Memo

- bashまたはFreeBSDのshでも、while文をパイプに接続した場合、つまり「`cat file | while read …`」の形で記述するとwhile文が暗黙のサブシェルになるため、whileループ中で`exit`できなかったり、設定したシェル変数等をwhileループを抜けた後に持ち越せないなどの問題が発生します。シェルとしてkshまたはzshを使うと、この場合でもwhile文はサブシェルになりません。

>Appendix
サンプルスクリプト

引数の解釈状況をチェックする

Linux (bash)
FreeBSD (sh)
Solaris (sh)

使い方 argcheck [引数 ...]

例
```
argcheck -f "$file"  ……いろいろな引数を付けて展開結果をチェックする
```

基本事項

シェルの解釈の結果、実際にどのような引数がコマンドに渡されているかをチェックする「argcheck」というシェルスクリプトを作成します。

解説

シェル上でコマンドを起動する際には、その引数やコマンド名に対して、パラメータ展開、コマンド置換などの各種解釈が行われます。この解釈は場合によっては複雑になり、とくに、シングルクォート(' ')、ダブルクォート(" ")などのクォートも絡んでいる場合、結局どのような引数がコマンドに渡されるのか、ひと目では判断できなくなることがあります。

そこで、**リストA**の「argcheck」というシェルスクリプトを作成しておき、適宜引数の解釈状況をチェックすると便利でしょう。この「argcheck」は**"$HOME"/bin**または**/usr/local/bin**などの実行パスの通ったディレクトリにインストールして使用します。

引数の解釈状況をチェックするしくみ

「argcheck」では、for文と特殊パラメータ"$@"を使い、argcheckコマンド自体に付けられたすべての引数に対してシェル変数argを使ってループしています。

ループ中にはechoコマンドがあり、1行に1引数ずつ表示が行われます。echoコマンドの引数の\'"$arg"\' は複雑な形に見えますが、これはシェル変数argの参照である$argを、シェルの解釈を避けるためダブルクォートで囲んで"$arg"とし、この前後に単なる文字としてシングルクォートを付加したものです。シングルクォートは、特殊な意味を打ち消すため、バックスラッシュを付けて\'としています。

シングルクォートを付けて表示するのは、引数がスペースや改行を含んでいた場合でもその内容が確認できるようにするためです。実際の引数の内容は、argcheckの実行結果から前後のシングルクォートを除いたものになります。

argcheckの使用例

このargcheckの使用例を**図A**に示します。

まず、図の最初の例のように-fと"$HOME"という2個の引数を付けた場合、-fはそのまま、"$HOME"はシェル変数**HOME**の値に展開されて表示されることがわかります。

次に、「'one two '」のように、引数中にスペースを含んだ文字列を、全体で1個の引数として与えると、スペースも保存され、全体で1個の引数としてそのまま解釈されていることがわかります。ここでは、「one」と「two」の間だけではなく、「two」の後ろにもスペースが付

いていることに注意してください。

その次のように、スペースを含んだ引数をいったんechoコマンドで出力し、その出力をコマンド置換で取り込むと、コマンド置換で取り込む際にスペースが区切り文字として解釈されてしまうため、argcheckの結果ではスペースが落ち、2個の引数に分かれて解釈されてしまうことがわかります。

そこで、最後の例のように、コマンド置換全体をダブルクォート(" ")で囲むと、シェルの解釈が避けられ、スペースを含めて全体で1個の引数として解釈されるようになります。

リストA argcheck

```
#!/bin/sh

for arg in "$@"  ……すべての引数についてfor文でループする
do  ……for文の始まり
  echo \'"$arg"\'  ……各引数をそのまま、シングルクォートをつけて表示する
done  ……for文の終わり
```

図A argcheckの使用例

```
$ argcheck -f "$HOME"          適当な引数を付けて試してみる
'-f'                           第1引数はそのまま-fになる
'/home/guest'                  第2引数は"$HOME"が展開されたものになる
$ argcheck 'one two '          スペースを含む文字列を、1個の引数として与えてみる
'one two '                     そのまま全体が1個の引数として正しく解釈される
$ argcheck `echo 'one two '`   echoして、コマンド置換で取り込んでみる
'one'                          コマンド置換の結果が解釈され、スペースが削除され、
'two'                          2つの引数に分かれてしまう
$ argcheck "`echo 'one two '`" コマンド置換全体をダブルクォートで囲む
'one two '                     スペースを含め、全体が1個の引数として解釈される
```

参照

特殊パラメータ"$@"(p.167)

標準出力／標準エラー出力の出力先をチェックする

Linux (bash)
FreeBSD (sh)
Solaris (sh)

使い方 echocheck

例 `echocheck 2> file` ……標準エラー出力のみfileに書き込むテスト

基本事項

標準出力と標準エラー出力の出力状況をチェックする「echocheck」というシェルスクリプトを作成します。

解説

シェルスクリプト上で各種リダイレクトを行う場合、その標準出力や標準エラー出力が意図通りにリダイレクトされているかどうか、簡単にチェックしたいことがあります。

そこで、**リストA**の「echocheck」というシェルスクリプトを作成しておき、この「echocheck」に対してリダイレクトを行えば、実際に標準出力と標準エラー出力の出力先を確認することができます。この「echocheck」は、**"$HOME"/bin** または **/usr/local/bin** などの実行パスの通ったディレクトリにインストールして使用します。

●標準出力／標準エラー出力の出力先をチェックするしくみ

echocheckは、単にechoコマンドを2つ並べて、それぞれ標準出力と標準エラー出力にメッセージを出力するだけの単純なシェルスクリプトです。標準エラー出力への出力には、1>&2というリダイレクトを利用しています[注1]。

リストA echocheck

```
#!/bin/sh

echo 'This is a stdout' ……標準出力にメッセージを出力
echo 'This is a stderr' 1>&2 ……標準エラー出力にメッセージを出力
```

注1　ファイル記述子を使ったリダイレクトの項(p.248)も合わせて参照してください。

echocheckの使用例

このechocheckの使用例を**図A**に示します。

まず、何もリダイレクトせずに単純にechocheckを実行すると、標準出力のメッセージと、標準エラー出力のメッセージの両方が画面に表示されます。echocheckの標準出力または標準エラー出力を、**/dev/null** または適当なファイルにリダイレクトすると、リダイレクトした出力は画面には表示されず、リダイレクトしていないほうの出力のみ、画面に表示されることがわかります。

さらに、標準出力と標準エラー出力の両方をリダイレクトする場合、2>&1の記述は> /dev/nullよりも右側になければなりませんが、このことをechocheckを使って実際に確認することができます。そのほか、グループコマンドの{ }や、パイプを使用する場合の動作についても確認できます。

図A echocheckの使用例

```
$ echocheck                          まず単純にechocheckを実行すると
This is a stdout                     標準出力のメッセージが表示される
This is a stderr                     標準エラー出力のメッセージも表示される
$ echocheck > /dev/null              標準出力を/dev/nullにリダイレクトして捨てると
This is a stderr                     標準エラー出力のみ表示される
$ echocheck 2> /dev/null             標準エラー出力を/dev/nullにリダイレクトすると
This is a stdout                     標準出力のみ表示される
$ echocheck > /dev/null 2>&1         標準出力と標準エラー出力の両方をリダイレクトすると、何も表示されない

$ echocheck 2>&1 > /dev/null         2>&1の記述位置を変えると
This is a stderr                     正しくリダイレクトされず、標準エラー出力が画面に表示される

$ { echocheck 2>&1; } > /dev/null    グループコマンドを使ってリダイレクトしてもよい
$ echocheck 2>&1 | cat > /dev/null   標準出力と標準エラー出力の両方をパイプに通すこともできる
```

参照

echo(p.137)　　ファイル記述子を使ったリダイレクト(p.248)

.tar.gz/.tar.bz2自動展開

Linux (bash) / FreeBSD (sh) / Solaris (sh)

Solarisではgnu tarをインストールする必要がある

使い方 extract [圧縮アーカイブ名] [ディレクトリ名]

例

```
extract file.tar.gz ……file.tar.gzが/tmpに展開される
```

基本事項

.tar.gzまたは**.tar.bz2**形式の圧縮アーカイブを、引数に指定するだけで自動的に展開する「extract」というシェルスクリプトを作成します。

解説

ファイルの圧縮アーカイブ形式としては、UNIX系OSでは**tar** + **gzip**(拡張子**.tar.gz**)または**tar** + **bzip2**(拡張子**.tar.bz2**)がよく使われます。さらに、古いものでは、**tar** + **compress**(拡張子**.tar.Z**)が使われていることもあります。

これらの圧縮アーカイブはGNU tarで展開できますが、この時、**.tar.gz**(または**.tar.Z**)ならばtar zxvf、**.tar.bz2**ならばtar jxvfと、オプションのzとjを使い分ける必要があります。

さらに、展開先のディレクトリは-Cオプションで指定しますが、無指定時にカレントディレクトリに展開するのではなく、どこか適当なディレクトリ(たとえば**/tmp**)に展開したほうが便利な場合があります。そこで、このような「extract」という名前のシェルスクリプトを作成してみましょう。

extractシェルスクリプト

シェルスクリプト「extract」は**リストA**のとおりです。extractでは、単にextract *file*.tar.gzと実行すると、「*file*.tar.gz」の内容が**/tmp**以下に展開されます。「*file*.tar.bz2」についても同じです。extractに第2引数を付け、extract *file*.tar.gz */some/dir*とすると、**/tmp**の代わりに「*/some/dir*」に展開されます。

リストAでは、まずシェル変数dirに、パラメータ展開を使って第2引数または省略時には/tmpという値を代入しています[注2]。

そのあと、第1引数の拡張子をcase文で場合分けし、それぞれtar zxvfかtar jxvfかの適切なオプションでtarコマンドが実行されるようにしています。この時、先のdirの値を-Cオプションで指定し、展開先ディレクトリを指定しています。

なお、**.tar.Z**のcompress形式のファイルもgzipで展開できるため、case文のパターンに含めて**.tar.gz**と同じ処理を行うようにしています。そのほか、**.tgz**(**.tar.gz**の省略形)や**.tbz**(**.tar.bz2**の省略形)に対応する場合は、それぞれcase文のパターンに|で区切って追加してください。

注2 「${パラメータ:-値}と${パラメータ-値}」(p.187)も合わせて参照してください。

リストA extract

```
#!/bin/sh

dir=${2-/tmp} ……引数2で指定の展開先ディレクトリを
                  シェル変数dirに代入(省略時は/tmp)
case $1 in ……引数1で指定の圧縮アーカイブファイルの
               拡張子によって場合分け
  *.tar.gz|*.tar.Z) ……拡張子が.tar.gzまたは*.tar.Zだった場合
    tar zxvf "$1" -C "$dir" ……tar zxvfで展開(gzip使用)
    ;; ……このパターンのリストの終了
  *.tar.bz2) ……拡張子が.tar.bz2だった場合
    tar jxvf "$1" -C "$dir" ……tar jxvfで展開(bzip2使用)
    ;; ……このパターンのリストの終了
  *) ……そのほかの拡張子だった場合
    echo "$1"'の展開方法が不明です' 1>&2 ……エラーメッセージを出力
    exit 1 ……エラーで終了
    ;; ……このパターンのリストの終了
esac ……case文の終了
```

Memo

- リストAのcase文中にunzipやlhaコマンドなどを追加して、さらにほかの圧縮アーカイブ形式に対応することもできます。

参照

${パラメータ:-値}と${パラメータ-値}(p.187)

Shift_JIS➡EUC-JP一括変換

Linux (bash)
FreeBSD (sh)
Solaris (sh)

使い方 **sjistoeuc** [ディレクトリ名]

例

```
sjistoeuc /some/dir ……/some/dir内の拡張子.txtのShift_JISファイルが変換される
```

基本事項

カレントディレクトリまたは引数で指定したディレクトリ上の、Shift_JIS(シフトJIS)で書かれたファイルをEUC-JPに一括変換する「sjistoeuc」というシェルスクリプトを作成します。

解説

リストAは、引数で指定したディレクトリ上に存在する、拡張子**.txt**のファイルをShift_JISで書かれたファイルであるとみなし、これをEUC-JPに変換して、拡張子を**.euc**に変更したファイルに出力するシェルスクリプトです。引数を省略するとカレントディレクトリを指定したものとみなされます。

リストAではまずcdコマンドとパラメータ展開を使い、引数($1)が指定されていればそのディレクトリに、指定されていなければカレントディレクトリ(.)に移動しています[注3]。

その後、for文によるループで、拡張子**.txt**のファイルすべてについて、文字コード変換フィルタコマンドであるnkfを実行しています。nkfには-Sedというオプションを付け、Shift_JIS➡EUC-JPという変換を行う(-Seオプション)と同時に、改行コードを**CR**＋**LF**から**LF**のみに変換(-dオプション)しています。

出力のファイル名は、basenameコマンドを使って、元のファイル名から**.txt**を削除したあとに**.euc**を付加したものを、シェル変数outfileに代入して用いています。

これで、Shift_JISで書かれた拡張子**.txt**のファイルのあるディレクトリに移動してsjistoeucを実行するか、または「sjistoeuc [ディレクトリ名]」と指定して実行すると、同じディレクトリに、EUC-JPに変換された拡張子**.euc**のファイルができるはずです。

リストA sjistoeuc

```
cd "${1-.}" ……引数で指定のディレクトリに移動(引数省略時はカレントディレクトリ)
for file in *.txt ……このディレクトリ上の拡張子.txtのファイルについてループ
do ……ループの開始
  outfile=`basename "$file" .txt`.euc ……拡張子.txtを.eucに変更したファイル名をoutfileに代入
  nkf -Sed "$file" > "$outfile" ……nkfで文字コードを変換し、拡張子.eucのファイルに出力
done ……ループの終了
```

注3 「${パラメータ:-値}と${パラメータ-値}」(p.187)も合わせて参照してください。

Memo

●リストAのnkfコマンドを別のフィルタコマンドに変更し、拡張子の指定も変更して、文字コード変換以外の一括変換(たとえば画像ファイルのフォーマット変換など)のシェルスクリプトに応用できます。

Column

文字コードの変換を行うフィルタコマンド

日本語を表現する文字コードとしては、「ISO-2022-JP」(JIS)、「EUC-JP」、「Shift_JIS」、「UTF-8」などが用いられており、状況により、適宜文字コードを変換することが必要になります。

文字コードの変換のためのフィルタコマンドには、本文で用いたnkf以外にも、iconvやその他のコマンドが存在します。nkfおよびiconvを使う場合のおもなオプションを**表a**にまとめておきます。

表a nkfとiconvのおもなオプション

コマンド	動作
nkf -Se	Shift_JISをEUC-JPに変換
iconv -f Shift_JIS -t EUC-JP	Shift_JISをEUC-JPに変換
nkf -Sed	Shift_JISをEUC-JPに変換し、改行コードもCR＋LFからLFのみに変換
nkf -Es	EUC-JPをShift_JISに変換
iconv -f EUC-JP -t Shift_JIS	EUC-JPをShift_JISに変換
nkf -Esc	EUC-JPをShift_JISに変換し、改行コードもLFのみからCR＋LFに変換
nkf -Je	ISO-2022-JPをEUC-JPに変換
iconv -f ISO-2022-JP -t EUC-JP	ISO-2022-JPをEUC-JPに変換
nkf -Ej	EUC-JPをISO-2022-JPに変換
iconv -f EUC-JP -t ISO-2022-JP	EUC-JPをISO-2022-JPに変換
nkf -We	UTF-8をEUC-JPに変換
iconv -f UTF-8 -t EUC-JP	UTF-8をEUC-JPに変換
nkf -Ew	EUC-JPをUTF-8に変換
iconv -f EUC-JP -t UTF-8	EUC-JPをUTF-8に変換

参照

${パラメータ:-値}と${パラメータ-値}(p.187)　basename(p.263)

EUC-JPの文字一覧出力

Linux (bash)
FreeBSD (sh)
Solaris (sh)

FreeBSDやSolarisでは、GNU coreutilsのprintfを使う必要がある

使い方 kanji

基本事項

EUC-JPの文字を一覧出力する「kanji」というシェルスクリプトを作成します。

解説

2バイトのEUC-JPの文字すべてを、その文字コードとともに一覧出力するコマンドがあると便利です。そのようなコマンドはC言語を使えば比較的簡単に記述できますが、ここでは、シェルスクリプトを使ってEUC-JPの文字を一覧出力するコマンドを作成してみることにします。

EUC-JPの文字一覧出力のしくみ

「kanji」シェルスクリプトは**リストA**のとおりです。このシェルスクリプトでは、2バイトの文字コードを16進数4桁で表し、それぞれの桁を上から順にd3、d2、d1、d0というシェル変数に保持し、これら4つのシェル変数を使ってfor文を4重にネスティングしてループしています。

EUC-JPの文字コードは「a1a1」から「fefe」まであり、上位バイト／下位バイトとも、「a0」と「ff」は除かれるため、case文を使って「a0」と「ff」の場合分けを行っています。

実際に表示する文字は、そのEUC-JPの文字コードを、printfの'\xab'の形式の16進表記でシェル変数に蓄積し、一定分まとめてprintfコマンドで表示します。具体的には、シェル変数lineに1行分を、blockに上位バイトが同じ文字の分を、groupに16進、最上位桁が同じ文字の分をまとめ、このgroupごとにprintfを実行します。こうすることにより、毎回printfを呼び出す必要がなくなり、動作が高速化されます。

なお、FreeBSDやSolarisのprintfコマンドでは、'\xab'の形式の16進表記が使えないため、GNU coreutilsに含まれるprintfをインストールして使用する必要があります。

リストA kanji

```
#!/bin/sh

for d3 in a b c d e f ……16進、最上位桁のa～fまでのループ
do ……ループの開始
  group= ……16進、最上位桁が同じ文字を蓄積するためのgroupを空文字列で初期化
  for d2 in 0 1 2 3 4 5 6 7 8 9 a b c d e f ……16進、上から2番目の桁のループ
  do ……ループの開始
    case $d3$d2 in ……16進上位2桁（上位バイト）で条件判断
      a0|ff) ……上位バイトがa0またはffの場合
        continue;; ……この部分には文字は存在しないため、ループを飛ばす
    esac ……case文の終了
    block= ……上位バイトが同じ文字を蓄積するためのblockを空文字列で初期化
    for d1 in a b c d e f ……16進、上から3番目の桁のa～fまでのループ
    do ……ループの開始
      line=$d3$d2$d1'0 ' ……現在の16進文字コード(最下位桁は0)とスペースをlineに代入
      for d0 in 0 1 2 3 4 5 6 7 8 9 a b c d e f ……16進、最下位桁のループ
      do ……ループの開始
        case $d1$d0 in ……16進下位2桁（下位バイト）で条件判断
          a0|ff) ……下位バイトがa0またはffの場合
            line=$line'  ';; ……文字は存在しないため、代わりに半角スペース2つをlineに追加
          *) ……下位バイトがそれ以外の場合
            line=$line\\x$d3$d2\\x$d1$d0;; ……lineに、\xabの記法で2バイト文字コードを追加
        esac ……case文の終了
      done ……16進、最下位桁のループの終了
      block=$block$line\\n ……lineに改行をつけて、blockに蓄積
    done ……16進、上から3番目の桁のループの終了
    group=$group$block ……上位バイトが同じ文字が蓄積されたblockを、groupに蓄積
  done ……16進、上から2番目の桁のループの終了
  printf "$group" ……16進、最上位桁が同じ文字が蓄積されたgroupを表示
done ……16進、最上位桁のループの終了
```

kanjiの実行

作成したシェルスクリプト「kanji」を実行している様子を**図A**に示します。出力行数が多くなるため、図のようにlessコマンドなどにパイプでつないで表示するとよいでしょう。実行すると、全角記号、全角英数字、かな、漢字などが、そのEUC-JPの文字コードとともに表示されるはずです。

図A kanjiの実行

```
$ ./kanji | less                                    lessにパイプでつないでkanjiを実行
a1a0     、。、。・：；？！゛゜´｀¨                  以下のように文字一覧が表示される
a1b0 ^ ￣＿ヽヾゝゞ〃仝々〆〇ー―‐／
a1c0 ＼～∥｜…‥ ‘ ’ “ ” （） 〔〕 ［］
a1d0 ｛｝ 〈〉 《》 「」 『』 【】 ＋－±×
a1e0 ÷＝≠＜＞≦≧∞∴♂♀° ′ ″ ℃￥
  <中略>
a3b0 ０１２３４５６７８９
a3c0  ＡＢＣＤＥＦＧＨＩＪＫＬＭＮＯ
a3d0 ＰＱＲＳＴＵＶＷＸＹＺ
a3e0  ａｂｃｄｅｆｇｈｉｊｋｌｍｎｏ
a3f0 ｐｑｒｓｔｕｖｗｘｙｚ
a4a0  ぁあぃいぅうぇえぉおかがきぎく
  <中略>
b0a0  亜唖娃阿哀愛挨姶逢葵茜穐悪握渥
b0b0 旭葦芦鯵梓圧斡扱宛姐虻飴絢綾鮎或
b0c0 粟袷安庵按暗案闇鞍杏以伊位依偉囲
  <中略>
f3e0 黴靨黶黼黻黼黽鼇鼈皷鼕鼠鼬鼾齊齒
f3f0 龕齡齟齠齡齦齧齬齪齷齲齶龕龜龠
f4a0  堯槇遙瑤
  <以下略>
```

参照

for文(p.55)　case文(p.49)　printf(p.143)

プログレスバー

Linux (bash)
FreeBSD (sh)
Solaris (sh)

使い方 progressbar

基本事項

矢印とパーセント値によるプログレスバーを表示する「progressbar」というシェルスクリプトを作成します。

解説

シェルスクリプトで何らかの作業を実行中に、その作業の進行状況をプログレスバーで表示したいことがあります。そこで、シェルスクリプトによるプログレスバーを作成してみましょう。

なお、一般にカーソル移動をともなう画面表示では、カーソル制御用のエスケープシーケンスを用いる必要がありますが、プログレスバーの表示を1行で行う場合は、カーソルを左端に戻すCR(*Carriage Return*)[注4]を出力して、1行分の文字列を書き換えることにより、カーソル制御用のエスケープシーケンスを使わなくてもプログレスバーの表示を行えます。

progressbarシェルスクリプト

プログレスバーを表示するシェルスクリプト「progressbar」は**リストA**のとおりです。

ここでは、0～100までのパーセント値を引数とし、その値に応じてプログレスバーを表示する「print_bar」というシェル関数を定義し、このprint_barをデモ用のメインルーチンのwhile文によるループから1秒ごとに呼び出しています。

なお、「progressbar」の最初の部分のcase文は、Solarisのsh内部のechoコマンドの仕様が異なり、echoコマンドが常に-eオプション付きと同様の動作をし、かつ-eをオプションとは認識しないことに対応するためのものです。ここでは、echo -eを実際に実行し、その結果、普通の文字として-eが表示された場合はechoの-eオプションを付けないようにしています。シェル変数**ECHO**には、判定結果としてecho -eまたはechoのどちらかが代入された状態になります。

シェル関数print_barは、その引数で指定されたパーセント値に応じて、プログレスバーの矢印の長さなどを計算し、表示のための文字列を作成します。ここでは、矢印の長さに応じた表示文字列の作成のためのwhile文のループで、位置パラメータの個数を表す特殊パラメータである$#をカウンタ代わりに使用していることに注意してください。具体的には、適当な文字列である「dummy」を使ってset dummyとすると$#が「1」にセットされ、set - "$@" dummyとすると$#の値に「1」が加算されます。こうすることにより、外部コマンドのexprの呼び出しが不要になるぶん、高速化できます。なお、位置パラメータの内容は書き換えられてしまうため、シェル関数の冒頭で$1の値をpercentに代入して退避していることにも注意してください。

注4　echo -eコマンドでは'\r'で出力可。

リストA progressbar

```
#!/bin/sh

ECHO='echo -e' ……echoコマンドには-eオプションを付けるのを標準とする
case `$ECHO` in ……Solarisにも対応するため、echo -eの実行結果をテスト
  -e) ……結果、-eが文字列として表示されてしまった場合
    ECHO=echo;; ……常に-eオプションが有効なechoコマンドであるとみなし、-eを付けないようにする
esac ……case文の終了

print_bar() ……パーセント値を引数としてプログレスバーを表示するシェル関数print_barの定義の開始
{
  percent=$1 ……シェル関数の引数をシェル変数percentに代入

  column=`expr 71 \* "$percent" / 100` ……percentの値から矢印の長さを計算しcolumnに代入
  nspace=`expr 71 - "$column"` ……同様に、矢印の右側のスペースの数をnspaceに代入

  bar='\r[' ……プログレスバーのカーソルを左端に戻すリターンコードと[の文字をbarに代入
  set dummy ……シェル関数の位置パラメータの数($#)を1にリセット(カウンタとして流用)
  while [ $# -le "$column" ] ……$#の値がcolumnの値以下であるかぎりループ
  do ……ループの開始
    bar=$bar'=' ……barに=という文字(矢印の棒)を追加
    set - "$@" dummy ……$#の値に1を加算
  done ……ループの終了
  bar=$bar'>' ……barに>という文字(矢印の矢)を追加

  set dummy ……再びシェル関数の位置パラメータの数($#)を1にリセット
  while [ $# -le "$nspace" ] ……$#の値がnspaceの値以下であるかぎりループ
  do ……ループの開始
    bar=$bar' ' ……barにスペースを追加
    set - "$@" dummy ……$#の値に1を加算
  done ……ループの終了
  bar=$bar'] '$percent'%\c' ……barに]の文字と、パーセント数値と、改行抑制エスケープを追加

  $ECHO "$bar" ……barに代入されている1行分のプログレスバーを表示
} ……シェル関数の定義の終了


i=0 ……barに]の文字と、パーセント数値と、改行抑制エスケープを追加
while [ "$i" -le 100 ] ……iの値が100以下であればループ
do ……ループの開始
  print_bar "$i" ……iの値を引数としてシェル関数print_barを呼び出す
  i=`expr "$i" + 1` ……iの値に1を加算
  sleep 1 ……1秒間待つ
done ……ループの終了
echo ……最後に1行改行
```


プログレスバーの表示文字列は、シェル変数barに蓄えておき、1行分の文字列をすべて作成し終わってから、シェル関数の終わりにまとめてechoコマンドで出力するようにしています。文字列には、右向きの矢印(=>)と、数値でのパーセント表示(%)と、改行コード抑制のための'\c'が含まれます。なお、echoコマンドは実際には**$ECHO**と記述し、前述のとおり、echo -eまたはechoが実行されます。

progressbarの実行

実際にこのprogressbarを実行している様子を**図A**に示します。このように、1行で矢印とパーセント値が表示されます。パーセント値は1秒に1%ずつ増加し、それにともない、矢印が左端から右端まで伸びて行きます。100%になると終了です。

なお、実際のシェルスクリプトでは、progressbarのシェルスクリプトのメインルーチン部分を実際の作業を行うコマンドに置き換え、作業の進行状況に応じて適宜シェル関数print_barを呼び出してやれば、プログレスバーの表示が行えます。

図A progressbarの実行

```
$ ./progressbar
[==================================================>                ] 75%
```

参照

シェル関数(p.76)　echo(p.137)　特殊パラメータ$#(p.171)　set(p.120)