC`, `LFD`, `SPC`, `RET`, e `TAB` todas significam elas mesmas quando vistas neste texto, ou em um arquivo init (veja-se Seção 8.3 [Arquivo Init de Readline], Página 113). Se no seu teclado falta uma tecla `LFD`, a digitação de `C-j` produzirá o carácter desejado. A tecla `RET` pode ser rotulada como `Return` ou `Enter` em alguns teclados.

8.2 Interação com Readline

Frequentemente, durante uma sessão interativa, você digita uma linha longa de texto, apenas para se dar conta de que a primeira palavra naquela linha está escrita errada. A biblioteca Readline te dá um conjunto de comandos para manipular o texto a medida que você o digita, te permitindo simplesmente consertar seu erro, e não te forçando a redigitar a maioria da linha. Utilizando esses comandos de edição, você movimenta o cursor para o local que necessita de correção, e deleta ou insere o texto das correções. Então, que você estiver satisfeito com a linha, você simplesmente pressiona `RET`. Você não precisa estar ao final

da linha para pressionar RET.; a linha inteira é aceita independentemente da localização do cursor dentro da linha.

8.2.1 Mínimo Essencial sobre Readline

Para colocar caracteres em uma linha, simplesmente digite-os. O carácter digitado aparece onde o cursor estava, e então o cursor se movimenta um espaço para a direita. Se você errar a digitação de um carácter, você pode utilizar o seu carácter de apagamento para voltar e deletar o carácter digitado errado.

Algumas vezes você pode digitar um carácter errado, e não se dar conta do erro até que você tenha digitado vários outros caracteres. Nesse caso, você pode digitar *C-b* para movimentar o cursor para a esquerda, e então corrigir seu erro. Posteriormente, você pode mover o cursor para a direita com *C-f*.

Quando você adiciona texto no meio de uma linha, você notará que os caracteres do lado direito do cursor são "empurrados" para dar espaço para o texto que você acabou de inserir. Da mesma maneira, quando você deleta texto sob o cursor, os caracteres do lado direito do cursor são "puxados" para preencher o espaço em braco criado pela remoção do texto. Segue uma lista do essencial para a edição do texto de uma linha de entrada.

C-b Movimenta um carácter para trás.

C-f Movimenta um carácter para frente.

DEL ou Backspace
: Apaga o carácter a esquerda do cursor.

C-d Apaga o carácter embaixo do cursor.

Printing characters
: Insere o carácter na linha na posição do cursor.

C-_ ou *C-x C-u*
: Desfaz o comando de edição mais recente. Você pode desfazer todo o comando até uma linha vazia.

(Dependendo da sua configuração, a tecla Backspace está configurada para apagar o carácter a esquerda do cursor e a tecla DEL configurada para apagar o carácter embaixo do cursor, como *C-d*, em vez do carácter a esquerda do cursor).

8.2.2 Comandos de Movimento em Readline

A tabela acima descreve os toques mais básicos que você precisa com o objetivo de editar a linha de entrada. Para a sua conveniência, muitos outros comandos foram adicionados juntamente a *C-b*, *C-f*, *C-d*, e DEL. Aqui estão alguns comandos para movimentar-se mais rapidamente ao longo da linha.

C-a Movimenta para o início da linha.

C-e Movimenta para o final da linha.

M-f Movimenta uma palavra para frente, onde uma palavra é composta de letras e dígitos.

M-b Movimenta uma palavra para trás.

`C-l` Limpa a tela, reimprimindo a linha atual no topo.

Perceba como `C-f` movimenta o cursor um carácter para frente, enquanto que `M-f` movimenta o cursor uma palavra para frente. É uma convenção não escrita que o pressionamento de teclas de controle opera sobre caracteres enquanto que o pressionamento de tecla meta opera sobre palavras.

8.2.3 Comandos Readline para Killing ("Recortar")

Killing texto significa retirar o texto da linha, porém salvá-lo fora para uso posterior, usualmente com *yanking* (reinserir) o texto na linha. ("Recortar" e "colar" são jargão mais recente para 'kill' e 'yank').

Se a descrição para um comando diz que ele 'kills' texto, então você pode ter certeza que você pode ter o texto de volta em um local diferente (ou o mesmo) depois.

Quando você utiliza um comando kill, o texto é salvo em um *kill-ring*. Qualquer número de kills consecutivos salva todo o texto killed junto, de forma que quando você yank o texto de volta, você obtém o texto todo. O "kill ring" não é específico de linha; o texto que você "matou" em uma linha digitada previamente está disponível para ser "arrancado" de volta mais tarde, quando você estiver digitando outra linha.

Aqui está a lista de comandos para recortar "killing" texto.

`C-k` Recorta o texto a partir da posição atual do cursor até o final da linha.

`M-d` Recorta a partir do cursor até o final da palavra atual, ou, se entre palavras, até o final da próxima palavra. Os limites de palavra são os mesmos que aqueles utilizados por `M-f`.

`M-DEL` Recorta a partir do cursor o início da palavra atual, ou, se entre palavras, até o início da palavra anterior. Os limites de palavra são os mesmos que aqueles utilizados por `M-b`.

`C-w` Recorta a partir do cursor até o espaço em braco anterior. Isso é diferente de `M-DEL`, pois os limites de palavra diferem.

Aqui está como "arrancar" (*yank*) o texto de volta para a linha. Yanking significa copiar o texto mais recentemente "killed" a partir da área intermediária de memória "kill".

`C-y` "Yank" o texto mais recentemente "killed" de volta na área intermediária de memória no cursor.

`M-y` Rotaciona o "kill-ring", e "yank" o novo topo. Você somente pode fazer isso se o comando anterior for `C-y` ou `M-y`.

8.2.4 Argumentos em Readline

Você pode passar argumentos numéricos aos comandos Readline. Algumas vezes o argumento atua como um contador de repetição, outras é o *sinal* do argumento que é significante. Se você passar um argumento negativo para um comando o qual normalmente atua na direção para frente, então esse comando atuará na direção para trás. Por exemplo, para "kill" texto de volta para o começo da linha, você pode digitar '`M-- C-k`'.

A maneira geral de passar argumentos numéricos para um comando é digitar meta dígitos antes do comando. Se o primeiro "dígito" digitado for um sinal de menos ('`-`'), então o sinal

do argumento será negativo. Uma vez que você tenha digitado um meta dígito para iniciar o argumento, você pode digitar o restante dos dígitos, e então o comando. Por exemplo, para dar o argumento 10 para o comando `C-d`, você poderia digitar '`M-1 0 C-d`',, o qual deletaria os próximos dez caracteres na linha de entrada.

8.2.5 Buscando Comandos no Histórico

Readline provê comandos para a busca ao longo do histórico de comando (veja-se Seção 9.1 [Facilidades do Histórico de Bash], Página 145) por linhas contendo a sequência de caracteres especificada. Existem dois modos *incremental* e *não-incremental*.

As buscas incrementais iniciam antes que o usuário tenha finalizado a digitação da sequência de caracteres de busca. A medida que cada carácter da sequência de caracteres de busca é digitado, Readline exibe a próxima entrada a partir do histórico que coincida com a sequência de caracteres digitada até agora. Uma busca incremental exige apenas tantos caracteres quantos necessários para encontrar a entrada de histórico desejada. Para pesquisar por uma sequência de caracteres particular para trás no histórico, digite `C-r`. Digitando-se `C-s` pesquisa-se para frente no histórico. Os caracteres presentes no valor da variável `isearch-terminators` são utilizados para terminar um pesquisa incremental. Se a essa variável não foi atribuído um valor, então os caracteres `ESC` e `C-J` terminarão uma pesquisa incremental. `C-g` abortará uma pesquisa incremental e restaurará a linha original. Quando a pesquisa é terminada, a entrada de histórico contendo a sequência de caracteres de busca se torna a linha atual.

Para encontrar outras entradas coincidentes na lista de histórico, digite `C-r` ou `C-s`, conforme apropriado. Isso pesquisará para trás ou para frente no histórico pela próxima entrada que coincida com a sequência de caracteres de pesquisa digitada até agora. Qualquer outras sequência de tecla vinculada a um comando Readline terminará a pesquisa e executará aquele comando. Por exemplo, um `RET` (enter) terminará a pesquisa e aceitará a linha, por conseguinte executando o comando a partir da lista de histórico. Um comando de movimento terminará a pesquisa, tornará a última linha encontrada a linha atual, e iniciar a edição.

Readline se lembra da última sequência de caracteres de pesquisa incremental. Se dois `C-r`s forem digitados sem quaisquer caracteres intervenientes que definam uma nova sequência de caracteres de busca, então qualquer sequência de caracteres de busca guardada é utilizada.

As buscas não incrementais leem a sequência de caracteres de pesquisa inteira antes de iniciar a busca por linhas de histórico coincidentes. A sequência de caracteres de busca pode ser digitada pelo usuário ou ser parte do conteúdo da linha atual.

8.3 Arquivo Init de Readline

Apesar que a biblioteca Readline vem com um conjunto de vínculos de tecla estilo Emacs instalado por padrão, é possível se utilizar um conjunto diferente de vínculos de tecla. Qualquer usuário pode personalizar programas que utilizem Readline colocando comandos em um arquivo *inputrc*, convencionalmente dentro de seu diretório "home". O nome desse arquivo é tomado do valor da variável de shell `INPUTRC`. Se essa variável estiver desconfigurada, então o padrão é `~/.inputrc`. Se esse arquivo não existir ou não puder ser lido, então o padrão final é `/etc/inputrc`.

Quando um programa que utiliza a biblioteca Readline inicializa, o arquivo init é lido, e as vinculações de tecla são configuradas.

Adicionalmente, o comando `C-x C-r` relê esse arquivo init, assim incorporando quaisquer modificações que você eventualmente possa ter feito nele.

8.3.1 Sintaxe do Arquivo Init de Readline

Existem somente umas poucas construções básicas permitidas no arquivo init de Readline. As linhas em branco são ignoradas. As linhas iniciando com um '`#`' são comentários. As linhas começando com um '`$`' indicam construções condicionais (veja-se Seção 8.3.2 [Construtores Condicionais Init], Página 121). Outras linhas denotam configurações de variáveis e vinculações de teclas.

Configurações de Variáveis

Você pode modificar o comportamento em tempo de execução de Readline alterando os valores de variáveis em Readline utilizando o comando `set` dentro do arquivo init. A sintaxe é simples:

```
set variable value
```

Aqui, por exemplo, está como alternar da vinculação padrão de tecla estilo Emacs para utilizar comandos de edição de linha `vi`:

```
set editing-mode vi
```

Os nomes de variáveis e valores, onde apropriado, são reconhecidos sem levar em consideração maiúsculas e minúsculas. Os nomes de variável não reconhecidos são ignorados.

As variáveis booleanas (aquelas que podem ser configuradas para ligada ou desligada) são configuradas para ligada se o valor for nulo ou vazio, *on* (sem levar em consideração maiúsculas e minúsculas), ou 1. Qualquer outro valor resulta na variável sendo configurada para desligada.

O comando `bind -V` lista os nomes e valores atuais da variável Readline. Veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52.

Uma grande parte do comportamento em tempo de execução é modificável com as seguintes variáveis.

`bell-style`
: Controla o que acontece quando Readline deseja soar o alerta sonoro do terminal. Se configurado para '`none`', então Readline nunca soa o alerta. Se configurada para '`visible`', então Readline utiliza um alerta visível se um estiver disponível. Se configurado para '`audible`' (o padrão), Readline tenta soar o alerta sonoro do terminal.

`bind-tty-special-chars`
: Se configurado para '`on`', Readline tenta vincular os caracteres de controle tratados especialmente pelo controlador de terminal do kernel para o equivalente Readline deles.

`colored-stats`
: Se configurado para '`on`', Readline exibe as completações possíveis utilizando cores diferentes para indicar o tipo de arquivo delas. As

definições de cor são tomadas do valor da variável de ambiente `LS_COLORS`. O padrão é '`off`'.

`comment-begin`
: A sequência de caracteres a inserir no inicio da linha quando o comando `insert-comment` for executado. O valor padrão é `"#"`.

`completion-display-width`
: O número de colunas de tela utilizado para exibir possíveis coincidências quando da realização de completação. O valor é ignorado se for menor que 0 ou maior que a largura de tela do terminal. Um valor de 0 fará com que coincidências sejam exibidas uma por linha. O valor pré-definido é -1.

`completion-ignore-case`
: Se configurado para '`on`', Readline realiza coincidência de nome de arquivo e completação em uma forma que não leva em consideração maiúsculas e minúsculas. O valor pré-definido é '`off`'.

`completion-map-case`
: Se configurado para '`on`', e *completion-ignore-case* estiver habilitada, então Readline trata hifens ('`-`') e sublinhados ('`_`') como equivalentes, quando da realização de coincidência e completação, sem levar em consideração maiúsculas e minúsculas, de nome de arquivo.

`completion-prefix-display-length`
: O comprimento em caracteres do prefixo comum de uma lista de completações possíveis que é exibida sem modificação. Quando configurado para um valor maior que zero, os prefixos comuns maiores que esse valor são substituídos com uma elipse, quando da exibição de completações possíveis.

`completion-query-items`
: O número das completações possíveis que determina quando o usuário é questionado se a lista de possibilidades deveria ser exibida. Se o número de completações possíveis for maior que esse valor, então Readline perguntará ao usuário se ou não deseja visualizá-las; do contrário, elas simplesmente são listadas. Essa variável deve necessariamente ser configurada para um número inteiro maior que ou igual a 0. Um valor negativo significa que Readline nunca deveria perguntar. O limite pré-definido é `100`.

`convert-meta`
: Se configurado para '`on`', então Readline converterá caracteres com o oitavo bit ligado para uma sequência de tecla ASCII, descartando o oitavo bit e prefixando um carácter `ESC`, convertendo-os para um sequência de tecla meta-prefixada. O valor pré-definido é '`on`'.

`disable-completion`
: Se configurado para '`On`', Readline inibirá a completação de palavra. Os caracteres de completação será inseridos na linha como se

tivessem sido mapeados para `self-insert`. O valor pré-definido é '`off`'.

`editing-mode`
: A variável `editing-mode` controla qual conjunto pré-definido de vinculações de tecla é utilizado. Por predefinição, Readline inicializa no modo de edição Emacs, onde os toques de tecla são mais similares ao Emacs. Essa variável pode ser configurada para ou '`emacs`' ou '`vi`'.

`echo-control-characters`
: Quando configurado para '`on`', em sistemas operacionais que indicam que os suportam, Readline ecoa um carácter correspondente a um sinal gerado a partir do teclado. O valor pré-definido é '`on`'.

`enable-keypad`
: Quando configurado para '`on`', Readline tentará habilitar o teclado numérico da aplicação quando ele for chamado. Alguns sistemas necessitam disso para habilitar as teclas direcionais. O valor pré-definido é '`off`'.

`enable-meta-key`
: Quando configurado para '`on`', Readline tentará habilitar qualquer tecla meta modificadora que o terminal alegue suportar, quando for chamado. Em muitos terminais, a tecla meta é utilizada para enviar caracteres de oito bits. O valor pré-definido é '`on`'.

`expand-tilde`
: Se configurado para '`on`', a expansão de til é realizada quando Readline tenta a completação de palavra. O valor pré-definido é '`off`'.

`history-preserve-point`
: Se configurado para '`on`', o código de histórico tenta colocar o ponto (a posição atual do cursor) na mesma localização em cada linha de histórico recuperada com `previous-history` ou `next-history`. O valor pré-definido é '`off`'.

`history-size`
: Configura o número máximo de entradas de histórico salvas na lista de histórico. Se configurada para zero, então quaisquer entradas de histórico existentes são deletadas e nenhuma entrada nova é salva. Se configurada para um valor menor que zero, então o número de entradas de histórico não é limitada. Por predefinição, o número de entradas de histórico não é limitada.

`horizontal-scroll-mode`
: Essa variável pode ser configurada para ou '`on`' ou '`off`'. A configuração dela para '`on`' significa que o texto das linhas sendo editadas passará na tela horizontalmente em uma única linha de tela quando forem maiores que a largura da tela, em vez de acondicioná-lo em uma nova linha de tela. Por predefinição, essa variável é configurada para '`off`'.

`input-meta`
: Se configurado para '`on`', Readline habilitará a entrada de oito bits (não limpará o oitavo bit nos caracteres que lê), independentemente do que o terminal alega que pode suportar. O valor pré-definido é '`off`'. O nome `meta-flag` é um sinônimo para essa variável.

`isearch-terminators`
: A sequência de caracteres dos caracteres que deveriam terminar uma pesquisa incremental sem executar subsequentemente o carácter como um comando (veja-se Seção 8.2.5 [Searching], Página 113). Se a essa variável não tiver sido dado um valor, então os caracteres `ESC` e *`C-J`* terminarão uma pesquisa incremental.

`keymap`
: Configura a ideia de Readline acerca do mapa de teclas atual para os comandos de vinculação de tecla. Os nomes aceitáveis de `keymap` são `emacs`, `emacs-standard`, `emacs-meta`, `emacs-ctlx`, `vi`, `vi-move`, `vi-command`, e `vi-insert`. `vi` é equivalente a `vi-command`; `emacs` é equivalente a `emacs-standard`. O valor pré-definido é `emacs`. O valor da variável `editing-mode` também afeta o mapa de teclas pré-definido.

`keyseq-timeout`
: Especifica a duração que Readline aguardará por um carácter quando da leitura de uma sequência de tecla ambígua (aquela que pode formar uma sequência de tecla completa utilizando a entrada lida até agora, ou pode tomar entrada adicional para completar uma sequência de tecla mais longa). Se nenhuma entrada é recebida dentro do intervalo de tempo, então Readline utilizará a sequência de tecla mais curta, porém completa. Readline utiliza esse valor para determinar quando ou não a entrada está disponível na fonte atual de entrada (por predefinição `rl_instream`). O valor é especificado em milissegundos, de forma que um valor de 1000 significa que Readline aguardará um segundo por entrada adicional. Se essa variável não estiver configurada para um valor menor que ou igual a zero, ou para um valor não numérico, então Readline aguardará até que outra tecla seja pressionada para decidir qual sequência de tecla completar. O valor pré-definido é `500`.

`mark-directories`
: Se configurado para '`on`', então os nomes completados de diretórios terão uma barra acrescentada. O valor pré-definido é '`on`'.

`mark-modified-lines`
: Essa variável, quando configurada para '`on`', faz com que Readline exiba um asterisco ('`*`') no inicio de linhas de histórico que tenham sido modificadas. Essa variável está '`off`' por predefinição.

`mark-symlinked-directories`
: Se configurada para '`on`', então os nomes completados que sejam vínculos simbólicos para diretórios terão uma barra acrescentada

(sujeita ao valor de `mark-directories`). O valor pré-definido é '`off`'.

`match-hidden-files`
: Essa variável, quando configurada para '`on`', faz com que Readline coincida arquivos cujos nomes iniciem com um '`.`' (arquivos ocultos) quando da realização de completação de nome de arquivo. Se configurada para '`off`', então o '`.`' inicial deve necessariamente ser fornecido pelo usuário no nome de arquivo para ser completado. Essa variável está '`on`' por predefinição.

`menu-complete-display-prefix`
: Se configurada para '`on`', então a completação de menu exibe o prefixo comum da lista de possíveis completações (as quais podem estar vazias) antes de circular ao longo da lista. O valor pré-definido é '`off`'.

`output-meta`
: Se configurada para '`on`', então Readline exibirá caracteres com o oitavo bit configurado diretamente, em vez de uma sequência de escape meta prefixada. O valor pré-definido é '`off`'.

`page-completions`
: Se configurada para '`on`', então Readline utiliza um paginador interno estilo `more` para exibir uma tela completa de possíveis completações por vez. Essa variável está '`on`' por predefinição.

`print-completions-horizontally`
: Se configurada para '`on`', então Readline exibirá as completações com as coincidências ordenadas horizontalmente em ordem alfabética, em vez de o fazer tela abaixo. O valor pré-definido é '`off`'.

`revert-all-at-newline`
: Se configurada para '`on`', então Readline desfazerá todas as modificações para as linhas de histórico antes do retorno quando `accept-line` for executada. Por predefinição, as linhas de histórico podem ser modificadas e reter listas individuais de desfazer entre chamadas a `readline`. O valor pré-definido é '`off`'.

`show-all-if-ambiguous`
: Isso altera o comportamento pré-definido das funções de completação. Se configurada para '`on`', então as palavras as quais tenham mais que uma completação possível fazem com que as coincidências sejam listadas imediatamente, em vez de soar o alarme sonoro. O valor pré-definido é '`off`'.

`show-all-if-unmodified`
: Isso altera o comportamento pré-definido das funções de completação de uma maneira similar a *show-all-if-ambiguous*. Se configurada para '`on`', então as palavras as quais tenham

mais que uma completação possível sem qualquer possível completação parcial (as completações possíveis não compartilham um prefixo comum) fazem com que as coincidências sejam listadas imediatamente, em vez de soar o alarme sonoro. O valor pré-definido é '`off`'.

`show-mode-in-prompt`
: Se configurado para '`on`', então adiciona um carácter ao inicio do prompt, indicando o modo de edição: emacs ('`@`'); comando vi ('`:`'); ou inserção vi ('`+`'). O valor pré-definido é '`off`'.

`skip-completed-text`
: Se configurada para '`on`', então isso altera o comportamento pré-definido de completação quando da inserção, na linha, de uma coincidência única. Essa facilidade está ativa somente quando da realização de completação no meio de uma palavra. Se habilitada, Readline não insere caracteres a partir da completação que coincidam com caracteres após o ponto na palavra sendo completada, de forma que as porções da palavra seguintes ao cursor não são duplicadas. Por exemplo, se habilitada, a tentativa de completação quando o cursor estiver após o '`e`' em '`Makefile`' resultará em '`Makefile`', em vez de '`Makefilefile`', presumindo que exista uma única completação possível. O valor pré-definido é '`off`'.

`visible-stats`
: Se configurada para '`on`', então um carácter denotando um tipo do arquivo é acrescentado ao nome do arquivo, quando da listagem das completações possíveis. O valor pré-definido é '`off`'.

Key Bindings
: A sintaxe para controle das vinculações de tecla no arquivo init é simples. Primeiro, você precisa encontrar o nome do comando que você deseja modificar. As seções seguintes contém tabelas de nome de comando, a vinculação padrão, se existente, e uma breve descrição do que o comando faz.

  Uma vez que você saiba o nome do comando, simplesmente coloque em uma linha no arquivo init o nome da tecla a qual você deseja vincular o comando, uma vírgula, e então o nome do comando. Não pode existir espaço em branco entre o nome da tecla e a vírgula – isso será interpretado como sendo parte do nome da tecla. O nome da tecla pode ser expresso em diferentes maneiras, dependendo do que você acha mais confortável.

  Adicionalmente a nomes de comando, Readline permite que teclas sejam vinculadas a uma sequência de caracteres que é inserida quando a tecla for pressionada (uma *macro*).

  O comando `bind -p` exibe nomes de função Readline e vinculações em um formato que pode ser colocado diretamente em um arquivo de inicialização. veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52.

  keyname: *function-name* or *macro*
  : *keyname* é o nome de uma tecla escrito em Inglês. Por exemplo:

```
Control-u: universal-argument
Meta-Rubout: backward-kill-word
Control-o: "> output"
```

No exemplo acima, *C-u* é vinculada à função `universal-argument`, *M-DEL* é vinculada à função `backward-kill-word`, e *C-o* é vinculada para executar a macro expressa no lado direito (isto é, para inserir o texto '`> output`' na linha).

Um número de nomes simbólicos de caracteres são reconhecidos quando do processamento dessa sintaxe de vinculação de tecla: *DEL*, *ESC*, *ESCAPE*, *LFD*, *NEWLINE*, *RET*, *RETURN*, *RUBOUT*, *SPACE*, *SPC*, and *TAB*.

"*keyseq*": *function-name* or *macro*

keyseq se difere de *keyname* acima na medida em que sequências de caracteres que denotam uma sequência de tecla inteira podem ser especificadas, colocando-se a sequência de tecla entre aspas duplas. Alguns escapes de tecla estilo GNU Emacs podem ser utilizados, como no seguinte exemplo, porém os nomes especiais de caracteres não são reconhecidos.

```
"\C-u": universal-argument
"\C-x\C-r": re-read-init-file
"\e[11~": "Function Key 1"
```

No exemplo acima, *C-u* é novamente vinculada à função `universal-argument` (da mesma forma que o foi no primeiro exemplo), '*C-x C-r*' é vinculada à função `re-read-init-file`, e '`ESC [ 1 1 ~`' é vinculada para inserir o texto '`Function Key 1`'.

As seguintes sequências de escape estilo GNU Emacs estão disponíveis quando da especificação de sequência de tecla:

`\C-` prefixo control

`\M-` prefixo meta

`\e` um carácter de escape

`\\` barra invertida

`\"` ", uma marca de aspa dupla

`\'` ', uma aspa simples ou apóstrofo

Adicionalmente às sequências de escape estilo GNU Emacs, um segundo conjunto de escapes de barra invertida está disponível:

`\a` alerta sonoro (sino)

`\b` backspace

`\d` delete

`\f` alimentar formulário

`\n` nova linha

`\r` retorno de carro

`\t` tab horizontal

`\v` tab vertical

`\nnn` o carácter de oito bits cujo valor é o valor octal *nnn* (um a três dígitos)

`\xHH` o carácter de oito bit cujo valor é o valor hexadecimal *HH* (um ou dois dígitos hexadecimais)

Quando da entrada de texto de uma macro, aspas simples ou duplas devem necessariamente ser utilizadas para indicar uma definição de macro. Texto sem aspas é presumido que seja um nome de função. No corpo da macro, os escapes de barra invertida descritos acima são expandidos. Barra invertida encapsulará qualquer outro carácter no texto da macro, incluindo '`"`' e '`'`'. Por exemplo, o seguinte vinculamento fará com que '*C-x* `\`' insira uma única '`\`' na linha:

```
"\C-x\\": "\\"
```

8.3.2 Construtores Condicionais Init

Readline implementa uma facilidade similar em espírito às características de compilação condicional do preprocessador C, o qual permite que as configurações de variáveis e vinculações de tecla sejam realizadas como resultado de testes. Existem quatro diretivas de analisador utilizadas.

`$if` O construtor `$if` permite que vinculações sejam feitas baseadas no modo de edição, o terminal sendo utilizado, ou a aplicação utilizando Readline. O texto do teste estende para o fim da linha; nenhum carácter é exigido para isolá-lo.

`mode` A forma `mode=` da diretiva `$if` é utilizada para testar se Readline está no modo `emacs` ou `vi`. Isso pode ser utilizado conjuntamente com o comando '`set keymap`', por exemplo, para configurar vinculações nos mapas de teclas `emacs-standard` e `emacs-ctlx` somente se Readline for inicializado em modo `emacs`.

`term` A forma `term=` pode ser utilizada para incluir vinculações de tecla específicas para determinado terminal, talvez para vincular as saídas de sequências de tecla pelas teclas de função do terminal. A palavra no lado direito do '`=`' é testada contra ambas o nome completo do terminal e a porção do nome do terminal antes do primeiro '`-`'. Isso permite que `sun` coincida com ambos `sun` e `sun-cmd`, por exemplo.

`application`
O construtor *application* é utilizado para incluir configurações específicas para determinada aplicação. Cada programa utilizando a biblioteca Readline configura o *application name*, e você pode testar para um valor em particular. Isso poderia ser utilizado para vincular sequências de tecla a funções úteis para um programa específico. Por exemplo, o seguinte comando adiciona uma sequência de tecla que encapsula a palavra atual ou a prévia no Bash:

```
$if Bash
# Encapsula a palavra atual ou a prévia
"\C-xq": "\eb\"\ef\""
$endif
```

`$endif` Esse comando, conforme visto no exemplo anterior, termina um comando `$if`.

`$else` Os comandos nesta ramificação da diretiva `$if` são executados se o teste falhar.

`$include` Essa diretiva toma um único nome de arquivo como um argumento e lê comandos e vinculações a partir daquele arquivo. Por exemplo, a seguinte diretiva lê a partir de `/etc/inputrc`:

```
$include /etc/inputrc
```

8.3.3 Arquivo Init de Exemplo

Eis aqui um exemplo de um arquivo *inputrc*. Isso ilustra vinculação de tecla, atribuição de variável e sintaxe condicional.

```
# Este arquivo controla o comportamento de edição de entrada de linha
# para programas que utilizam a biblioteca GNU Readline.   Programas
# existentes são FTP, Bash e GDB.
#
# Você pode re-ler o arquivo inputrc com C-x C-r.
# As linhas começando com '#' são comentários.
#
# Primeiro, incluir quaisquer vinculações para todo o sistema e
# atribuições de variável a partir de /etc/Inputrc
$include /etc/Inputrc

#
# Configura várias vinculações para o modo emacs.

set editing-mode emacs

$if mode=emacs

Meta-Control-h:  backward-kill-word Texto após o nome de função é ignorado

#
# Teclas de setas direcionais no modo mini teclado
#
#"\M-OD":        backward-char
#"\M-OC":        forward-char
#"\M-OA":        previous-history
#"\M-OB":        next-history
#
# Teclas de setas direcionais no modo ANSI
#
"\M-[D":        backward-char
"\M-[C":        forward-char
"\M-[A":        previous-history
"\M-[B":        next-history
#
# Teclas de setas direcionais no modo mini teclado de 8 bits
#
#"\M-\C-OD":       backward-char
#"\M-\C-OC":       forward-char
#"\M-\C-OA":       previous-history
#"\M-\C-OB":       next-history
#
# Teclas de setas direcionais no modo ANSI de 8 bits
#
#"\M-\C-[D":       backward-char
#"\M-\C-[C":       forward-char
```

```
#"\M-\C-[A":       previous-history
#"\M-\C-[B":       next-history

C-q: quoted-insert

$endif

# Uma vinculação estilo antigo.   Isso chega a ser o padrão.
TAB: complete

# Macros que são convenientes para interação de shell
$if Bash
# editar o caminho
"\C-xp": "PATH=${PATH}\e\C-e\C-a\ef\C-f"
# preparar para digitar uma palavra entre aspas --
# insere aspas duplas abrindo e fechando
# e move para logo após a aspa de abrir
"\C-x\"": "\"\"\C-b"
# insere uma barra invertida (testando escapes de barra invertida
# em sequências e macros)
"\C-x\\": "\\"
# Encapsula a palavra atual e a prévia
"\C-xq": "\eb\"\ef\""
# Adiciona uma vinculação para atualizar/redesenhar a linha, a qual é
# desvinculada
"\C-xr": redraw-current-line
# Editar variável na linha atual.
"\M-\C-v": "\C-a\C-k$\C-y\M-\C-e\C-a\C-y="
$endif

# utiliza um alerta visível se um estiver disponível
set bell-style visible

# não reduz caracteres para 7 bits quando da leitura
set input-meta on

# permite que caracteres iso-latin1 sejam inseridos em vez de
# convertidos para sequências de meta prefixo
set convert-meta off

# exibe caracteres com o oitavo bit configurado diretamente em vez de os
# exibir como caracteres meta prefixados
set output-meta on

# se existirem mais que 150 complementações possíveis para uma palavra,
# pergunta ao usuário se esse deseja ver todas elas
set completion-query-items 150
```

```
# Para FTP
$if Ftp
"\C-xg": "get \M-?"
"\C-xt": "put \M-?"
"\M-.": yank-last-arg
$endif
```

8.4 Comandos de Readline Vinculáveis

Esta seção descreve os comandos Readline que podem ser vinculados à sequências de tecla. Você pode listar as suas vinculações de tecla executando `bind -P` ou, para um formato mais conciso, adequado para um arquivo *inputrc*, `bind -p`. (veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52). Os nomes de comando sem uma sequência de tecla que as acompanhe são desvinculados por padrão.

Nas seguintes descrições, *point* se refere à posição atual do cursor, e *mark* se refere à posição do cursor salva pelo comando `set-mark`. O testo entre o ponto e a marca é referenciado como sendo a *region*.

8.4.1 Comandos Para Movimentação

`beginning-of-line (C-a)`
: Move para o início da linha atual.

`end-of-line (C-e)`
: Move para o fim da linha.

`forward-char (C-f)`
: Move um carácter para frente.

`backward-char (C-b)`
: Move um carácter para trás.

`forward-word (M-f)`
: Move para frente para o fim da próxima palavra. Palavras são compostas de letras e dígitos.

`backward-word (M-b)`
: Move para trás para o início da palavra atual ou da anterior. Palavras são compostas de letras e dígitos.

`shell-forward-word ()`
: Move para frente para o fim da próxima palavra. Palavras são delimitadas por meta caracteres de shell não encapsulados.

`shell-backward-word ()`
: Move para trás para o início da palavra atual ou da anterior. Palavras são delimitadas por meta caracteres de shell não encapsulados.

`clear-screen (C-l)`
: Limpa a tela e redesenha a linha atual, deixando a linha atual no topo da tela.

`redraw-current-line ()`
: Atualiza a linha atual. Por padrão, isso é desvinculado.

8.4.2 Comandos Para Manipular O Histórico

`accept-line (Newline or Return)`
: Aceita a linha independentemente de onde o cursor estiver. Se essa linha não estiver vazia, adiciona ela à lista de histórico de acordo com a configuração das variáveis `HISTCONTROL` e `HISTIGNORE`. Se essa linha for uma linha de histórico modificada, então restaura a linha de histórico ao estado original dela.

`previous-history (C-p)`
: Move "para trás" ao longo da lista de histórico, resgatando o comando anterior.

`next-history (C-n)`
: Move "para frente" ao longo da lista de histórico, resgatando o próximo comando.

`beginning-of-history (M-<)`
: Move para a primeira linha no histórico.

`end-of-history (M->)`
: Move para o final do histórico de entrada, isto é, a linha atualmente sendo fornecida.

`reverse-search-history (C-r)`
: Pesquisa na direção reversa iniciando na linha atual e movendo "para cima" ao longo do histórico conforme necessário. Isso é uma pesquisa incremental.

`forward-search-history (C-s)`
: Pesquisa para frente iniciando na linha atual e movendo "para baixo" ao longo do histórico conforme necessário. Isso é uma pesquisa incremental.

`non-incremental-reverse-search-history (M-p)`
: Pesquisa na direção reversa iniciando na linha atual e movendo "para cima" ao longo do histórico conforme necessário utilizando uma pesquisa não incremental para uma sequência de caracteres fornecida pelo usuário.

`non-incremental-forward-search-history (M-n)`
: Pesquisa para frente iniciando na linha atual e movendo "para baixo" ao longo do histórico conforme necessário utilizando uma pesquisa não incremental para uma sequência de caracteres fornecida pelo usuário.

`history-search-forward ()`
: Pesquisa para frente ao longo do histórico por uma sequência de caracteres entre o início da linha atual e o ponto. A sequência de caracteres de pesquisa deve necessariamente coincidir no início de uma linha de histórico. Isso é uma pesquisa não incremental. Por padrão, esse comando é desvinculado.

`history-search-backward ()`
: Pesquisa na direção reversa ao longo do histórico por uma sequência de caracteres entre o início da linha atual e o ponto. A sequência de caracteres de pesquisa deve necessariamente coincidir no início de uma linha de histórico. Isso é uma pesquisa não incremental. Por padrão, esse comando é desvinculado.

`history-substr-search-forward ()`
: Pesquisa para frente ao longo do histórico por uma sequência de caracteres entre o início da linha atual e o ponto. A sequência de caracteres de pesquisa pode

coincidir em qualquer lugar em uma linha de histórico. Isso é uma pesquisa não incremental. Por padrão, esse comando é desvinculado.

`history-substr-search-backward ()`
: Pesquisa na direção reversa ao longo do histórico por uma sequência de caracteres entre o início da linha atual e o ponto. A sequência de caracteres de pesquisa pode coincidir em qualquer lugar em uma linha de histórico. Isso é uma pesquisa não incremental. Por padrão, esse comando é desvinculado.

`yank-nth-arg (M-C-y)`
: Insere o primeiro argumento ao comando anterior (normalmente a segunda palavra na linha anterior) no ponto. Com um argumento *n*, insere a *n*ésima palavra a partir do comando anterior (as palavras no comando anterior começam com palavra 0). Um argumento negativo insere a *n*ésima palavra a partir do final do comando anterior. Tão logo o argumento *n* seja computado, o argumento é extraído como se a expansão de histórico '`!`*n*' tivesse sido especificada.

`yank-last-arg (M-. or M-_)`
: Insere o último argumento para o comando anterior (a última palavra da entrada de histórico anterior). Com um argumento numérico, se comporta exatamente como `yank-nth-arg`. As chamadas sucessivas a `yank-last-arg` movem para trás ao longo da lista de histórico, inserindo a última palavra (ou a palavra especificada pelo argumento à primeira chamada) de cada linha em sequência. Qualquer argumento numérico fornecido a essas chamadas sucessivas determina a direção a se mover ao longo do histórico. Um argumento negativo permuta a direção ao longo do histórico (para trás ou para frente). As facilidades de expansão de histórico são utilizadas para extrair o último argumento, como se a expansão de histórico '`!$`' tivesse sido especificada.

8.4.3 Comandos Para Modificação de Texto

`end-of-file` `(usually C-d)`
: O carácter que indica o final de arquivo conforme configurado, por exemplo, por `stty`. Se esse carácter for lido quando não existirem caracteres na linha, e o ponto estiver no início da linha, Readline o interpreta como o fim da entrada e retorna EOF.

`delete-char (C-d)`
: Deleta o carácter no ponto. Se essa função estiver vinculada ao mesmo carácter como o carácter EOF do tty, como *`C-d`* comumente está, veja acima para os efeitos.

`backward-delete-char (Rubout)`
: Deleta o carácter por trás do cursor. Um argumento numérico significa kill ("recortar") os caracteres ao invés de deletá-los.

`forward-backward-delete-char ()`
: Deleta o carácter sob o cursor, a menos que o cursor esteja no fim da linha, caso no qual o carácter por trás do cursor é deletado. Por padrão, isso não é vinculado a uma tecla.

`quoted-insert (C-q or C-v)`
: Adiciona o próximo carácter digitado à linha literal. Isso é como inserir sequências de tecla como *C-q*, por exemplo.

`self-insert (a, b, A, 1, !, ...)`
: Insere a si mesmo.

`transpose-chars (C-t)`
: Arrasta o carácter antes do cursor para frente por sobre o carácter no cursor, movendo o cursor para frente também. Se o ponto de inserção estiver no final da linha, então isso transpõe os últimos dois carácteres da linha. Argumentos negativos não tem efeito.

`transpose-words (M-t)`
: Arrasta a palavra antes do ponto e cola a palavra após o ponto, movendo o ponto e colando aquela palavra também. Se o ponto de inserção estiver no fim da linha, isso transpõe as últimas duas palavras na linha.

`upcase-word (M-u)`
: Torna a palavra atual (ou seguinte) em maiúscula. Com um argumento negativo, torna um letras maiúsculas a palavra anterior, porém não move o cursor.

`downcase-word (M-l)`
: Torna a palavra atual (ou seguinte) em minúscula. Com um argumento negativo, torna um letras minúsculas a palavra anterior, porém não move o cursor.

`capitalize-word (M-c)`
: Torna a primeira letra da palavra atual (ou seguinte) em maiúscula. Com um argumento negativo, torna em maiúscula a primeira letra da palavra anterior, porém não move o cursor.

`overwrite-mode ()`
: Permuta para o modo de sobrescrita. Com um argumento numérico positivo explícito, passa para o modo de sobrescrita. Com um argumento numérico não positivo explícito, passa para o modo de inserção. Esse comando afeta somente o modo `emacs`; o modo `vi` faz sobrescrita diferentemente. Cada chamada a `readline()` inicia no modo de inserção.

  No modo de sobrescrita, os carácteres vinculados ao `self-insert` substituem o texto no ponto em vez de empurrar o texto para a direita. Os caracteres vinculados a `backward-delete-char` substituem o carácter antes do ponto com um espaço.

  Por padrão, esse comando é desvinculado.

8.4.4 Killing ("Recortando") And Yanking ("Colando")

`kill-line (C-k)`
: Kill ("recorta") o texto desde o ponto até o fim da linha.

`backward-kill-line (C-x Rubout)`
: Kill ("recorta") para trás até o início da linha.

`unix-line-discard (C-u)`
: Kill ("recorta") desde o cursor até o início da linha atual.

`kill-whole-line ()`
: Kill ("recorta") todos os caracteres na linha atual, não importando onde o ponto está. Por padrão, isso está desvinculado.

`kill-word (M-d)`
: Kill ("recorta") a partir do ponto até o fim da palavra atual, ou se entre palavras, até o fim da próxima palavra. Os limites de palavra são os mesmos que `forward-word`.

`backward-kill-word (M-DEL)`
: Kill ("recorta") a palavra por trás do ponto. Os limites de palavra são os mesmos que `backward-word`.

`shell-kill-word ()`
: Kill ("recorta") a partir do ponto até o fim da palavra atual, ou se entre palavras, até o fim da próxima palavra. Os limites de palavra são os mesmos que `shell-forward-word`.

`shell-backward-kill-word ()`
: Kill ("recorta") a palavra por trás do ponto. Os limites de palavra são os mesmos que `shell-backward-word`.

`unix-word-rubout (C-w)`
: Kill ("recorta") a palavra por trás do ponto, utilizando espaço em branco como um limite de palavra. O texto recortado é salvo na área auxiliar chamada kill-ring.

`unix-filename-rubout ()`
: Kill ("recorta") a palavra por trás do ponto, utilizando espaço em branco e o carácter barra como os limites de palavra. O texto recortado é salvo na área auxiliar chamada kill-ring.

`delete-horizontal-space ()`
: Deleta todos os espaços e tabs ao redor do ponto. Por padrão, isso está desvinculado.

`kill-region ()`
: Kill ("recorta") o texto na região atual. Por padrão, esse comando está desvinculado.

`copy-region-as-kill ()`
: Copia o texto na região para a área de memória intermediária de recorte, de maneira que o texto possa ser yanked ("colado") em algum outro lugar. Por padrão, esse comando está desvinculado.

`copy-backward-word ()`
: Copia a palavra antes do ponto para a área de memória intermediária de recorte. Os limites de palavra são os mesmos que `backward-word`. Por padrão, esse comando está desvinculado.

`copy-forward-word ()`
: Copia a palavra seguinte ao ponto para a área de memória intermediária de recorte. Os limites de palavra são os mesmos que `forward-word`. Por padrão, esse comando está desvinculado.

`yank (C-y)`
: Yank ("cola") o topo da área auxiliar de recorte na área de memória intermediária no ponto.

`yank-pop (M-y)`
: Rotaciona a área auxiliar de recorte, e yank ("cola") o novo topo. Você somente pode fazer isso se o comando prévio for `yank` ou `yank-pop`.

8.4.5 Especificando Argumentos Numéricos

`digit-argument (M-0, M-1, ... M--)`
: Adiciona esse dígito ao argumento já em acumulação, ou inicia um novo argumento. *M--* inicia um argumento negativo.

`universal-argument ()`
: Essa é uma outra maneira de especificar um argumento. Se esse comando for seguido por um ou mais dígitos, opcionalmente com um sinal de menos inicial, aqueles dígitos define o argumento. Se o comando for seguido por dígitos, a execução de `universal-argument` novamente finaliza o argumento numérico, porém do contrário é ignorado. Como um caso especial, se esse comando for imediatamente seguido por um carácter que nem seja um dígito nem um sinal de menos, então o contador de argumento para o próximo comando é multiplicado por quatro. O contador de argumento é inicialmente um, de forma que a execução dessa função pela primeira vez faz com que o argumento contabilize quatro; uma segunda vez faz com que o argumento contabilize dezesseis; e assim por diante. Por padrão, isso não é vinculado a uma tecla.

8.4.6 Deixando Readline Digitar Por Você

`complete (TAB)`
: Tenta realizar a complementação sobre o teste antes do ponto. A atual complementação realizada é específica da aplicação. Bash tenta a complementação tratando o texto como uma variável (se o texto se inicia com '`$`'); nome de usuário (se o texto se inicia com '`~`'); nome de máquina (se o texto se inicia com '`@`'); ou comando (incluindo apelidos e funções) em sequência. Se nenhuma dessas produzir uma coincidência, então a complementação de nome de arquivo é tentada.

`possible-completions (M-?)`
: Lista as possíveis complementações do texto antes do ponto. Quando da exibição das complementações, Readline configura o número de colunas utilizadas para exibir para o valor de `completion-display-width`; o valor da variável de ambiente `COLUMNS`; ou a largura da tela, nessa exata ordem.

`insert-completions (M-*)`
: Insere todas as complementações do texto antes do ponto que poderiam ter sido geradas por `possible-completions`.

`menu-complete ()`
: Semelhante a `complete`, porém substitui a palavra a ser completada com uma coincidência única a partir da lista de possíveis complementações. A

execução repetida de `menu-complete` passeia ao longo da lista de possíveis complementações, inserindo cada coincidência em sequência. No fim da lista de complementações, o alarme sonoro é soado (objeto da configuração de `bell-style`) e o texto original é restaurado. Um argumento de *n* movimenta *n* posições para frente na lista de coincidências; um argumento negativo pode ser utilizado para mover para trás ao longo da lista. Esse comando é entendido para ser vinculado à tecla `TAB`, porém está desvinculado por padrão.

`menu-complete-backward ()`
: Idêntico a `menu-complete`, porém movimenta para trás ao longo da lista de possíveis complementações, como se a `menu-complete` tivesse sido dado um argumento negativo.

`delete-char-or-list ()`
: Deleta o carácter sob o cursor se não estiver no início oi final da linha (como `delete-char`). Se estiver no fim da linha, se comporta identicamente a `possible-completions`. Esse comando está desvinculado por padrão.

`complete-filename (M-/)`
: Tenta a complementação de nome de arquivo sobre o texto antes do ponto.

`possible-filename-completions (C-x /)`
: Lista as possíveis complementações do texto antes do ponto, tratando-as como um nome de arquivo.

`complete-username (M-~)`
: Tenta complementação sobre o texto antes do ponto, tratando-o como um nome de usuário.

`possible-username-completions (C-x ~)`
: Lista as possíveis complementações do texto antes do ponto, tratando-as como um nome de usuário.

`complete-variable (M-$)`
: Tenta complementação sobre texto antes do ponto, tratando-o como uma variável de shell.

`possible-variable-completions (C-x $)`
: Lista as possíveis complementações do texto antes do ponto, tratando-as como uma variável de shell.

`complete-hostname (M-@)`
: Tenta complementação sobre o texto antes do ponto, tratando-o como um nome de máquina.

`possible-hostname-completions (C-x @)`
: Lista as possíveis complementações do texto antes do ponto, tratando-as como um nome de máquina.

`complete-command (M-!)`
: Tenta complementação sobre o texto antes do ponto, tratando-o como um nome de comando. A complementação de comando tenta coincidir o texto contra apelidos, palavras reservadas, funções de shell, comandos internos do shell, e, finalmente, nomes de arquivos executáveis, nessa ordem.

`possible-command-completions (C-x !)`
: Lista as possíveis complementações do texto antes do ponto, tratando-as como um nome de comando.

`dynamic-complete-history (M-TAB)`
: Tenta complementação sobre o texto antes do ponto, comparando o texto contra linhas originadas da lista de histórico para possíveis coincidências de complementação.

`dabbrev-expand ()`
: Tenta complementação de menu sobre o texto antes do ponto, comparando o texto contra linhas originadas da lista de histórico para possíveis coincidências de complementação.

`complete-into-braces (M-{)`
: Realiza complementação de nome de arquivo e insere a lista das complementações possíveis encapsuladas em chaves, de maneira que a lista esteja disponível para o shell (veja-se Seção 3.5.1 [Expansão de Chave], Página 23).

8.4.7 Macros ("Sequências de Comandos") de Teclado

`start-kbd-macro (C-x ()`
: Inicia o salvamento de caracteres digitados na macro de teclado atual.

`end-kbd-macro (C-x ))`
: Para o salvamento de caracteres digitados na macro de teclado atual e salva a definição.

`call-last-kbd-macro (C-x e)`
: Re-executa a última macro de teclado definida, fazendo com que os caracteres na macro apareçam como se digitados ao teclado.

`print-last-kbd-macro ()`
: Imprime a última macro de teclado definida em uma formato adequado para o arquivo *inputrc*.

8.4.8 Alguns Comandos Variados

`re-read-init-file (C-x C-r)`
: Lê o conteúdo do arquivo *inputrc*, e incorpora quaisquer vinculações ou atribuições à variáveis encontradas.

`abort (C-g)`
: Aborta o comando de edição atual e soa o alarme sonoro do terminal (objeto da configuração de `bell-style`).

`do-uppercase-version (M-a, M-b, M-x, ...)`
: Se o carácter de meta campo *x* for minúsculo, executa o comando que está vinculado ao correspondente carácter maiúsculo.

`prefix-meta (ESC)`
: "Metifica" o próximo carácter digitado. Isso é para teclados sem a tecla meta. Digitar-se '`ESC f`' é equivalente a se digitar *`M-f`*.

`undo (C-_ or C-x C-u)`
: Desfazer incremental, relembrado separadamente para cada linha.

`revert-line (M-r)`
: Desfazer todas as modificações feitas para esta linha. Isso é como se executar o comando `undo` suficientes vezes para se voltar ao início.

`tilde-expand (M-&)`
: Realiza a expansão de til sobre a palavra atual.

`set-mark (C-@)`
: Configura a marca para o ponto. Se um argumento numérico for fornecido, então a marca é configurada para aquela posição.

`exchange-point-and-mark (C-x C-x)`
: Troca o ponto pela marca. A posição atual do cursor é configurada para a posição salva, e a posição anterior do cursor é salva como a marca.

`character-search (C-])`
: Um carácter é lido e o ponto é movido para a próxima ocorrência daquele carácter. Um contador negativo busca pelas ocorrências prévias.

`character-search-backward (M-C-])`
: Um carácter é lido e o ponto é movido para a ocorrência prévia daquele carácter. Um contador negativo busca pelas ocorrências subsequentes.

`skip-csi-sequence ()`
: Lê caracteres suficientes para consumir uma sequência multi tecla como aquelas definidas para teclas como Home e End. Tais sequências iniciam com um Indicador de Sequência de Controle (ISC), geralmente ESC-[. Se essa sequência estiver vinculada à `"\e["`, então as teclas que produzem tais sequências não terão efeito, a menos que explicitamente vinculadas a um comando readline, em vez de inserir caracteres perdidos na área de memória intermediária de edição. Isso está desvinculado por padrão, porém usualmente vinculado a ESC-[.

`insert-comment (M-#)`
: Sem um argumento numérico, o valor da variável `comment-begin` é inserido no início da linha atual. Se um argumento numérico for fornecido, esse comando atua como um alternador: se os caracteres no início da linha não coincidirem com o valor de `comment-begin`, então o valor é inserido; do contrário, os caracteres em `comment-begin`, são deletados desde o início da linha. Em qualquer caso, a linha é aceita como se um newline tivesse sido digitado. O valor padrão de `comment-begin` faz com que esse comando torne a linha atual um comentário de shell. Se um argumento numérico, faz com que o carácter de comentário seja removido, a linha será executada pelo shell.

`dump-functions ()`
: Imprime todas as funções e suas vinculações de tecla para o fluxo de saída de Readline. Se um argumento numérico for fornecido, então a saída é formatada de tal maneira que ela pode se tornar parte de um arquivo *inputrc*. Esse comando é desvinculado por padrão.

`dump-variables ()`
: Imprime todas as variáveis configuráveis e seus valores para o fluxo de saída de Readline. Se um argumento numérico for fornecido, então a saída é formatada de tal maneira que ela pode ser parte de um arquivo *inputrc*. Esse comando é desvinculado por padrão.

`dump-macros ()`
: Imprime todas as sequências de tecla Readline vinculadas à macro e as sequências de caracteres que elas produzem como saída. Se um argumento numérico for fornecido, a saída é formatada de tal maneira que ela pode ser parte de um arquivo *inputrc*. Esse comando é desvinculado por padrão.

`glob-complete-word (M-g)`
: A palavra antes do ponto é tratada como um modelo para a expansão de nome de caminho, com um asterisco adicionado implicitamente. Esse modelo é utilizado para gerar a lista de nomes de arquivo de coincidentes para complementações possíveis.

`glob-expand-word (C-x *)`
: A palavra antes do ponto é tratada como um modelo para a expansão de nome de caminho, e a lista de nomes de arquivos coincidentes é inserida, substituindo a palavra. Se um argumento numérico for fornecido, então um '`*`' é acrescentado antes da expansão de nome de caminho.

`glob-list-expansions (C-x g)`
: A lista de expansões que deveria ter sido gerada por `glob-expand-word` é exibida, e a linha é redesenhada. Se um argumento numérico for fornecido, então um '`*`' é acrescentado antes da expansão de nome de caminho.

`display-shell-version (C-x C-v)`
: Exibe informação de versão acerca da instância atual de Bash.

`shell-expand-line (M-C-e)`
: Expande a linha como o shell faz. Isso realiza expansão de histórico e de apelido bem como todas as expansões de palavras do shell (veja-se Seção 3.5 [Expansões de Shell], Página 22).

`history-expand-line (M-^)`
: Realiza expansão de histórico sobre a linha atual.

`magic-space ()`
: Realiza expansão de histórico sobre a linha atual e insere um espaço (veja-se Seção 9.3 [History Interaction], Página 147).

`alias-expand-line ()`
: Realiza expansão de apelido sobre a linha atual (veja-se Seção 6.6 [Apelidos], Página 96).

`history-and-alias-expand-line ()`
: Realiza expansão de apelido e de histórico sobre a linha atual.

`insert-last-argument (M-. or M-_)`
: Um sinônimo para `yank-last-arg`.

`operate-and-get-next (C-o)`
: Aceita a linha atual para execução e traz a próxima linha relativa à linha atual a partir do histórico para edição. Qualquer argumento é ignorado.

`edit-and-execute-command (C-x C-e)`
: Invoca um editor sobre a linha de comando atual, e executa e resultado como comandos de shell. Bash tenta invocar `$VISUAL`, `$EDITOR`, e `emacs` como o editor, nessa ordem.

8.5 Modo vi de Readline

Enquanto a biblioteca Readline não tem um conjunto completo de funções de edição `vi`, ela contém o suficiente para permitir a edição simples da linha. O modo `vi` de Readline se comporta conforme especificado no padrão POSIX.

Com o objetivo de trocar interativamente entre os modos de edição `emacs` e `vi`, utilize os comandos '`set -o emacs`' e '`set -o vi`' (veja-se Seção 4.3.1 [O Comando Interno Set], Página 63). O padrão de Readline é o modo `emacs`.

Quando você entra uma linha no modo `vi`, você já está colocado no modo de "inserção", como se você tivesse digitado um '`i`'. Pressionar-se `ESC` troca-se para o modo de "comando", onde você pode editar o texto da linha com as teclas de movimento padrão do `vi`, mover para as linhas de histórico prévias com '`k`' e linhas subsequentes com '`j`', e assim por diante.

8.6 Complementação Programável

Quando a complementação de palavra for tentada para um argumento a um comando para o qual uma especificação de complementação (uma *compspec*) tiver sido definida utilizando-se o comando interno `complete` (veja-se Seção 8.7 [Comandos Internos à Complementação Programável], Página 137), as facilidades de complementação programáveis são invocadas.

Primeiro, o nome do comando é identificado. Se uma compspec tiver sido definida para aquele comando, então a compspec é utilizada para gerar a lista das complementações possíveis para a palavra. Se a palavra do comando for a sequência de caracteres vazia (complementação tentada no início de uma linha vazia), então qualquer compspec definida com a opção `-E` para `complete` é utilizada. Se a palavra de comando for um nome de caminho completo, uma compspec para o nome de caminho completo é pesquisada primeiramente. Se nenhuma compspec for encontrada para o nome de caminho completo, então uma tentativa é feita para encontrar uma compspec para a porção seguinte à barra final. Se essas buscas não resultarem em uma compspec, então qualquer compspec definida com a opção `-D` para `complete` é utilizada como o padrão.

Uma vez que uma compspec tenha sido encontrada, ela é utilizada para gerar a lista de palavras coincidentes. Se uma compspec não for encontrada, a complementação padrão de Bash descrita abaixo (veja-se Seção 8.4.6 [Commands For Completion], Página 130) é realizada.

Primeiro, as ações especificadas pela compspec são utilizadas. Somente coincidências as quais sejam prefixadas pela palavra sendo completada são retornadas. Quando a opção `-f` ou a `-d` forem utilizadas para complementação de nome de arquivo ou de nome de diretório, a variável de shell `FIGNORE` é utilizada para filtrar as coincidências. Veja-se Seção 5.2 [Variáveis do Bash], Página 75, para uma descrição de `FIGNORE`.

Quaisquer complementações especificadas por um modelo de expansão de nome de arquivo para a opção `-G` são geradas depois. As palavras geradas pelo modelo precisam coincidir com a palavra sendo complementada. A variável de shell `GLOBIGNORE` não é utilizada para filtrar as coincidências, porém a variável de shell `FIGNORE` é utilizada.

Depois, a sequência de caracteres especificada como um argumento para a opção `-W` é considerada. A sequência de caracteres é primeiro dividida utilizando os caracteres na variável especial `IFS` como delimitadores. O encapsulamento do shell é respeitado. Cada palavra é então expandida utilizando a expansão de chave, expansão de til, expansão de parâmetro e variável, substituição de comando, e expansão aritmética, conforme descrito abaixo (veja-se Seção 3.5 [Expansões de Shell], Página 22). Os resultados são divididos utilizando-se as regras descritas acima (veja-se Seção 3.5.7 [Divisão de Palavra], Página 31). Os resultados da expansão são coincididos no prefixo contra a palavra sendo complementada, e as palavras coincidentes se tornam as complementações possíveis. Após essas coincidências terem sido geradas, qualquer função de shell ou comando especificado com as opções `-F` e `-C` é invocado. Quando o comando ou função é invocada, às variáveis `COMP_LINE`, `COMP_POINT`, `COMP_KEY`, e `COMP_TYPE` são atribuídos valores conforme descrito acima (veja-se Seção 5.2 [Variáveis do Bash], Página 75). Se uma função de shell está sendo invocada, as variáveis `COMP_WORDS` e `COMP_CWORD` também são configuradas. Quando a função ou comando é invocado, o primeiro argumento ($1) é o nome do comando cujos argumentos estão sendo complementados, o segundo argumento ($2) é a palavra sendo complementada, e o terceiro argumento ($3) é a palavra que precede a palavra sendo complementada na linha de comando atual. Não é realizada nenhuma filtragem das complementações geradas contra a palavra sendo complementada; a função ou comando tem liberdade completa na geração de coincidências.

Qualquer função especificada com `-F` é invocada primeiro. A função pode utilizar quaisquer das facilidades de shell, incluindo os comandos internos `compgen` e `compopt` descritos abaixo (veja-se Seção 8.7 [Comandos Internos à Complementação Programável], Página 137), para gerar as coincidências. É necessário colocar as complementações possíveis na variável de vetor `COMPREPLY`, uma por elemento do vetor.

Depois, qualquer comando especificado com a opção `-C` é invocado em um ambiente equivalente a substituição de comando. O comando deveria imprimir uma lista de complementações, uma por linha, para a saída padrão. Barra invertida pode ser utilizada para encapsular um newline, se necessário.

Após todas as complementações possíveis serem geradas, qualquer filtro especificado com a opção `-X` é aplicado à lista. O filtro é um modelo como o utilizado para a expansão de nome de caminho; um '`&`' no modelo é substituído com o texto da palavra sendo complementada. Um '`&`' literal pode ser encapsulado com uma barra invertida; a barra invertida é removida antes da tentativa de coincidência. Qualquer complementação que coincida com o modelo será removida da lista. Um '`!`' inicial nega o modelo; nesse caso, qualquer complementação não coincidente com o modelo será removida.

Finalmente, qualquer prefixo e sufixo especificado com as opções `-P` e `-S` são adicionados a cada membro da lista de complementação, e o resultado é retornado para o código de complementação de Readline como a lista das complementações possíveis.

Se as ações previamente aplicadas não gerarem quaisquer coincidências, e a opção `-o dirnames` foi fornecida para `complete` quando o compspec foi definido, então uma complementação de nome de diretório é tentada.

Se a opção `-o plusdirs` foi fornecida a `complete` quando o compspec foi definido, então a complementação de nome de diretório é tentada e quaisquer coincidências são adicionadas aos resultados das outras ações.

Por padrão, se uma compspec for encontrada, o que quer que ela gere é retornado ao código de complementação como o conjunto completo das complementações possíveis. As complementações padrão de Bash não são tentadas, e o padrão Readline de complementação de nome de arquivo é desabilitado. Se a opção `-o bashdefault` foi fornecida a `complete` quando a compspec foi definida, então as complementações padrão de Bash são tentadas se a compspec não gerar coincidências. Se a opção `-o default` foi fornecida a `complete` quando a compspec foi definida, então a complementação padrão de Readline será realizada se a compspec (e, se tentada, as complementações padrão de Bash) não gerarem coincidências.

Quando uma compspec indica que a complementação de nome de diretório é desejada, as funções de complementação programáveis forçam Readline a adicionar uma barra aos nomes complementados os quais são links simbólicos a diretórios, sujeitos ao valor da variável de Readline *mark-directories*, não importando a configuração da variável de Readline *mark-symlinked-directories*.

Existe algum suporte para complementações dinamicamente modificantes. Isso é útil geralmente quando utilizado em combinação com uma complementação padrão especificada com `-D`. É possível para funções de shell executadas como manipuladores de complementação para indicar que a complementação deveria ser retentada retornando um código de saída de 124. Se uma função de shell retorna 124, e modifica a compspec associada com o comando sobre o qual a complementação está sendo tentada (fornecido como o primeiro argumento quando a função for executada), a complementação programável reinicia do início, com uma tentativa de encontra uma nova compspec para aquele comando. Isso permite que um conjunto de complementações seja construído dinamicamente conforme a complementação seja tentada, em vez de serem carregadas todas de uma vez.

Por exemplo, presumindo-se que existe uma biblioteca de compspecs, cada uma mantida em um arquivo correspondente ao nome do comando, a seguinte função de complementação padrão carregaria complementações dinamicamente:

```
_completion_loader()
{
    . "/etc/bash_completion.d/$1.sh" >/dev/null 2>&1 && return 124
}
complete -D -F _completion_loader -o bashdefault -o default
```

8.7 Comandos Internos à Complementação Programável

Três comandos internos estão disponíveis para manipular as facilidades de complementação programáveis: uma para especificar como os argumentos a um comando particular são para ser complementados, e dois para modificar a complementação conforme ela está ocorrendo.

`compgen`

`compgen [`*option*`] [`*word*`]`

Gera coincidências de complementação possíveis para *word* de acordo com as *option*s, o qual pode ser qualquer opção aceita pelo comando interno `complete` com exceção de `-p` e `-r`, e escreve as coincidências para a saída padrão. Quando

da utilização das opções `-F` ou `-C`, as várias variáveis de shell configuradas pelas facilidades de complementação programáveis, enquanto disponíveis, não terão valores úteis.

As coincidências serão geradas da mesma maneira como se o código de complementação programável as tivesse gerado diretamente a partir da especificação de complementação com os mesmos flags. Se *word* for especificada, somente aquelas complementação que coincidirem com *word* serão exibidas.

O valor de retorno é verdadeiro, a menos que uma opção inválida seja fornecida, ou nenhuma coincidência seja gerada.

`complete`

```
complete [-abcdefgjksuv] [-o comp-option] [-DE] [-A action] [-
G globpat] [-W wordlist]
[-F function] [-C command] [-X filterpat]
[-P prefix] [-S suffix] name [name ...]
complete -pr [-DE] [name ...]
```

Especifica como argumentos a cada *name* deveriam ser complementados. Se a opção `-p` for fornecida, ou se nenhuma opção for fornecida, as especificações de complementação existentes são impressas de uma maneira a permitir que sejam reutilizadas como entrada. A opção `-r` remove uma especificação de complementação para cada *name*, ou, se nenhum *name* for fornecido, todas as especificações de complementação. A opção `-D` indica que as opções remanescentes e ações deveriam se aplicar à complementação "padrão" de comando; isto é, a complementação tentada sobre um comando para o qual nenhuma complementação tenha sido definida previamente. A opção `-E` indica que as opções restantes e ações deveriam se aplicar a complementação "vazia" de comando; isto é, a complementação tentada sobre uma linha em branco.

O processo de aplicação dessas especificações de complementação quando a complementação de palavra é tentada está descrita acima (veja-se Seção 8.6 [Complementação Programável], Página 135). A opção `-D` tem precedência sobre `-E`.

Outras opções, se especificadas, tem os seguintes significados. Os argumentos para as opções `-G`, `-W`, e `-X` (e, se necessário, as opções `-P` e `-S`) deveriam estar encapsuladas para protegê-las da expansão antes que o comando interno `complete` seja invocado.

`-o` *comp-option*

comp-option controla vários aspectos do comportamento de compspec além da simples geração de complementações. *comp-option* pode ser um de:

`bashdefault`
: Realiza o restante das complementações padrão de Bash se o compspec não gerar coincidências.

`default`
: Utiliza a complementação de nome de arquivo padrão de Readline se o compspec não gerar coincidências.

`dirnames` Realiza complementação de nome de diretório se o compspec não gerar coincidências.

`filenames`
Informa a Readline que a compspec gera nomes de arquivo, de forma que possa realizar qualquer processamento específico de nome de arquivo (como adicionar uma barra aos nomes de diretório; encapsular caracteres especiais; ou suprimir espaços em branco ao final). Essa opção é concebida para ser utilizada com funções de shell especificadas com `-F`.

`noquote` Informa a Readline para não encapsular as palavras complementadas se elas forem nomes de arquivo (o encapsulamento de nomes de arquivo é o padrão).

`nospace` Informa a Readline para não acrescentar um espaço (o padrão) a palavras complementadas no fim da linha.

`plusdirs` Após quaisquer coincidências definidas pela compspec serem geradas, a complementação de nome de diretório é tentada e quaisquer coincidências são adicionadas aos resultados de outras ações.

`-A` *`action`* *action* pode ser um do seguinte para gerar uma lista de possíveis complementações:

`alias` Nomes de apelidos. Também pode ser especificada como `-a`.

`arrayvar` Nomes de variáveis vetor.

`binding` Nomes de vinculação de tecla de Readline (veja-se Seção 8.4 [Comandos de Readline Vinculáveis], Página 125).

`builtin` Nomes de comandos internos ao shell. Também pode ser especificada como `-b`.

`command` Nomes de comando. Também pode ser especificada como `-c`.

`directory`
Nomes de diretório. Também pode ser especificada como `-d`.

`disabled` Nomes de comandos internos ao shell desabilitados.

`enabled` Nomes de comandos internos ao shell habilitados.

`export` Nomes de variáveis de shell exportadas. Também pode ser especificada como `-e`.

`file` Nomes de arquivo. Também pode ser especificada como `-f`.

`function` Nomes de funções de shell.

`group` Nomes de grupo. Também pode ser especificada como `-g`.

`helptopic`
Tópicos de ajuda conforme aceitos pelo comando interno `help` (veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52).

`hostname` Nomes de máquina, conforme extraídos a partir do arquivo especificado pela variável de shell `HOSTFILE` (veja-se Seção 5.2 [Variáveis do Bash], Página 75).

`job` Nomes de tarefa, se o controle de tarefa estiver ativo. Também pode ser especificada como `-j`.

`keyword` Palavras reservadas de shell. Também pode ser especificada como `-k`.

`running` Nomes de tarefas em execução, se o controle de tarefas estiver ativo.

`service` Nomes de serviço. Também pode ser especificada como `-s`.

`setopt` Argumentos válidos para a opção `-o` ao comando interno `set` (veja-se Seção 4.3.1 [O Comando Interno Set], Página 63).

`shopt` Nomes de opção do shell conforme aceitos pelo comando interno `shopt` (veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52).

`signal` Nomes de sinal.

`stopped` Nomes de tarefas paradas, se o controle de tarefas estiver ativo.

`user` Nomes de usuário. Também pode ser especificada como `-u`.

`variable` Nomes de todas as variáveis do shell. Também pode ser especificada como `-v`.

`-C` *command*
command é executado em um ambiente de sub-shell, e a saída dele é utilizada como as complementações possíveis.

`-F` *function*
A função de shell *function* é executada no ambiente atual de shell. Quando ela é executada, $1 é o nome do comando cujos argumentos estão sendo complementados; $2 é a palavra sendo complementada; e $3 é a palavra que precede a palavra sendo complementada, conforme descrito acima (veja-se Seção 8.6 [Complementação Programável], Página 135). Quando ela finaliza, as possíveis complementações são resgatadas do valor da variável de vetor `COMPREPLY`.

`-G` *globpat*
: O modelo de expansão de nome de arquivo *globpat* é expandido para gerar as possíveis complementações.

`-P` *prefix*
: *prefix* é adicionado no início de cada complementação possível, após todos as outras opções terem sido aplicadas.

`-S` *suffix*
: *suffix* é adicionado no final de cada complementação possível, após todos as outras opções terem sido aplicadas.

`-W` *wordlist*
: A *wordlist* é dividida utilizando os caracteres contidos na variável especial `IFS` como delimitadores, e cada palavra resultante é expandida. As complementações possíveis são os membros da lista resultante, a qual coincide com a palavra que está sendo complementada.

`-X` *filterpat*
: *filterpat* é um modelo conforme utilizado para expansão de nome de arquivo. É aplicado à lista das possíveis complementações geradas pelas opções precedentes e argumentos, e cada complementação coincidindo com *filterpat* é removida da lista. Um '`!`' inicial no *filterpat* nega o modelo; nesse caso, qualquer complementação que não coincida com *filterpat* é removida.

O valor de retorno é verdadeiro, a menos que uma opção inválida seja fornecida; outra opção que não `-p` ou `-r` seja fornecida sem um argumento *name*; uma tentativa for feita de remover uma especificação de complementação para um *name* para o qual nenhuma especificação exista; ou um erro ocorrer quando do adicionamento de uma especificação de complementação.

`compopt`

`compopt [-o` *option*`] [-DE] [+o` *option*`] [`*name*`]`

Modifica opções de complementação para cada *name*, de acordo com as *option*s, ou para a complementação atualmente em execução, se nenhum *name* for fornecido. Se nenhuma *option* for dada, então exibe as opções de complementação para cada *name* ou a complementação atual. Os possíveis valores de *option* são aqueles válidos para o comando interno `complete` descrito acima. A opção `-D` indica que as opções restantes deveriam se aplicar à complementação "padrão" do comando; isto é, a complementação tentada sobre um comando para o qual nenhuma complementação tenha previamente sido definida. A opção `-E` indica que as opções restantes deveriam se aplicar à complementação "vazia" do comando; isto é, complementação tentada sobre uma linha em branco.

A opção `-D` tem precedência sobre `-E`.

O valor de retorno é verdadeiro, a menos que uma opção inválida seja fornecida; uma tentativa seja feita de modificar as opções para um *name* para o qual nenhuma especificação de complementação exista; ou um erro de saída ocorra.

8.8 Um Exemplo de Complementação Programável

A maneira mais comum de obter funcionalidade adicional de complementação, além das ações padrão que `complete` e `compgen` proveem é utilizar uma função de shell e vinculá-la a um comando em particular utilizando `complete -F`.

A função seguinte provê complementações para o comando interno `cd`. Ela é um razoável bom exemplo do que as funções de shell devem necessariamente fazer quando utilizadas para complementação. Esta função utiliza a palavra passada como `$2` para determinar o nome de diretório a complementar. Você também pode utilizar a variável de vetor `COMP_WORDS`; a palavra atual é indexada pela variável `COMP_CWORD`.

A função depende dos comandos internos `complete` e `compgen` para fazer a maioria do trabalho, adicionando somente as coisas que o `cd` de Bash faz além de aceitar nomes básicos de diretório: Expansão de til (veja-se Seção 3.5.2 [Expansão de Til], Página 24); busca por diretórios em *$CDPATH*; o qual é descrito acima (veja-se Seção 4.1 [Comandos Internos do Shell Bourne], Página 44); e suporte básico para a opção de shell `cdable_vars` (veja-se Seção 4.3.2 [O Comando Interno Shopt], Página 68). `_comp_cd` modifica o valor de *IFS*, de forma que ela contém apenas um marcador "newline" para acomodar nomes de arquivo que contenham espaços e tabs – `compgen` imprime as complementações possíveis que gera, uma por linha.

As complementações possíveis vão para a variável e vetor *COMPREPLY*, uma complementação por elemento do vetor. O sistema de complementação programável resgata as complementações a partir dali quando a função retorna.

```
# Uma função de complementação para o comando interno cd baseada na
# função de complementação cd oriunda do pacote bash_completion
_comp_cd()
{
    local IFS=$' \t\n'    # normaliza a IFS
    local cur _skipdot _cdpath
    local i j k

    # Expansão de til, com o efeito colateral de expandir til para nome
    # de caminho completo
    case "$2" in
    \~*)    eval cur="$2" ;;
    *)      cur=$2 ;;
    esac

    # nenhum cdpath ou nome de caminho absoluto -- complementação
    # estrita de diretório
    if [[ -z "${CDPATH:-}" ]] || [[ "$cur" == @(./*|../*|/*) ]]; then
        # compgen imprime caminhos um por linha; também poderia utilizar
        # um loop while
        IFS=$'\n'
        COMPREPLY=( $(compgen -d -- "$cur") )
        IFS=$' \t\n'
    # CDPATH+diretórios no diretório atual se não no CDPATH
```

```
    else
        IFS=$'\n'
        _skipdot=false
        # pré-processa CDPATH para converter nomes nulos de diretórios
        # para .
        _cdpath=${CDPATH/#:/.:}
        _cdpath=${_cdpath//::/:.:}
        _cdpath=${_cdpath/%:/:.}
        for i in ${_cdpath//:/$'\n'}; do
            if [[ $i -ef . ]]; then _skipdot=true; fi
            k="${#COMPREPLY[@]}"
            for j in $( compgen -d -- "$i/$cur" ); do
                COMPREPLY[k++]=${j#$i/}        # extrai diretório
            done
        done
        $_skipdot || COMPREPLY+=( $(compgen -d -- "$cur") )
        IFS=$' \t\n'
    fi

    # nomes de variável se apropriada opção de shell e nenhuma
    # complementação
    if shopt -q cdable_vars && [[ ${#COMPREPLY[@]} -eq 0 ]]; then
        COMPREPLY=( $(compgen -v -- "$cur") )
    fi

    return 0
}
```

Nós instalamos a função de complementação utilizando a opção `-F` a `complete`:

```
# Informa a readline para encapsular apropriadamente e acrescentar
# barras ao final a diretórios; utiliza a complementação padrão de bash
# para outros argumentos
complete -o filenames -o nospace -o bashdefault -F _comp_cd cd
```

Dado que nós desejamos que Bash e Readline se encarreguem de alguns outros detalhes para nós, nós utilizamos várias outras opções para informar a Bash e Readline o que fazer. A opção `-o filenames` informa a Readline que as possíveis complementações deveriam ser tratadas como nomes de arquivos, e encapsuladas apropriadamente. Essa opção também fará com que Readline acrescente ao final uma barra a nomes de arquivo que possa determinar que são diretórios (o que é o motivo pelo qual nós talvez desejemos estender `_comp_cd` para acrescentar uma barra ao final se nós estivermos utilizando diretórios encontrados via *CDPATH*: Readline não pode dizer se tais complementações são diretórios). A opção `-o nospace` informa a Readline para não acrescentar um carácter espaço ao nome de diretório, no caso de nós desejarmos acrescentar a ele. A opção `-o bashdefault` traz o restante das complementações "padrão Bash" – possível complementação que Bash adiciona ao conjunto padrão de Readline. Essas incluem coisas como complementação de nome de comando; complementação de variável para palavras iniciando com '`{`'; complementações

contendo modelos de expansão de nomes de caminho (veja-se Seção 3.5.8 [Expansão de Nome de Arquivo], Página 32); e assim por diante.

Uma vez instalada utilizando `complete`, `_comp_cd` será chamada cada vez que nós tentarmos a complementação de palavra para um comando `cd`.

Muitos exemplos mais – uma coleção extensa de complementações para a maioria dos comandos comuns GNU, Unix, e Linux – estão disponíveis como parte do projeto bash_completion. Isso é instado por padrão em muitas distribuições de GNU/Linux. Originalmente escrito por Ian Macdonald, o projeto agora reside em `http://bash-completion.alioth.debian.org/`. Existem portagens para outros sistemas, tais como Solaris e Mac OS X.

Uma versão antiga do pacote bash_completion é distribuída com bash no subdiretório `examples/complete`.

9 Utilizando o Histórico Interativamente

Este capítulo descreve como utilizar a biblioteca GNU History interativamente, a partir do ponto de vista do usuário. Ele deveria ser considerado um guia do usuário. Para informação sobre a utilização da biblioteca GNU History em outros programas, veja-se o Manual da Biblioteca GNU Readline.

9.1 Facilidades do Histórico de Bash

Quando a opção `-o history` ao comando interno `set` está habilitada (veja-se Seção 4.3.1 [O Comando Interno Set], Página 63), o shell provê acesso ao *histórico de comandos*, a lista de comandos previamente digitados. O valor da variável de shell `HISTSIZE` é utilizada como o número de comandos a salvar em uma lista de histórico. O texto dos últimos `$HISTSIZE` comandos (padrão 500) é salvo. O shell armazena cada comando em uma lista de histórico prévia a expansão de parâmetro e variável, porém após a expansão de histórico ser realizada, objeto dos valores das variáveis de shell `HISTIGNORE` e `HISTCONTROL`.

Quando o shell inicializa, o histórico é inicializado a partir do arquivo nomeado pela variável `HISTFILE` (padrão `~/.bash_history`). O arquivo nomeado pelo valor de `HISTFILE` é truncado, se necessário, para conter não mais que o número de linhas especificadas pelo valor da variável `HISTFILESIZE`. Quando um shell com histórico habilitado sai, as últimas `$HISTSIZE` linhas são copiadas a partir da lista de histórico para o arquivo nominado por `$HISTFILE`. Se a opção de shell `histappend` for configurada (veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52), então as linhas são acrescentadas ao final do arquivo de histórico, do contrário o arquivo de histórico é sobrescrito. Se `HISTFILE` for desconfigurada, ou se o arquivo de histórico estiver sem permissão de escrita, então o histórico não é salvo. Após o salvamento do histórico, o arquivo de histórico é truncado para conter não mais que `$HISTFILESIZE` linhas. Se `HISTFILESIZE` estiver desconfigurada, ou configurada para "null", para um valor não numérico ou um valor numérico menor que zero, então o arquivo de histórico não é truncado.

Se a `HISTTIMEFORMAT` estiver configurada, então a informação de marca temporal associada com cada entrada de histórico é escrita no arquivo de histórico, marcada com o carácter de comentário de histórico. Quando o arquivo de histórico é lido, as linhas iniciando com o carácter de comentário de histórico seguidas imediatamente por um dígito são interpretadas como marcas temporais para a linha de histórico prévia.

O comando interno `fc` pode ser utilizado para listar ou editar e re-executar uma porção da lista de histórico. O comando interno `history` pode ser utilizado para exibir ou modificar a lista de histórico e manipular o arquivo de histórico. Quando da utilização da edição de linha de comando, os comandos de busca estão disponíveis em cada modo de edição que provê acesso à lista de histórico (veja-se Seção 8.4.2 [Commands For History], Página 126).

O shell permite controle sobre quais comandos são salvos na lista de histórico. As variáveis `HISTCONTROL` e `HISTIGNORE` podem ser configuradas para fazer com que o shell salve somente um subconjunto dos comandos fornecidos. A opção de shell `cmdhist`, se habilitada, faz com que o shell tente salvar cada linha de um comando multilinha na mesma entrada de histórico, adicionando ponto e vírgula onde for necessário para preservar a correção sintática. A opção de shell `lithist` faz com o shell salve o comando com marcadores "newline" embutidos, em vez de ponto e vírgula. O comando interno `shopt` é utilizado para

configurar essas opções. Veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52, para uma descrição de `shopt`.

9.2 Comandos Internos ao Histórico de Bash

Bash provê dois comandos internos os quais manipulam a lista de histórico e o arquivo de histórico.

`fc`

```
fc [-e ename] [-lnr] [first] [last]
fc -s [pat=rep] [command]
```

A primeira forma seleciona um intervalo de comandos, desde *first* até *last*, da lista de histórico e exibe ou edita e os re-executa. *first* e *last* podem ser especificadas como uma sequência de caracteres (para localizar o mais recente comando iniciando com aquela sequência de caracteres) ou como um número (um índice na lista de histórico, onde um número negativo é utilizado como uma compensação a partir do número atual de comando). Se *last* não for especificado, então é configurado para *first*. Se *first* não for especificado, então é configurado para o comando prévio para edição e −16 para listagem. Se o sinalizador `-l` for dado, então os comandos são listados na saída padrão. O sinalizador `-n` suprime os números de comando quando da listagem. O sinalizador `-r` reverte a ordem da listagem. Do contrário, o editor dado por *ename* é invocado sobre um arquivo contendo aqueles comandos. Se *ename* não for dado, então o valor da seguinte expansão de variável é utilizado: `${FCEDIT:-${EDITOR:-vi}}`. Isso diz para utilizar o valor da variável `FCEDIT` se configurado, ou o valor da variável `EDITOR` se essa estiver configurada, ou `vi` se nenhuma delas estiver configurada. Quando a edição estiver completa, os comandos editados são ecoados e executados.

Na segunda forma, *command* é re-executado após cada instância de *pat* no comando selecionado ser substituída por *rep*. *command* é interpretado do mesmo jeito que *first* acima.

Um apelido útil para se utilizar com o comando `fc` é `r='fc -s'`, de maneira que digitar '`r cc`' executa o último comando iniciando com `cc` e digitar '`r`' re-executa o último comando (veja-se Seção 6.6 [Apelidos], Página 96).

`history`

```
history [n]
history -c
history -d offset
history [-anrw] [filename]
history -ps arg
```

Sem opções, exibe a lista de histórico com números de linha. As linhas prefixadas com um '`*`' foram modificadas. Um argumento de *n* lista somente as últimas *n* linhas. Se a variável de shell `HISTTIMEFORMAT` estiver configurada e não nula, então ela é utilizada como uma sequência de caracteres de formato para *strftime* exibir a marca temporal associada com cada entrada de histórico

exibida. Nenhum espaço em branco interveniente é impresso entre a marca temporal formatada e a linha de histórico.

As opções, se fornecidas, tem os seguintes significados:

`-c` Limpa a lista de histórico. Isso pode ser combinado com as outras opções para substituir a lista de histórico completamente.

`-d` *offset* Deleta a entrada de histórico na posição *offset*. *offset* deveria ser especificado conforme aparece quando o histórico é exibido.

`-a` Acrescenta ao final as novas linhas de histórico (linhas de histórico entradas desde o início da sessão atual de Bash) ao arquivo de histórico.

`-n` Acrescenta as linhas de histórico ainda não lidas a partir do arquivo de histórico à atual lista de histórico. Essas são linhas adicionadas ao arquivo de histórico desde o início da sessão atual de Bash.

`-r` Lê o arquivo de histórico e adiciona seu conteúdo à lista de histórico.

`-w` Escreve a atual lista de histórico ao arquivo de histórico.

`-p` Realiza substituição de histórico sobre os *arg*s e exibe o resultado na saída padrão, sem armazenar os resultados na lista de histórico.

`-s` Os *arg*s são adicionados ao final da lista de histórico como uma entrada única.

Quando qualquer das opções `-w`, `-r`, `-a`, ou `-n` é utilizada, se *filename* for dado, então ele é utilizado como o arquivo de histórico. Se não, então o valor da variável `HISTFILE` é utilizada.

9.3 Expansão de Histórico

A biblioteca History provê uma característica de expansão de histórico que é semelhante à expansão de histórico provida por `csh`. Esta seção descreve a sintaxe utilizada para manipular a informação de histórico.

As expansões de histórico introduzem palavras originadas da lista de histórico no fluxo de entrada, tornando fácil repetir comandos, inserir os argumentos ao comando anterior na linha de entrada atual, ou corrigir erros em comandos anteriores rapidamente.

A expansão de histórico tem lugar em duas partes. A primeira é para determinar qual linha a partir da lista de histórico deveria ser utilizada durante a substituição. A segunda é selecionar porções daquela linha para inclusão na atual. A linha selecionada a partir do histórico é chamada o *evento*, e as porções daquela linha sobre as quais se atua são chamadas *palavras*. Vários *modificadores* estão disponíveis para manipular as palavras selecionadas. A linha é quebrada em palavras no mesmo estilo que Bash faz, de forma que várias palavras envolvidas por aspas são consideradas uma palavra. As expansões de histórico são introduzidas pela aparência do carácter de expansão de histórico, o qual é '`!`' por padrão. Somente '\' e '' podem ser utilizados para encapsular o carácter de expansão de histórico.

Várias opções de shell configuráveis com o comando interno `shopt` (veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52) podem ser utilizados para adaptar o comportamento

da expansão de histórico. Se a opção de shell `histverify` estiver habilitada, e Readline estiver sendo utilizada, então as substituições de histórico não são passadas imediatamente ao analisador do shell. Em vez disso, a linha expandida é recarregada na área de memória intermediária de edição de Readline para modificações mais amplas. Se Readline estiver sendo utilizada, e a opção de shell `histreedit` estiver habilitada, então uma expansão falha de histórico será recarregada na área de memória intermediária de edição de Readline para correção.

A opção `-p` ao comando interno `history` pode ser utilizada para se ver qual expansão de histórico fará antes de utilizá-la. A opção `-s` ao comando interno `history` pode ser utilizada para adicionar comandos ao final da lista de histórico sem atualmente executá-los, de forma que eles estejam disponíveis para rechamadas subsequentes. Isso é útil geralmente em conjunção com Readline.

O shell permite controle dos vários caracteres utilizados pelo mecanismo de expansão de histórico com a variável `histchars`, conforme explanado acima (veja-se Seção 5.2 [Variáveis do Bash], Página 75). O shell utiliza o carácter de comentário de histórico para assinalar marcas temporais quando da escrita do arquivo de histórico.

9.3.1 Designadores de Evento

Um designador de evento é uma referência a uma entrada de linha de comando na lista de histórico. A menos que a referência seja absoluta, os eventos são relativos à posição atual na lista de histórico.

`!` Inicia uma substituição de histórico, exceto quando seguida por um espaço, tab, o final da linha, '`=`' ou '`(`' (quando a opção de shell `extglob` estiver habilitada utilizando-se o comando interno `shopt`).

`!`*n* Refere-se à linha de comando *n*.

`!-`*n* Refere-se ao comando *n* linhas atrás.

`!!` Refere-se ao comando prévio. Isso é um sinônimo para '`!-1`'.

`!`*string* Refere-se ao comando mais recente precedente à posição atual na lista de histórico iniciando com *string*.

`!?`*string*`[?]`
Refere-se ao comando mais recente precedente à posição atual na lista de histórico contendo *string*. O '`?`' ao final pode ser omitido se a *string* for seguida imediatamente por um marcador newline.

`^`*string1*`^`*string2*`^`
Substituição rápida. Repete o último comando, substituindo *string1* com *string2*. Equivalente a `!!:s/`*string1*`/`*string2*`/`.

`!#` A linha de comando inteira digitada longe.

9.3.2 Designadores de Palavra

Os designadores de palavra são utilizados para selecionar palavras desejadas a partir do evento. Um '`:`' separa a especificação de evento do designador de palavra. Pode ser omitido se o designador de palavra se inicia com um '`^`', '`$`', '`*`', '`-`', ou '`%`'. As palavras são

numeradas a partir do inicio da linha, com a primeira palavra sendo denotada por 0 (zero). As palavras são inseridas na linha atual separadas por espaços únicos.

Por exemplo,

`!!` designa o comando precedente. Quando você digita isso, o comando precedente é repetido literalmente.

`!!:$` designa o último argumento do comando precedente. Isso pode ser abreviado para `!$`.

`!fi:2` designa o segundo argumento do comando mais recente iniciando com as letras `fi`.

Aqui estão os designadores de palavra:

`0 (zero)` A `0`enésima palavra. Para muitas aplicações, isso é a palavra do comando.

n A *n*ésima palavra.

`^` O primeiro argumento; isto é, palavra 1.

`$` O último argumento.

`%` A palavra coincidida pela mais recente busca '`?`*string*`?`'.

x`-`*y* Um intervalo de palavras; '`-`*y*' abrevia '`0-`*y*'.

`*` Todas as palavras, exceto a `0`enésima. Isso é um sinônimo para '`1-$`'. É um erro utilizar '`*`' se existir apenas uma palavra no evento; a sequência de caracteres vazia é retornada nesse caso.

x`*` Abrevia '*x*`-$`'

x`-` Abrevia '*x*`-$`' como '*x*`*`', porém omite a última palavra.

Se um designador de palavra for fornecido sem uma especificação de evento, então o comando prévio é utilizado como o evento.

9.3.3 Modificadores

Após o designador opcional de palavra, você pode adicionar uma sequência de um ou mais dos seguintes modificadores, cada um precedido por '`:`'.

`h` Remove um componente final de nome de caminho, deixando somente a cabeça.

`t` Remove todos os componentes finais de nome de caminho, deixando a cauda.

`r` Remove um sufixo inicial da forma '*`.suffix`*', deixando o nome de base.

`e` Remove tudo, menos o sufixo final.

`p` Imprime o comando novo, porém não o executa.

`q` Encapsula as palavras substituídas, encapsulando substituições adicionais.

`x` Encapsula as palavras substituídas como com '`q`', porém quebra em palavras nos espaços, tabs, e nos marcadores newline.

`s/old/new/`
: Substitui *new* para a primeira ocorrência de *old* na linha de evento. Qualquer delimitador pode ser utilizado no lugar de '`/`'. O delimitador pode ser encapsulado em *old* e *new* com uma barra invertida única. Se '`&`' aparece em *new*, então ele é substituído por *old*. Uma barra invertida única encapsulará o '`&`'. O delimitador final é opcional se ele for o último carácter na linha de entrada.

`&`
: Repete a substituição prévia.

`g`
`a`
: Faz com que mudanças sejam aplicadas sobre a linha de evento inteira. Utilizado em conjunção com '`s`', como em `gs/old/new/`, ou com '`&`'.

`G`
: Aplica o modificador '`s`' seguinte uma vez a cada palavra no evento.

10 Instalando o Bash

Este capítulo provê instruções básicas para a instalação de Bash nas várias plataformas suportadas. A distribuição suporta os sistemas operacionais GNU, quase cada versão de Unix, e vários sistemas não Unix tais como BeOS e Interix. Outras portagens independentes existem para MS-DOS, OS/2, e plataformas Windows.

10.1 Instalação Básica

Estas são instruções de instalação para Bash.

A maneira mais simples de compilar Bash é:

1. mudar (`cd`) para o diretório contendo o código fonte e digitar '`./configure`' para configurar Bash para o seu sistema. Se você estiver utilizando `csh` em uma versão antiga de System V, pelo contrário talvez você precise digitar '`sh ./configure`' para prevenir `csh` de tentar executar o próprio `configure`.

   A execução de `configure` leva algum tempo. Enquanto em execução, ele imprime mensagens dizendo quais características ele está verificando.
2. Digite '`make`' para compilar Bash e construir o script de relatamento de erro `bashbug`.
3. Opcionalmente, digite '`make tests`' para executar a suíte de teste do Bash.
4. Digite '`make install`' para instalar `bash` e `bashbug`. Isso também instalará as páginas de manual e arquivo Info.

O script de shell `configure` tenta adivinhar os valores corretos para várias variáveis dependentes de sistema utilizadas durante a compilação. Ele utiliza tais valores para criar um `Makefile` em cada diretório do pacote (o diretório topo, dos diretórios `builtins`, `doc`, e `support`, cada diretório sob `lib`, e vários outros). O script também cria um arquivo `config.h` contendo definições dependentes de sistema. Finalmente, ele cria um script de shell chamado `config.status` que você pode executar no futuro para recriar a configuração atual, um arquivo `config.cache` que salva os resultados de seus testes para acelerar a reconfiguração, e um arquivo `config.log` contendo saída do compilador (útil principalmente para depurar `configure`). Se em certo ponto `config.cache` contém resultados que você não deseja manter, então você pode remover ou editá-lo.

Para descobrir mais sobre as opções e argumentos que o script `configure` entende, digite

```
bash-2.04$ ./configure --help
```

no prompt do Bash no seu diretório de fonte do Bash.

Se você precisar fazer coisas fora do usual para compilar Bash, por favor tente compreender como `configure` poderia verificar quando ou não fazer tais coisas, e envie as instruções ou diffs, via mensagem, para `bash-maintainers@gnu.org` de forma que possam ser consideradas para o próximo lançamento.

O arquivo `configure.ac` é utilizado para criar `configure` por um programa chamado Autoconf. Você somente precisa de `configure.ac` se você desejar modificá-lo ou regenerar `configure` utilizando uma versão mais nova de Autoconf. Se você fizer isso, tenha certeza de que você está utilizando Autoconf versão 2.50 ou superior.

Você pode remover os binários do programa e arquivos de objeto do diretório de código fonte digitando '`make clean`'. Para remover também os arquivos que `configure` criou (de

forma que você pode compilar Bash para um tipo diferente de computador), digite '`make distclean`'.

10.2 Compiladores e Opções

Alguns sistemas exigem opções não usuais para compilação ou lincagem que o script `configure` não conhece. Você pode dar a `configure` valores iniciais para variáveis configurando elas no ambiente. Na utilização de um shell compatível com Bourne, você pode fazer isso em linha de comando como isto:

```
CC=c89 CFLAGS=-O2 LIBS=-lposix ./configure
```

Em sistemas que tem o programa `env`, você pode fazê-lo assim:

```
env CPPFLAGS=-I/usr/local/include LDFLAGS=-s ./configure
```

O processo de configuração utiliza GCC para construir Bash se GCC estiver disponível.

10.3 Compilando Para Múltiplas Arquiteturas

Você pode compilar Bash para mais que uma espécie de computador ao mesmo tempo, colocando os arquivos objeto para cada arquitetura em seus próprios diretórios. Para fazer isso, você deve necessariamente utilizar uma versão de `make` que suporte a variável `VPATH`, tal como GNU `make`. Vá (`cd`) para o diretório onde você deseja que os executáveis e arquivos objeto estejam e execute o script `configure` a partir do diretório fonte. Você talvez precise fornecer o argumento `--srcdir=PATH` para informar a `configure` onde estão os arquivos fonte. `configure` automaticamente verifica o código fonte no diretório que `configure` está e em '..'.

Se você tiver que utilizar um `make` que não suporte a variável `VPATH`, então você pode compilar Bash para uma arquitetura por vez no diretório de código fonte. Após você ter instalado Bash para uma arquitetura, utilize '`make distclean`' antes de reconfigurar para uma outra arquitetura.

Alternativamente, se o seu sistema suporta links simbólicos, você pode utilizar o script `support/mkclone` para criar uma árvore de construção a qual tem link simbólicos de volta a cada arquivo no diretório fonte. Aqui está um exemplo que cria um diretório de construção no diretório atual a partir de um diretório fonte `/usr/gnu/src/bash-2.0`:

```
bash /usr/gnu/src/bash-2.0/support/mkclone -s /usr/gnu/src/bash-2.0 .
```

O script `mkclone` exige Bash, assim você deve necessariamente já ter construído Bash para pelo menos uma arquitetura antes que você possa criar diretórios de construção para outras arquiteturas.

10.4 Nomes de Instalação

Por padrão, '`make install`' instalará em `/usr/local/bin`, `/usr/local/man`, etc. Você pode especificar um outro prefixo de instalação diferente de `/usr/local` dando a `configure` a opção `--prefix=`*`PATH`*, ou especificando um valor para a variável de '`make`' `DESTDIR` quando da execução de '`make install`'.

Você pode especificar prefixos de instalação separados para arquivos específicos a uma arquitetura e arquivos independentes de arquitetura. Se você der a `configure` a opção

`--exec-prefix=`*PATH*, então '`make install`' utilizará *PATH* como o prefixo para a instalação de programas e bibliotecas. Documentação e outros arquivos de dados ainda utilizarão o prefixo regular.

10.5 Especificando o Tipo do Sistema

Talvez existam algumas características que `configure` não pode compreender automaticamente, porém precisa determinar pelo tipo de máquina na qual Bash executará. Usualmente `configure` pode compreender isso, mas se Bash imprime uma mensagem dizendo que não pode adivinhar o tipo de máquina, então dê a Bash a opção `--host=TYPE`. '`TYPE`' pode ser ou um nome curto para o tipo de sistema, tal como '`sun4`', ou um nome canônico com três campos: '`CPU-COMPANY-SYSTEM`' (por exemplo, '`i386-unknown-freebsd4.2`').

Veja-se o arquivo `support/config.sub` para os valores possíveis de cada campo.

10.6 Compartilhando Padrões

Se você desejar configurar valor padrão para os scripts `configure` para compartilhar, você pode criar um script local de shell chamado `config.site` o qual te dá valores padrão para variáveis como `CC`, `cache_file`, e `prefix`. `configure` procura se `PREFIX/share/config.site` existe. Ou, você pode configurar a variável de ambiente `CONFIG_SITE` para a localização do script local. Um alerta: O `configure` de Bash procura por um script local, porém nem todos os scripts `configure` o fazem.

10.7 Controles de Operação

`configure` reconhece as seguintes opções para controlar como ele opera.

`--cache-file=`*file*
: Utiliza e salva os resultados dos testes em *file* em vez de `./config.cache`. Configure *file* para `/dev/null` para desabilitar o caching, para a depuração de `configure`.

`--help`
: Imprime um sumário das opções para `configure`, e sai.

`--quiet`
`--silent`
`-q`
: Não imprime mensagens dizendo quais verificações estão sendo feitas.

`--srcdir=`*dir*
: Procura pelo código fonte de Bash no diretório *dir*. Usualmente `configure` pode determinar esse diretório automaticamente.

`--version`
: Imprime a versão de Autoconf utilizada para gerar o script `configure`, e sai.

`configure` também aceita algumas outras, não amplamente utilizadas, opções clichê. '`configure --help`' imprime a lista completa.

10.8 Características Opcionais

O `configure` de Bash tem um número de opções `--enable-`*feature*, onde *feature* indica uma parte opcional de Bash. Existem também várias opções `--with-`*package*, onde *package* é alguma coisa como '`bash-malloc`' ou '`purify`'. Para desligar o uso padrão de um pacote, utilize `--without-`*package*. Para configurar Bash sem uma característica que está habilitada por padrão, utilize `--disable-`*feature*.

Aqui está uma lista completa das opções `--enable-` e `--with-` que o `configure` de Bash reconhece.

`--with-afs`
: Define se você está utilizando o Sistema de Arquivo Andrew oriundo da Transarc.

`--with-bash-malloc`
: Utilize a versão Bash de `malloc` no diretório `lib/malloc`. Esse não é o mesmo `malloc` que aparece em GNU libc, mas uma versão mais antiga originalmente derivada do `malloc` do BSD 4.2. Esse `malloc` é muito rápido, porém desperdiça algum espaço em cada alocação. Essa opção está habilitada por padrão. O arquivo `NOTES` contém uma lista dos sistemas para os quais isso deveria estar desligado, e `configure` desabilita essa opção automaticamente para um número de sistemas.

`--with-curses`
: Utiliza a biblioteca curses em vez da biblioteca termcap. Isso deveria ser fornecido se o seu sistema tem uma base de dados termcap inadequada ou incompleta.

`--with-gnu-malloc`
: Um sinônimo para `--with-bash-malloc`.

`--with-installed-readline[=`*PREFIX*`]`
: Defina isso para fazer com que Bash vincule com uma versão instalada localmente de Readline em vez da versão em `lib/readline`. Isso somente funciona com Readline 5.0 e versões posteriores. Se *PREFIX* for `yes` ou não fornecido, `configure` utiliza os valores das variáveis de make `includedir` e `libdir`, as quais são subdiretórios de `prefix` por padrão, para encontrar a versão instalada de Readline se ela não estiver nos diretórios padrão de sistema include e library. Se *PREFIX* for `no`, então Bash vincula com a versão em `lib/readline`. Se *PREFIX* estiver configurada para qualquer outro valor, então `configure` a trata como um nome de caminho de diretório para a versão instalada de Readline nos subdiretórios daquele diretório (arquivos include em *PREFIX*`/include` e a biblioteca em *PREFIX*`/lib`).

`--with-purify`
: Defina isso para utilizar o verificador de alocação de memória Purify da Rational Software.

`--enable-minimal-config`
: Isso produz um shell com características mínimas, próximo do histórico shell Bourne.

Existem várias opções `--enable-` que alteram como Bash é compilado e vinculado, em vez de modificar características de tempo de execução.

`--enable-largefile`
: Habilita o suporte para large files (`http://www.sas.com/standards/large_file/x_open.20Mar96.html`) se o sistema operacional exige opções especiais de compilador para construir programas os quais podem acessar arquivos grandes. Isso está habilitado por padrão, se o sistema operacional provê suporte a arquivo grande.

`--enable-profiling`
: Isso constrói um binário Bash que produz informação de perfil para ser processada por `gprof` cada vez que for executada.

`--enable-static-link`
: Isso faz com que Bash seja vinculado estaticamente, se `gcc` estiver sendo utilizado. Isso poderia ser empregado para construir uma versão para se utilizar como o shell do root.

A opção '`minimal-config`' pode ser utilizada para desabilitar todas as opções seguintes, mas ela é processada primeiro, de forma que as opções individuais podem ser habilitadas utilizando-se '`enable-`*`feature`*'.

Todas as opções seguintes, exceto '`disabled-builtins`', '`directpand-default`', e '`xpg-echo-default`', estão habilitadas por padrão, a menos que o sistema operacional não forneça o suporte necessário.

`--enable-alias`
: Permite a expansão de apelido e inclui os comandos internos `alias` e `unalias` (veja-se Seção 6.6 [Apelidos], Página 96).

`--enable-arith-for-command`
: Inclui suporte para o formato alternativo do comando `for` que se comporta como a declaração `for` da linguagem C (veja-se Seção 3.2.4.1 [Construtores de Ciclos], Página 10).

`--enable-array-variables`
: Inclui suporte para variáveis de shell vetor unidimensional (veja-se Seção 6.7 [Vetores], Página 97).

`--enable-bang-history`
: Inclui suporte para a substituição de histórico no estilo `csh` (veja-se Seção 9.3 [History Interaction], Página 147).

`--enable-brace-expansion`
: Inclui a expansão de chave ao estilo `csh` (`b{a,b}c` ↦ `bac bbc`). Veja-se Seção 3.5.1 [Expansão de Chave], Página 23, para uma descrição completa.

`--enable-casemod-attributes`
: Inclui suporte para os atributos de modificação de caso no comando interno `declare` e declarações de atribuição. As variáveis com o atributo *uppercase*, por exemplo, terão os seus valores convertidos para maiúsculo quando da atribuição.

`--enable-casemod-expansion`
: Inclui suporte para as expansões de palavra de modificação de caso.

`--enable-command-timing`
: Inclui suporte para o reconhecimento de `time` como sendo uma palavra reservada e para a exibição de estatísticas de temporização para o canal de comunicação seguinte a `time` (veja-se Seção 3.2.2 [Canais de Comunicação], Página 8). Isso permite que os canais de comunicação bem como as funções e comandos internos de shell sejam temporizados.

`--enable-cond-command`
: Inclui suporte para o comando condicional `[[`. (veja-se Seção 3.2.4.2 [Construtores Condicionais], Página 11).

`--enable-cond-regexp`
: Inclui suporte para se coincidir as expressões regulares POSIX utilizando-se o operador binário '`=~`' no comando condicional `[[`. (veja-se Seção 3.2.4.2 [Construtores Condicionais], Página 11).

`--enable-coprocesses`
: Inclui suporte para co-processos e a palavra reservada `coproc` (veja-se Seção 3.2.2 [Canais de Comunicação], Página 8).

`--enable-debugger`
: Inclui suporte para o depurador bash (distribuído separadamente).

`--enable-direxpand-default`
: Faz com a opção de shell `direxpand` (veja-se Seção 4.3.2 [O Comando Interno Shopt], Página 68) seja habilitada por padrão quando o shell inicializa. Essa opção normalmente é desabilitada por padrão.

`--enable-directory-stack`
: Inclui suporte para a pilha de diretório ao estilo `csh` e os comandos internos `pushd`, `popd`, e `dirs` (veja-se Seção 6.8 [A Pilha de Diretório], Página 98).

`--enable-disabled-builtins`
: Permite que os comandos internos sejam invocados via '`builtin xxx`' mesmo após `xxx` tiver sido desabilitado utilizando-se '`enable -n xxx`'. Veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52, para detalhes dos comandos internos `builtin` e `enable`.

`--enable-dparen-arithmetic`
: Inclui suporte para o comando `((...))` (veja-se Seção 3.2.4.2 [Construtores Condicionais], Página 11).

`--enable-extended-glob`
: Inclui suporte para as características de coincidência de padrão estendidas descritas acima sob Seção 3.5.8.1 [Coincidência de Modelo], Página 32.

`--enable-extended-glob-default`
: Configura o valor padrão da opção de shell *extglob*, descrita acima sob Seção 4.3.2 [O Comando Interno Shopt], Página 68, para ser habilitada.

`--enable-glob-asciirange-default`
: Configura o valor padrão da opção de shell *globasciiranges*, descrita acima sob Seção 4.3.2 [O Comando Interno Shopt], Página 68, para ser habilitada. Isso controla o comportamento dos intervalos de caracteres utilizados nas expressões de chave de coincidência de padrão.

`--enable-help-builtin`
: Inclui o comando interno `help`, o qual exibe ajuda sobre os comandos internos do shell e variáveis (veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52).

`--enable-history`
: Inclui histórico de comando e os comandos internos `fc` e `history` (veja-se Seção 9.1 [Facilidades do Histórico de Bash], Página 145).

`--enable-job-control`
: Isso habilita as características de controle de tarefa (Capítulo 7 [Controle de Tarefa], Página 106), se o sistema operacional as suporta.

`--enable-multibyte`
: Isso habilita suporte a caracteres multi-byte se o sistema operacional provê o necessário suporte.

`--enable-net-redirections`
: Isso habilita a manipulação especial de nomes de arquivo da forma `/dev/tcp/`*`host`*`/`*`port`* e `/dev/udp/`*`host`*`/`*`port`* quando utilizados nas redireções (veja-se Seção 3.6 [Redireções], Página 34).

`--enable-process-substitution`
: Isso habilita a substituição de processo (veja-se Seção 3.5.6 [Substituição de Processo], Página 31) se o sistema operacional provê o necessário suporte.

`--enable-progcomp`
: Habilita as facilidades de completação programáveis (veja-se Seção 8.6 [Complementação Programável], Página 135). Se Readline não estiver habilitada, então essa opção não tem efeito.

`--enable-prompt-string-decoding`
: Liga a interpretação de um número de caracteres encapsulados por barras invertidas nas sequências de caracteres de prompt `$PS1`, `$PS2`, `$PS3`, e `$PS4`. Veja-se Seção 6.9 [Controlando o Prompt], Página 100, para uma lista completa das sequências de encapsulamento de sequências de caracteres de prompt.

`--enable-readline`
: Inclui suporte para a edição de linha de comando e histórico com a versão Bash da biblioteca Readline (veja-se Capítulo 8 [Edição de Linha de Comando], Página 110).

`--enable-restricted`
: Inclui suporte para a *restricted shell*. Se isso estiver habilitado, então Bash, quando chamado como `rbash`, entra em um modo restrito. Veja-se Seção 6.10 [O Shell Restrito], Página 101, para uma descrição do modo restrito.

`--enable-select`
: Inclui o comando composto `select`, o qual permite a geração de menus simples (veja-se Seção 3.2.4.2 [Construtores Condicionais], Página 11).

`--enable-separate-helpfiles`
: Utiliza arquivos externos para a documentação exibida pelo comando interno `help` em vez de armazenar o texto internamente.

`--enable-single-help-strings`
: Armazena o texto exibido pelo comando interno `help` como uma sequência de caracteres única para tópico de ajuda. Isso auxilia na tradução do texto para linguagens diferentes. Você talvez precise desabilitar isso se o seu compilador não pode lidar com literais de sequências de caracteres longa demais.

`--enable-strict-posix-default`
: Faz com Bash fique conformante com o padrão POSIX por padrão (veja-se Seção 6.11 [O Modo POSIX de Bash], Página 102).

`--enable-usg-echo-default`
: Um sinônimo para `--enable-xpg-echo-default`.

`--enable-xpg-echo-default`
: Faz com que o comando interno `echo` expanda caracteres encapsulados por barra invertida por padrão, sem exigir a opção `-e`. Isso configura o valor padrão da opção de shell `xpg_echo` para `on`, o qual faz com que `echo` do Bash se comporte mais como a versão especificada na Single Unix Specification, versão 3. Veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52, para uma descrição das sequências de encapsulamento que `echo` reconhece.

O arquivo `config-top.h` contém declarações '`#define`' do Preprocessador C para opções que não são configuráveis via `configure`. Algumas dessas não foram concebidas para serem modificadas; tenha cuidado com as consequências se você modificar. Leia os comentários associados com cada definição para mais informações sobre seus efeitos.

Apêndice A Relatando Bugs

Por favor, relate todos os bugs que você encontrar no Bash. Mas primeiro, você deveria ter certeza que é realmente um bug, e que aparace na versão mais recente de Bash. A versão mais recente sempre está disponível para transferência via FTP a partir de `ftp://ftp.gnu.org/pub/gnu/bash/`.

Uma vez que você determinou que um bug existe atualmente, utilize o comando `bashbug` para submeter um relatório de bug. Se você tiver um reparo, você está encorajado a enviá-lo também! Sugestões e relatórios "filosóficos" de bug podem ser enviados para `bug-bash@gnu.org` ou postados no grupo de notícias Usenet `gnu.bash.bug`.

Todos os relatórios de bug deveriam incluir:

- O número de versão de Bash.
- O hardware e sistema operacional.
- O compilador utilizado para compilar Bash.
- Uma descrição do comportamento defeituoso.
- Um script curto ou "receita" que exercite o bug e pode ser utilizado para reproduzi-lo.

`bashbug` insere os primeiros três itens automaticamente no modelo que fornece para preenchimento de um relatório de bug.

Por favor, envie todos os relatórios relativos a este manual para `bug-bash@gnu.org`.

Apêndice B Maiores Diferenças Para o Shell Bourne

Bash implementa essencialmente a mesma gramática, expansão de parâmetro e variável, redireção, e encapsulamento entre aspas que o Bourne Shell. Bash utiliza o padrão POSIX como a especificação de como essas características devem ser implementadas. Existem algumas diferenças entre o tradicional shell Bourne e Bash; esta seção detalha rapidamente as diferenças de significância. Um número dessas diferenças estão explanadas em maior profundidade nas seções anteriores. Esta seção utiliza a versão de `sh` inclusa no SVR4.2 (a última versão do histórico shell Bourne) como a referência inicial.

- Bash é conformante com POSIX, mesmo onde a especificação POSIX difere do comportamento do `sh` tradicional (veja-se Seção 6.11 [O Modo POSIX de Bash], Página 102).
- Bash tem opções de invocação multi-carácter (veja-se Seção 6.1 [Invocando o Bash], Página 87).
- Bash tem edição de linha de comando (veja-se Capítulo 8 [Edição de Linha de Comando], Página 110) e o comando interno `bind`.
- Bash provê um mecanismo de completação de palavra programável (veja-se Seção 8.6 [Complementação Programável], Página 135), e comandos internos `complete`, `compgen`, e `compopt`, para manipulá-los.
- Bash tem histórico de comando (veja-se Seção 9.1 [Facilidades do Histórico de Bash], Página 145) e os comandos internos `history` e `fc` para manipulá-lo. A lista de histórico do Bash mantém informação de marcas temporais e utiliza o valor da variável `HISTTIMEFORMAT` para exibi-las.
- Bash implementa a expansão de histórico ao estilo `csh` (veja-se Seção 9.3 [History Interaction], Página 147).
- Bash tem variáveis vetor unidimensional (veja-se Seção 6.7 [Vetores], Página 97), e as apropriadas expansões de variável e sintaxe de atribuição para utilizá-las. Vários dos comandos internos do Bash tem opções para atuar sobre os vetores. Bash provê um número de variáveis vetor internas.
- A sintaxe de encapsulamento em aspas `$'...'`, o qual expande os caracteres encapsulados por barra invertida do ANSI-C presentes no texto entre as aspas simples, é suportada (veja-se Seção 3.1.2.4 [Encapsulamento ANSI-C], Página 6).
- Bash suporta a sintaxe de encapsulamento em aspas `$"..."` para fazer a tradução, ao estilo específico do local, dos caracteres entre as aspas duplas. As opções de invocação `-D`, `--dump-strings`, e `--dump-po-strings` listam as sequências de caracteres traduzíveis encontradas em um script (veja-se Seção 3.1.2.5 [Tradução do Locale], Página 7).
- Bash implementa a palavra chave `!` para negar o valor de retorno de um canal de comunicação (veja-se Seção 3.2.2 [Canais de Comunicação], Página 8). Muito útil quando uma declaração `if` precisa atuar somente se um teste falha. A opção de Bash '`-o pipefail`' para `set` fará com que um canal de comunicação retorne um estado de falha se qualquer comando falha.
- Bash tem a palavra reservada `time` e temporizador de comando (veja-se Seção 3.2.2 [Canais de Comunicação], Página 8). A exibição das estatísticas de temporização pode ser controlada com a variável `TIMEFORMAT`.

- Bash implementa a aritmética `for (( expr1 ; expr2 ; expr3 ))` para comando, similar a linguagem C (veja-se Seção 3.2.4.1 [Construtores de Ciclos], Página 10).
- Bash inclui o comando composto `select`, o qual permite a geração de menus simples (veja-se Seção 3.2.4.2 [Construtores Condicionais], Página 11).
- Bash inclui o comando composto `[[`, o qual torna o teste condicional parte da gramática do shell (veja-se Seção 3.2.4.2 [Construtores Condicionais], Página 11), incluindo a coincidência de expressão regular opcional.
- Bash provê a coincidência opcional sem distinção entre maiúsculas e minúsculas para os construtores `case` e `[[`.
- Bash inclui a expansão de chave (veja-se Seção 3.5.1 [Expansão de Chave], Página 23) e expansão de til (veja-se Seção 3.5.2 [Expansão de Til], Página 24).
- Bash implementa apelidos de comando e os comandos internos `alias` e `unalias` (veja-se Seção 6.6 [Apelidos], Página 96).
- Bash provê aritmética de shell, o comando composto `((` (veja-se Seção 3.2.4.2 [Construtores Condicionais], Página 11), e a expansão aritmética (veja-se Seção 6.5 [Aritmética de Shell], Página 95).
- As variáveis presentes no ambiente inicial do shell são automaticamente exportadas para os processos filhos. O shell Bourne normalmente não faz isso, a menos que as variáveis sejam explicitamente marcadas utilizando-se o comando `export`.
- Bash suporta o operador de atribuição '`+=`', o qual acrescenta ao valor da variável nominada no lado esquerdo.
- Bash inclui as expansões POSIX de remoção de padrão '`%`', '`#`', '`%%`' e '`##`' para remover sequências de caracteres iniciais ou finais dos valores das variáveis (veja-se Seção 3.5.3 [Expansão de Parâmetro de Shell], Página 25).
- A expansão `${#xx}`, a qual retorna o comprimento de `${xx}`, é suportado (veja-se Seção 3.5.3 [Expansão de Parâmetro de Shell], Página 25).
- A expansão `${var:`*offset*`[:`*length*`]}`, a qual expande para a subsequência de caracteres do valor de `var` do comprimento *length*, iniciando em *offset*, está presente (veja-se Seção 3.5.3 [Expansão de Parâmetro de Shell], Página 25).
- A expansão `${var/[/]`*pattern*`[/`*replacement*`]}`, a qual coincide com *pattern* e a substitui com *replacement* no valor de `var`, está disponível (veja-se Seção 3.5.3 [Expansão de Parâmetro de Shell], Página 25).
- A expansão `${!`*prefix*`*}` expansão, a qual expande para os nomes de todas as variáveis de shell cujos os nomes se iniciam com *prefix*, está disponível (veja-se Seção 3.5.3 [Expansão de Parâmetro de Shell], Página 25).
- Bash tem expansão de variável *indireta* utilizando `${!word}` (veja-se Seção 3.5.3 [Expansão de Parâmetro de Shell], Página 25).
- Bash pode expandir parâmetros posicionais além de `$9` utilizando `${`*num*`}`.
- A forma POSIX `$()` de substituição de comando está implementada (veja-se Seção 3.5.4 [Substituição de Comando], Página 30), e é preferida a `` ` ` `` do shell Bourne (a qual também está implementada para compatibilidade com versões anteriores).
- Bash tem substituição de processo (veja-se Seção 3.5.6 [Substituição de Processo], Página 31).

- Bash atribui valores automaticamente às variáveis que proveem informação sobre o usuário atual (`UID`, `EUID`, e `GROUPS`); a máquina atual (`HOSTTYPE`, `OSTYPE`, `MACHTYPE`, e `HOSTNAME`); e a instância de Bash que está em execução (`BASH`, `BASH_VERSION`, e `BASH_VERSINFO`). Veja-se Seção 5.2 [Variáveis do Bash], Página 75, para detalhes.
- A variável `IFS` é utilizada para dividir somente os resultados da expansão, e não todas as palavras (veja-se Seção 3.5.7 [Divisão de Palavra], Página 31). Isso fecha uma brecha de segurança do shell de longa data.
- O código da expansão de chave da expansão de nome de arquivo utiliza '`!`' e '`^`' para negar o conjunto de caracteres entre as chaves. O shell Bourne utiliza somente '`!`'.
- Bash implementa o conjunto completo de operadores de expansão de nome de arquivo POSIX, incluindo *character classes*, *equivalence classes*, e *collating symbols* (veja-se Seção 3.5.8 [Expansão de Nome de Arquivo], Página 32).
- Bash implementa características de coincidência de padrão estendidas quando a opções de shell `extglob` está habilitada (veja-se Seção 3.5.8.1 [Coincidência de Modelo], Página 32).
- É possível ter uma variável e uma função com o mesmo nome; `sh` não separa os dois espaços de nomes.
- É permitido às funções do Bash ter variáveis locais utilizando o comando interno `local`, e assim funções recursivas úteis podem ser escritas (veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52).
- As atribuições a variáveis precedendo a comandos afetam somente aquele comando, mesmo comandos internos e funções (veja-se Seção 3.7.4 [Ambiente], Página 40). No `sh`, todas as atribuições a variáveis precedendo a comandos são globais, a menos que o comando seja executado a partir do sistema de arquivo.
- Bash realiza expansão de nome de arquivo sobre nomes de arquivo especificados como operandos para operadores de redireção de entrada e saída (veja-se Seção 3.6 [Redireções], Página 34).
- Bash contém o operador de redireção '`<>`', permitindo que um arquivo seja aberto tanto para leitura quanto para escrita, e o operador de redireção '`&>`', para direcionamento da saída padrão e do erro padrão para o mesmo arquivo (veja-se Seção 3.6 [Redireções], Página 34).
- Bash inclui o operador de redireção '`<<<`', permitindo que uma sequência de caracteres seja utilizada como a entrada padrão para um comando.
- Bash implementa os operadores de redireção '`[n]<&`*word*' e '`[n]>&`*word*', os quais movem um descritor de arquivo para outro.
- Bash trata um número de nomes de arquivo especialmente quando eles estão sendo utilizados em operadores de redireção (veja-se Seção 3.6 [Redireções], Página 34).
- Bash pode abrir conexões de rede para máquinas arbitrárias e serviços com os operadores de redireção (veja-se Seção 3.6 [Redireções], Página 34).
- A opção `noclobber` está disponível para evitar a sobrescrita de arquivos existentes com a redireção de saída (veja-se Seção 4.3.1 [O Comando Interno Set], Página 63). O operador de redireção '`>|`' pode ser utilizado para substituir `noclobber`.
- Os comandos internos `cd` e `pwd` (veja-se Seção 4.1 [Comandos Internos do Shell Bourne], Página 44) cada tomam as opções `-L` e `-P` para permutar entre os modos lógico e físico.

- Bash permite que uma função neutralize um comando interno com o mesmo nome, e provê acesso àquela funcionalidade do comando interno dentro da função via comandos internos `builtin` e `command` (veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52).
- O comando interno `command` permite a desabilitação seletiva de funções quando uma pesquisa de comando é realizada (veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52).
- Comandos internos individuais podem ser habilitados ou desabilitados utilizando o comando interno `enable` (veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52).
- O comando interno `exec` toma opções adicionais que permitem que usuários controlem o conteúdo do ambiente passado para o comando executado, e o que o argumento zero para o comando é para ser (veja-se Seção 4.1 [Comandos Internos do Shell Bourne], Página 44).
- As funções de shell podem ser exportadas aos filhos via ambiente utilizando-se `export -f` (veja-se Seção 3.3 [Funções de Shell], Página 18).
- Os comandos internos de Bash `export`, `readonly`, e `declare` podem receber uma opção `-f` para atuar sobre funções de shell; uma opção `-p` para exibir variáveis com vários atributos configurados em um formato que pode ser utilizado como entrada de shell; uma opção `-n` para remover vários atributos de variáveis; e argumentos '`name=value`' para configurar atributos e valores de variáveis simultaneamente.
- O comando interno `hash` permite que um nome seja associado com um nome arbitrário de arquivo, mesmo quando esse nome de arquivo não pode ser encontrado pesquisando-se o `$PATH`, utilizando-se '`hash -p`' (veja-se Seção 4.1 [Comandos Internos do Shell Bourne], Página 44).
- Bash inclui um comando interno `help` para referência rápida às facilidades de shell (veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52).
- O comando interno `printf` está disponível para exibir saída formatada (veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52).
- O comando interno `read` (veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52) lerá uma linha terminando em '\' com a opção `-r`, e utilizará a variável `REPLY` como um padrão se nenhum argumento não opção for fornecido. O comando interno `read` também aceita uma sequência de caracteres de prompt com a opção `-p` e utilizará Readline para obter a linha quando lhe for dada a opção `-e`. O comando interno `read` também tem opções adicionais para controlar entrada: a opção `-s` desligará o eco de caracteres de entrada conforme forem lidos; a opção `-t` permitirá a `read` um intervalo de tempo se a entrada não chegar dentro de um número especificado de segundos; a opção `-n` permitirá a leitura somente de um número especificados de caracteres, em vez de uma linha inteira; e a opção `-d` lerá até um carácter particular, em vez de um marcador de nova linha.
- O comando interno `return` pode ser utilizado para abortar a execução de scripts executados com os comandos internos `.` ou `source` (veja-se Seção 4.1 [Comandos Internos do Shell Bourne], Página 44).
- Bash inclui o comando interno `shopt`, para controle mais apurado das capacidades opcionais de shell (veja-se Seção 4.3.2 [O Comando Interno Shopt], Página 68), e permite

que essas opções sejam configuradas ou desconfiguradas na invocação de shell (veja-se Seção 6.1 [Invocando o Bash], Página 87).

- Bash tem muito mais comportamento opcional controlável com o comando interno `set` (veja-se Seção 4.3.1 [O Comando Interno Set], Página 63).
- A opção '`-x`' (`xtrace`) exibe comandos outros que não os comandos simples quando realizando um rastreamento de execução (veja-se Seção 4.3.1 [O Comando Interno Set], Página 63).
- O comando interno `test` (veja-se Seção 4.1 [Comandos Internos do Shell Bourne], Página 44) é ligeiramente diferente, dado que ele implementa o algoritmo POSIX, o qual especifica o comportamento baseado em um número de argumentos.
- Bash inclui o comando interno `caller`, o qual exibe o contexto de qualquer chamada de sub-rotina ativa (uma função de shell ou um script executado com os comandos internos `.` ou `source`). Isso suporta o depurador de bash.
- O comando interno `trap` (veja-se Seção 4.1 [Comandos Internos do Shell Bourne], Página 44) permite uma especificação de pseudo-sinal `DEBUG`, similar a `EXIT`. Os comandos especificados com um coletor `DEBUG` são executados antes de cada comando simples, comando `for`, comando `case`, comando `select`, cada comando `for` aritmético, e antes do primeiro comando executar em uma função de shell. O coletor `DEBUG` não é herdado por funções de shell, a menos que a função tenha dado o atributo `trace` ou a opção `functrace` tenha sido habilitada utilizando o comando interno `shopt`. A opção de shell `extdebug` tem efeitos adicionais sobre o coletor `DEBUG`.

  O comando interno `trap` (veja-se Seção 4.1 [Comandos Internos do Shell Bourne], Página 44) permite uma especificação de pseudo-sinal `ERR`, similar a `EXIT` e `DEBUG`. Os comandos especificados com um coletor `ERR` são executados após um comando simples falhar, com umas poucas exceções. O coletor `ERR` não é herdado pelas funções de shell, a menos que a opção `-o errtrace` para o comando interno `set` esteja habilitada.

  O comando interno `trap` (veja-se Seção 4.1 [Comandos Internos do Shell Bourne], Página 44) permite uma especificação de pseudo-sinal `RETURN`, similar a `EXIT` e `DEBUG`. Os comandos especificados com um coletor `RETURN` são executados antes que a execução retome após uma função de shell ou um script de shell com `.` ou `source` retorne. O coletor `RETURN` não é herdado pelas funções de shell, a menos que a função tenha dado o atributo `trace` ou a opção `functrace` tenha sido habilitada utilizando o comando interno `shopt`.
- O comando interno `type` é mais extensivo e dá mais informação acerca dos nomes que ele encontra (veja-se Seção 4.2 [Comandos Internos ao Bash], Página 52).
- O comando interno `umask` permite à opção `-p` fazer com que a saída seja exibida na forma de um comando `umask` que pode ser reutilizado como entrada (veja-se Seção 4.1 [Comandos Internos do Shell Bourne], Página 44).
- Bash implementa uma pilha de diretório ao estilo `csh`, e provê os comandos internos `pushd`, `popd`, e `dirs` para manipulá-la (veja-se Seção 6.8 [A Pilha de Diretório], Página 98). Bash também torna a pilha de diretório visível como um valor da variável de shell `DIRSTACK`.
- Bash interpreta caracteres especiais encapsulados por barra invertida nas sequências de caracteres de prompt, quando interativo (veja-se Seção 6.9 [Controlando o Prompt], Página 100).

- O modo restrito de Bash é mais útil (veja-se Seção 6.10 [O Shell Restrito], Página 101); o modo restrito do shell SVR4.2 é limitado demais.
- O comando interno `disown` pode remover uma tarefa da tabela interna de tarefa de shell (veja-se Seção 7.2 [Comandos Internos do Controle de Tarefa], Página 107) ou suprimir o envio de `SIGHUP` para uma tarefa quando o shell sai como o resultado de um `SIGHUP`.
- Bash inclui um número de características para suportar um depurador separado para scripts de shell.
- O shell SVR4.2 tem dois comandos internos relacionados aos privilégios, (`mldmode` e `priv`), não presentes no Bash.
- Bash não tem os comandos internos `stop` ou `newgrp`.
- Bash não utiliza a variável `SHACCT` ou realiza contabilidade de shell.
- O `sh` SVR4.2 utiliza uma variável `TIMEOUT` como Bash utiliza `TMOUT`.

Mais características únicas a Bash podem ser encontradas em Capítulo 6 [Características de Bash], Página 87.

B.1 Diferenças de Implementação Com O Shell SVR4.2

Dado que Bash é uma implementação completamente nova, ele não sofre de muitas das limitações do shell SVR4.2. Por exemplo:

- Bash não bifurca um sub-shell quando da redireção em ou fora de uma estrutura de controle de shell tal como uma declaração `if` ou `while`.
- Bash não permite cotas desbalanceadas. O shell SVR4.2 inserirá silenciosamente uma aspa necessária de fechamento no `EOF` sob certas circunstâncias. Isso pode ser a causa de alguns erros difíceis de encontrar.
- O shell SVR4.2 utiliza um esquema de gerenciamento de memória barroco baseado no coletamento de `SIGSEGV`. Se o shell for inicializado a partir de um processo com `SIGSEGV` (por exemplo, utilizando-se a chamada de função da biblioteca C `system()`), então o SVR4.2 se comporta realmente muito mal.
- Em uma tentativa questionável na segurança, o shell SVR4.2, quando invocado sem a opção `-p`, alterará seu real e efetivo UID e GID, se eles são menores que algum valor limite mágico, normalmente 100. Isso pode levar a resultados inesperados.
- O shell SVR4.2 não permite a usuários coletar `SIGSEGV`, `SIGALRM`, ou `SIGCHLD`.
- O shell SVR4.2 não permite que as variáveis `IFS`, `MAILCHECK`, `PATH`, `PS1`, ou `PS2` sejam desconfiguradas.
- O shell SVR4.2 trata '`^`' como o equivalente não documentado de '`|`'.
- Bash permite argumentos de opções múltiplas quando é invocado (`-x -v`); o shell SVR4.2 somente permite um argumento de opção (`-xv`). De fato, algumas versões do shell apagam tudo se o segundo argumento inicia com um '`-`'.
- O shell SVR4.2 sai de um script se qualquer comando interno falha; Bash sai de um script somente se um dos comandos internos especiais POSIX falha, e apenas para certas falhas, conforme enumerado no padrão POSIX.
- O shell SVR4.2 se comporta diferentemente quando invocado como `jsh` (ele liga o controle de tarefa).

Apêndice C Licença de Documentação Livre GNU

Versão 1.3, 03 de novembro de 2008

0. PREÂMBULO

O propósito desta licença é tornar um manual, livro de texto, ou outro documento funcional e útil *livre* no sentido da liberdade: para assegurar a qualquer pessoa a liberdade efetiva para copiar e redistribuí-lo, com ou sem modificações, ambos comercialmente ou não comercialmente. Secundariamente, esta Licença preserva para o autor e editor uma maneira de obter crédito pelos seus trabalhos, ao mesmo tempo não sendo considerado responsável por modificações feitas por outros.

Esta Licença é uma espécie de "copyleft" ("esquerdos autorais"), o que significa que trabalhos derivados do documento devem necessariamente eles mesmos serem livres no mesmo sentido. Ela complementa a Licença Pública Geral GNU, a qual é uma licença de esquerdos autorais projetada para software livre.

Nós projetamos esta Licença para utilizá-la para manuais para software livre, porque software livre precisa de documentação livre: um programa livre deveria vir com manuais provendo as mesmas liberdades que o software provê. Porém esta Licença não é limitada a manuais de software; ela pode ser utilizada para qualquer trabalho textual, independentemente de questões de assunto ou se o trabalho textual for publicado como um livro impresso. Nós recomendamos esta Licença principalmente para trabalhos cujo propósito seja instrução ou referência.

1. APLICABILIDADE E DEFINIÇÕES

Esta Licença se aplica a qualquer manual ou outro trabalho, em qualquer meio, que contenha um aviso colocado pelo detentor dos direitos autorais dizendo que ele pode ser distribuído sob os termos desta Licença. Tal aviso concede uma licença mundial, livre de patente, ilimitada na duração, para utilizar aquele trabalho sob as condições nela declaradas. O "Documento", abaixo, se refere a quaisquer desses manuais ou trabalhos. Qualquer membro do publico é um titular da licença, e é mencionado como "você". Você aceita a licença se você copiar, modificar ou distribuir o trabalho em uma forma que exija permissão sob lei de direitos autorais.

Uma "Versão Modificada" do Documento significa qualquer trabalho contendo o Documento ou uma porção dele, seja literalmente copiado, ou com modificações e/ou traduzido em outra língua.

Uma "Seção Secundária" é um apêndice nomeado ou uma seção pré-textual do Documento que lida exclusivamente com o relacionamento dos editores ou autores do Documento para com o assunto global do Documento (ou com questões relacionadas) e não contém nada que possa se conformar diretamente com aquele assunto global. (Assim, se o Documento for em parte um livro texto de matemática, uma Seção Secundária não pode explanar nada acerca de cálculos matemáticos). O relacionamento poderia

ser uma questão de conexão histórica com o assunto ou com questões relacionadas, ou de posicionamento legal, comercial, filosófico, ético ou político respeitante a eles.

As "Seções Invariantes" são certas Seções Secundárias cujos títulos são projetados, como sendo aqueles de Seções Invariantes, no aviso que diz que o Documento é publicado sob esta Licença. Se uma seção não se encaixa na definição de Secundária acima, então a seção não está autorizada a ser designada como Invariante. O Documento pode conter zero Seções Invariantes. Se o Documento não identifica quaisquer Seções Invariantes, então não existe nenhuma.

Os "Textos de Capa" são certas passagens curtas de texto que são listadas, como Textos de Primeira Capa ou Textos de Quarta-Capa, no aviso que diz que o Documento é publicado sob esta Licença. Um Texto de Primeira Capa pode ter no máximo cinco (05) palavras, e um Texto de Quarta Capa pode ter no máximo vinte e cinco (25) palavras.

Uma cópia "Transparente" do Documento significa uma cópia legível por máquina, representada em um formato cuja especificação está disponível para o público em geral, que é adequada para revisar o documento diretamente com editores de texto genéricos ou (para imagens compostas de pixeis) programas de pintura genéricos ou (para desenhos) algum editor de desenho disponível amplamente, e que seja adequado para entrada a formatadores de texto ou para tradução automática a uma variedade de formatos próprios para entrada a formatadores de texto. Uma cópia feita em um formato de arquivo contrário ao Transparente, cuja linguagem de marcação, ou ausência de linguagem de marcação, tenha sido organizada para frustrar ou desencorajar modificações subsequentes por leitores, não é Transparente. Um formato de imagem não é Transparente se utilizado para qualquer quantidade substancial de texto. Uma cópia que não é "Transparente" é chamada "Opaca".

Exemplos de formatos adequados para cópias Transparentes incluem ASCII puro sem marcações; formato de entrada Texinfo; formato de entrada LaTeX; SGML ou XML utilizando um DTD disponível publicamente; HTML simples conformante com o padrão; PostScript ou PDF projetado para modificação humana. Exemplos de formatos transparantes de imagens incluem PNG, XCF e JPG. Formatos opacos incluem formatos proprietários que podem ser lidos e editados somente por processadores proprietários de palavra; SGML ou XML para os quais o DTD e/ou as ferramentas de processamentos não estejam disponíveis genericamente; e o HTML gerado por máquina; PostScript ou PDF produzidos por alguns processadores de palavra apenas para propósitos de saída.

A "Página de Título" significa, para um livro impresso, a própria página de título, mais tantas páginas seguintes quantas sejam necessárias para manter, legivelmente, o material que esta Licença exige para aparecer na página de título. Para trabalhos em formatos que não tenham qualquer página de título como tal, "Página de Título" significa o texto próximo da mais proeminente aparição do título do trabalho, precedendo o início do corpo do texto.

O "editor" significa qualquer pessoa ou entidade que distribui cópias do Documento ao público.

Uma seção "Intitulada XYZ" significa uma subunidade nomeada do Documento cujo título ou é precisamente XYZ ou contém XYZ entre parênteses seguinte ao texto que traduz XYZ em outra linguagem. (Aqui XYZ significa um nome específico de seção

mencionado abaixo, tais como “Agradecimentos”; “Dedicatórias”; “Patrocínios”; ou “Histórico”). “Preservar o Título” de tal seção quando você modificar o Documento significa que ele permanece uma seção “Intitulada XYZ” de acordo com essa definição.

O Documento pode incluir Declarações de Garantia próximas ao aviso que declara que esta Licença se aplica ao Documento. Essas Declarações de Garantia são consideradas como inclusas por referência nesta Licença, porém somente com relação à negação de garantias: qualquer outra implicação que essas Declarações de Garantia possam ter é inválida e não tem efeito sobre o significado desta Licença.

2. CÓPIA LITERAL

   Você pode copiar e distribuir o Documento em qualquer meio, ambos comercialmente e não comercialmente, contanto que esta Licença, os avisos de direitos autorais, e o aviso de licença dizendo que esta Licença se aplica ao Documento estejam reproduzidas em todas as cópias, e que você não adiciona quaisquer outras condições, quaisquer que sejam, àquelas desta Licença. Você não pode utilizar medidas técnicas para obstruir ou controlar a leitura ou posteriores cópias das cópias que você fizer ou distribuir. Entretanto, você pode aceitar remuneração em troca das cópias. Se você distribui um número de cópias grande o suficiente, você deve necessariamente também seguir as condições na seção três (3).

   Você também pode ceder cópias, sob as mesmas condições declaradas acima, e você pode publicamente exibir cópias.

3. CÓPIAS EM QUANTIDADE

   Se você publicar cópias impressas (ou cópias em mídia que geralmente tem capas impressas) do Documento, em número maior que cem (100), e o aviso de licença do Documento exigir Textos de Capa, você deve necessariamente encartar as cópias em capas que transportem, claramente e legivelmente, todos estes Textos de Capa: Textos de Primeira Capa na primeira capa, e Textos de Quarta Capa na capa traseira. Ambas as capas devem necessariamente também claramente e legivelmente identificar você como o editor dessas cópias. A capa frontal deve necessariamente apresentar o título completo com todas as palavras do título igualmente proeminentes e visíveis. Você pode adicionar outros materiais nas capas adicionalmente. As cópias com modificações limitadas às capas, tanto quanto preservem o título do Documento e satisfaçam essas condições, podem ser tratadas como cópias literais em relação a outros aspectos.

   Se os textos exigidos para ambas as capas forem muito volumosos para caber legivelmente, você deveria colocar os primeiros listados (tantos quantos caibam razoavelmente) na capa atual, e continuar o restante em páginas adjacentes.

   Se você publicar ou distribuir cópias Opacas do Documento em número maior que cem (100), você deve necessariamente ou incluir uma cópia Transparente, legível por máquina, junto com cada cópia Opaca, ou declarar, na ou com cada cópia Opaca, uma localização de rede de computador, a partir da qual o público usuário de rede geral tenha acesso para baixar, utilizando protocolos de rede de padrão público, uma cópia Transparente completa do Documento, livre do material adicionado.

   Se você se utilizar da última opção, você deve necessariamente adotar razoavelmente passos prudentes, quando você iniciar a distribuição de cópias Opacas em quantidade, para se assegurar que essa cópia Transparente permanecerá então acessível na localização declarada até pelo menos um ano após a última vez que você distribuiu uma

cópia Opaca (diretamente ou por intermédio dos seus agentes ou varejistas) daquela edição ao público.

É pedido, mas não exigido, que você contate os autores do Documento bem antes de redistribuir qualquer número grande de cópias, para dá-los a oportunidade de lhe fornecer uma versão atualizada do Documento.

4. MODIFICAÇÕES

Você pode copiar e distribuir uma Versão Modificada do Documento sob as condições das seções dois (2) e três (3) acima, contanto que você publique a Versão Modificada precisamente sob esta Licença, com a Versão Modificada preenchendo a função do Documento, portanto licenciando a distribuição e modificação da Versão Modificada a quem quer que possua uma cópia dela. Adicionalmente, você deve necessariamente fazer estas coisas na Versão Modificada:

A. Utilize na Página de Título (e nas capas, se existentes) um título distinto daquele do Documento, e daqueles das versões prévias (as quais deveriam, se existiu alguma, serem listadas na seção Histórico do Documento). Você pode utilizar o mesmo título que uma versão prévia, se o editor original daquela versão conceder permissão.

B. Liste na Página de Título, como autores, uma ou mais pessoas ou entidades responsáveis pela autoria das modificações na Versão Modificada, junto com ao menos cinco dos autores principais do Documento (todos os autores principais, se tiver menos que cinco), a menos que eles liberem você dessa exigência.

C. Declare na Página de Título o nome do editor da Versão Modificada, como o editor.

D. Preserve todos os avisos de direitos autorais do Documento.

E. Adicione um aviso apropriado de direitos autorais para suas modificações, adjacente aos outros avisos de direitos autorais.

F. Inclua, imediatamente após os avisos de direitos autorais, um aviso de licença concedendo ao público permissão para utilizar a Versão Modificada sob os termos desta Licença, na forma mostrada no Adendo abaixo.

G. Preserve, naquele aviso de licença, as listas completas de Seções Invariantes e Textos de Capa exigidos dados no aviso de licença do Documento.

H. Inclua uma cópia inalterada desta Licença.

I. Preserve a seção intitulada "Histórico", Preserve seu Título, e adicione a ele um item declarando ao menos o título, ano, novos autores, e editor da Versão Modificada, conforme dado na Página de Título. Se não existir uma seção intitulada "Histórico" no Documento, crie uma declarando o título, ano, autores, e editor do Documento, conforme dado em sua Página de Título, então adicione um item descrevendo a Versão Modificada, conforme declarado na frase prévia.

J. Preserve a localização de rede, se existente, dada no Documento para acesso público a uma cópia Transparente do Documento, e da mesma forma as localizações de rede dadas no Documento para versões prévias nas quais foi baseado. Essas podem ser colocadas na seção "Histórico". Você pode omitir uma localização de rede para um trabalho que foi publicado nos últimos quatro anos anteriores à publicação do próprio do Documento, ou se o editor original da versão à qual a localização de rede se refere conceder permissão.

K. Para cada seção Intitulada “Agradecimentos” ou “Dedicatórias”, Preserve o Título da seção, e preserve na seção toda a substância e tonalidade de cada um dos agradecimentos a contribuidores e/ou dedicatórias dadas nela.

L. Preserve todas as Seções Invariantes do Documento, inalteradas em seus textos e em seus títulos. Os números de Seção ou o equivalente não são considerados parte dos títulos de seção.

M. Delete quaisquer seções Intituladas “Patrocínios”. Tal seção não pode ser incluída na Versão Modificada.

N. Não reintitule qualquer seção existente para Intitulada “Patrocínios” ou para conflitar no título com qualquer Seção Invariante.

O. Preserve quaisquer Declarações de Garantia.

Se a Versão Modificada incluir novas seções pré textuais ou apêndices que se qualifiquem como Seções Secundárias e não contenham material copiado a partir do Documento, você pode, a sua escolha, designar algumas ou todas essas seções como Invariantes. Para fazer isso, adicione seus títulos à lista das Seções Invariantes no aviso de licença da Versão Modificada. Esses títulos devem necessariamente serem distintos de quaisquer outros títulos de seções.

Você pode adicionar uma seção Intitulada “Patrocínios”, contanto que ela não contenha nada além de patrocínios da sua Versão Modificada por vários patrocinadores—por exemplo, declarações de avaliadores ou aquelas de que o texto foi aprovado por uma organização como a definição autorizativa de um padrão.

Você pode adicionar uma passagem de até cinco palavras, como um Texto de Primeira Capa, e uma passagem de até vinte e cinco palavras, como um Texto de Quarta Capa, ao final da lista dos Textos de Capa na Versão Modificada. Somente uma passagem de Texto de Primeira Capa e uma de Texto de Quarta Capa podem ser adicionadas por (ou mediante acordos feitos por) qualquer uma entidade. Se o Documento já inclui um texto de capa para a mesma capa, previamente adicionado por você ou por acordo feito pela mesma entidade pela qual você está atuando, você não pode adicionar outro; porém você pode substituir o antigo, na permissão explícita do editor prévio que adicionou o antigo.

O(s) autor(s) e editor(s) do Documento, por esta Licença, não concedem permissão para utilizar seus nomes para publicidade para ou para afirmar ou implicar patrocínio de qualquer Versão Modificada.

5. COMBINANDO DOCUMENTOS

Você pode combinar o Documento com outros documentos publicados sob esta Licença, sob os termos definidos na seção quatro (4) acima para versões modificadas, contanto que você inclua na combinação todas as Seções Invariantes de todos os documentos originais, não modificados, e listá-los todos como Seções Invariantes do seu trabalho combinado no seu aviso de licença, e você preserva todas as Declarações de Garantias deles.

O trabalho combinado precisa conter somente uma cópia desta Licença, e múltiplas Seções Invariantes idênticas podem ser substituídas por uma cópia única. Se existirem múltiplas Seções Invariantes com o mesmo nome, mas conteúdos diferentes, torne o título de cada uma de tal seção único adicionando ao final dele, entre parênteses, o

nome do autor ou editor original daquela seção se conhecido, ou, do contrário, um número único. Faça o mesmo ajuste aos títulos da seção na lista de Seções Invariantes no aviso de licença do trabalho combinado.

Na combinação, você deve necessariamente combinar quaisquer seções Intituladas "Histórico" nos vários documentos originais, formando uma seção Intitulada "Histórico"; de mesma maneira, combine quaisquer seções Intituladas "Agradecimentos", e quaisquer seções Intituladas "Dedicatórias". Você deve necessariamente deletar todas as seções Intituladas "Patrocínios".

6. COLEÇÕES DE DOCUMENTOS

Você pode produzir uma coleção consistente do Documento e outros documentos publicados sob esta Licença, e substitua as cópias individuais desta Licença nos vários documentos por uma cópia única que esteja incluída na coleção, contanto que você siga as regras desta Licença para cópias literais de cada um dos documentos em todos os outros aspectos.

Você pode extrair um documento único de tal coleção, e distribuí-lo individualmente sob esta Licença, contanto que você insira uma cópia desta Licença no documento extraído, e siga esta Licença em todos os outros aspectos relativos à cópias literais daquele documento.

7. AGREGAÇÃO COM TRABALHOS INDEPENDENTES

Uma compilação do Documento ou seus derivados com outros documentos separados e independentes ou trabalhos, dentro ou junto a volume de armazenamento ou meio de distribuição, é chamado em "agregado" se os direitos autorais resultantes da compilação não forem utilizados para limitar os direitos legais dos usuários da compilação além do que os trabalhos individuais permitem. Quando o Documento for incluído em um agregado, esta Licença não se aplica aos outros trabalhos no agregado, os quais não são eles próprios trabalhos derivados do Documento.

Se a exigência do Texto de Capa da seção três (3) for aplicável a essas cópias do Documento, então se o Documento for menor que a metade do agregado inteiro, os Textos de Capa do Documento podem ser colocados em capas que encartem o Documento dentro do agregado, ou o equivalente eletrônico de capas se o Documento estiver em formato eletrônico. Do contrário, eles devem necessariamente aparecer nas capas impressas que encartem o agregado inteiro.

8. TRADUÇÃO

Tradução é considerada um tipo de modificação, de forma que você pode distribuir traduções do Documento sob os termos da seção quatro (4). A substituição de Seções Invariantes por traduções exige permissão especial de seus detentores dos direitos autorais, porém você pode incluir traduções de algumas ou todas as Seções Invariantes adicionalmente às versões originais dessas Seções Invariantes. Você pode incluir uma tradução desta Licença, e todos os avisos de licença no Documento, e quaisquer Declarações de Garantia, contanto que você inclua também a versão original em Inglês desta Licença e as versões originais daqueles avisos e declarações. No caso de uma divergência entre a tradução e a versão original desta Licença ou um aviso ou declaração, a versão original prevalecerá.

Se uma seção no Documento for Intitulada “Agradecimentos”, “Dedicatórias”, ou “Histórico”, a exigência (seção 4) de Preservar seu Título (seção 1) tipicamente exigirá a modificação do título atual.

9. FINALIZAÇÃO

   Você não pode copiar, modificar, sublicenciar, ou distribuir o Documento, exceto conforme expressamente provido sob esta Licença. Qualquer tentativa clandestina de copiar, modificar, sublicenciar, ou distribuir o Documento é inválida, e automaticamente finalizará seus direitos sob esta Licença.

   Entretanto, se você cessar todas as violações a esta Licença, então a sua licença oriunda de um detentor de direitos autorais em particular está restabelecida (a) provisoriamente, a menos e até que o detentor dos direitos autorais explicita e finalmente cancele sua licença; e (b) permanentemente, se o detentor dos direitos autorais falhar em notificar você da violação, por algum meio razoável, antes de sessenta (60) dias após a cessação.

   Além disso, a sua licença oriunda de um detentor de direitos autorais em particular está restabelecida permanentemente se o detentor dos direitos autorais notificar você sobre a violação por algum meio razoável, essa for a primeira vez que você recebeu um aviso de violação desta Licença (para qualquer trabalho) oriunda daquele detentor de direitos autorais, e você sanar a violação antes de decorridos trinta (30) dias após o seu recebimento do aviso.

   A finalização dos seus direitos sob esta seção não finaliza as licenças de varejistas que tenham recebido cópias ou direitos de você sob esta Licença. Se os seus direitos tiverem sido finalizados e não permanentemente restabelecidos, o recebimento de uma cópia de algum ou de tudo do mesmo material não concede a você direitos de utilizá-lo.

10. REVISÕES FUTURAS DESTA LICENÇA

    A Free Software Foundation pode publicar novas, revisadas versões da Licença de Documentação Livre GNU de tempos em tempos. Tais novas versões serão similares na essência à presente versão, porém podem diferir em detalhes para abarcar novos problemas ou assuntos. Veja-se `http://www.gnu.org/copyleft/`.

    Para cada versão da Licença é dado um número distintivo de versão. Se o Documento especifica que uma versão numerada em particular desta Licença “ou qualquer versão posterior” se aplica a ele, você tem a opção de seguir os termos e condições ou da versão especificada ou de qualquer versão posterior que tenha sido publicada (não como um rascunho) pela Free Software Foundation. Se o Documento não especifica um número de versão desta Licença, você pode escolher qualquer versão já publicada (não como um rascunho) pela Free Software Foundation. Se o Documento especifica que um procurador pode decidir quais versões futuras desta Licença podem ser utilizadas, essa declaração pública do procurador de aceitação de uma versão permanentemente autoriza você a escolher aquela versão para o Documento.

11. RELICENCIAMENTO

    “Sítio de Colaboração Massiva Multi autor” (ou “Sítio MMC”) significa qualquer servidor da Rede Mundial de Computadores que publica trabalhos sujeitos a direitos autorais e também provê facilidades proeminentes para qualquer pessoa editar esses trabalhos. Um wiki público que qualquer pessoa pode editar é um exemplo de tal servidor. Uma

"Colaboração Massiva Multi autor" (ou "MMC") contida no sítio significa qualquer conjunto de trabalhos sujeitos a direitos autorais assim publicados no sítio MMC.

"CC-BY-SA" significa a licença Creative Commons Attribution-Share Alike 3.0 publicada pela Creative Commons Corporation, uma corporação sem fins lucrativos com seu domicílio empresarial situado em São Francisco, Califórnia, Estados Unidos da América do Norte, bem como versões futuras de esquerdos autorais dessa licença publicadas pela mesma organização.

"Incorporar" significa publicar ou republicar um Documento, no todo ou em parte, como parte de outro Documento.

Um MMC é "elegível para relicenciamento" se ele for licenciado sob esta Licença, e se todos os trabalhos que foram primeiro publicados sob esta Licença em algum lugar que não esse MMC, e subsequentemente incorporados, no todo ou em parte, no MMC, (1) não tinham textos de capa ou seções invariantes; e (2) estavam assim incorporados antes de 01 de novembro de 2008.

O operador de um Sítio MMC pode republicar um MMC contido no sítio sob CC-BY-SA, no mesmo sítio, a qualquer tempo antes de 01 de agosto de 2009, contanto que o MMC seja elegível para relicenciamento.

ADENDO: Como utilizar esta Licença para seus documentos

Para utilizar esta Licença em um documento que você escreveu, inclua um cópia da Licença no documento e coloque os seguintes avisos de direitos autorais e licença pouco depois da página de título:

> Direitos autorais (C) *ano seu nome*.
> Permissão é concedida para copiar, distribuir e/ou modificar este documento sob os termos da Licença de Documentação Livre GNU, Versão 1.3 ou qualquer versão posterior publicada pela Free Software Foundation; sem Seções Invariantes, sem Textos de Primeira Capa, e sem Textos de Quarta Capa. Uma cópia da licença está inclusa na seção intitulada ‘‘Licença de Documentação Livre GNU’’.

Se você tiver Seções Invariantes, Textos de Primeira Capa e Textos de Quarta Capa, substitua a linha "sem...Capa" por isto:

> com as Seções Invariantes sendo *liste seus títulos*, com os Textos de Primeira Capa sendo *lista*, e com os Textos de Quarta Capa sendo *lista*.

Se você tiver Seções Invariantes sem Textos de Capa, ou alguma outra combinação dos três, mescle essas duas alternativas para adequar a situação.

Se o seu documento contém exemplos não triviais de código de programação, nós recomendamos publicar esses exemplos em paralelo, sob sua escolha de licença de software livre, tal como a Licença Pública Geral GNU, para permitir seu uso em software livre.

Apêndice D GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any,

be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their

titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Apêndice E Índices

E.1 Índice dos Comandos Internos ao Shell

T

U

W

E.2 Índice das Palavras Reservadas do Shell

!

[

]

|

C

D

E

F

I

S

T

U

W

E.3 Índice dos Parâmetros e Variáveis

!

#

$

*

-

?

@

_

0

A

B

C

D

E

F

G

H

I

K

L

M

O

P

R

S

T

U

V

E.4 Índice das Funções

A

B

C

D

E

F

G

H

I

K

M

N

O

P

Q

R

S

T

U

Y

E.5 Índice dos Conceitos

Á

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

Y