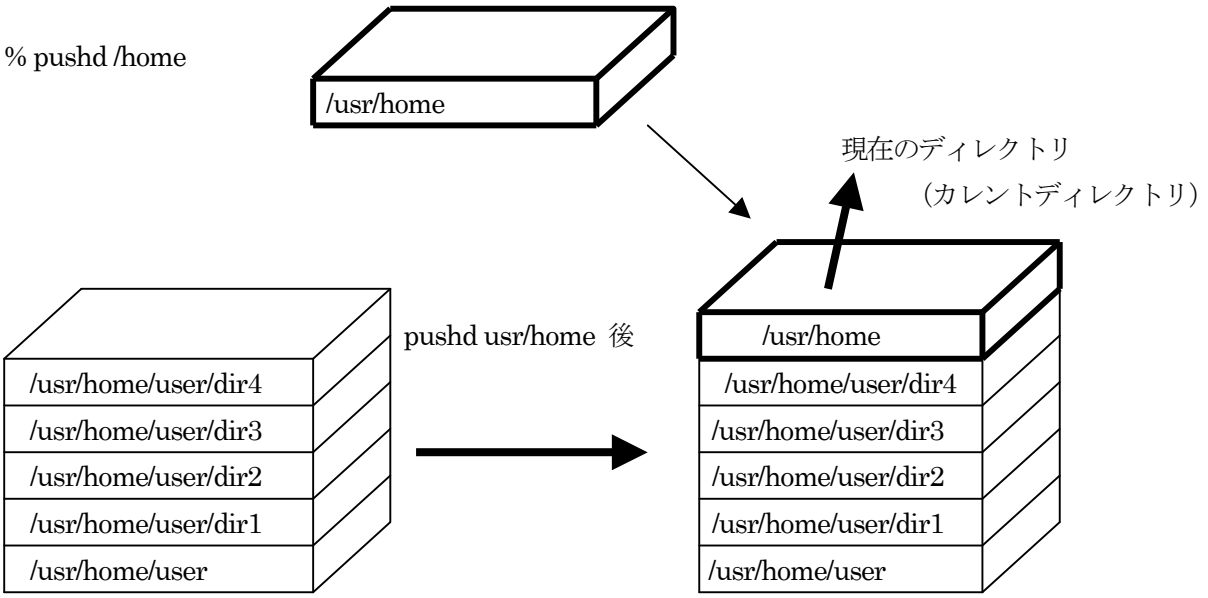
ディレクトリを異動する時に、わざわざ　パスを入力するのは、大変です。
そのような時に、ディレクトリスタックと言う仕組みを使います。

ディレクトリスタックで使うコマンドは下記の３つです。
pushd (push directory)
popd　(pop directory)
dirs　(directory stock)

ディレクトリスタックに積んである。スタックの確認を　dirs コマンドを実行して
~ 表示されて、スタックに~ がスタックに積まれていることを確認します。
現在いる。カレントディレクトリがディレクトリスタックに積まれて行く事を確認します。
pushd コマンドを引数無しで実行します。
pushd: No other directory　と表示されます。
次に、mkdir コマンドで　ディレクトリ　/usr/home/user/dir1, dir2, dir3, dir4　を作ります。
そして、pushd コマンドで、ディレクトリスタックを積みます。

```
% pwd
/usr/home/user
% dirs
~/usr/home/user
% pushd
pushd: No other directory
%mkdir dir1 dir2 dir3 dir4
% pushd /usr/home/user/dir1
~ /dir1 ~
% pwd
/usr/home/user/dir1
% pushd /usr/home/user/dir2
~ /dir2 ~ /dir1~
% pwd
/usr/home/user/dir2
% pushd /usr/home/user/dir3
~/dir3 ~ /dir2 ~ /dir1 ~
% pwd
/usr/home/user/dir3
% pushd /usr/home/user/dir4
~ /dir4 ~/dir3 ~ /dir2 ~ /dir1 ~
% pwd
/usr/home/user/dir4
%
```

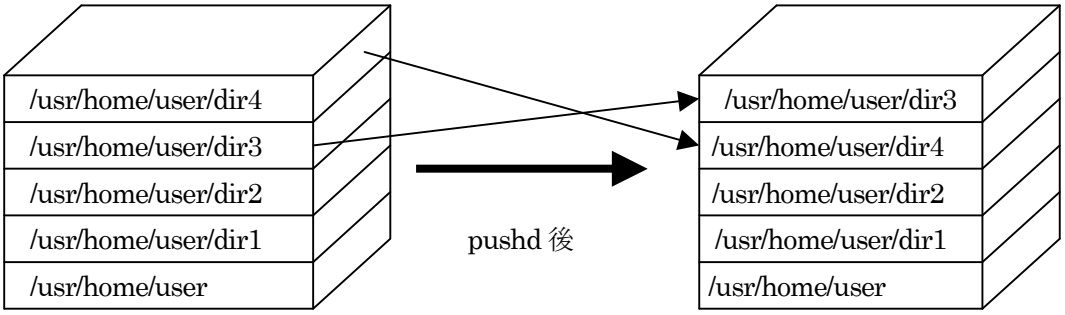
先程のように push コマンドを使うと、ディレクトリが積まれていきます。
そして、カレントディレクトリが、push コマンドで指定したディレクトリに変わります。
dirs コマンドで、下記の図のように一番上のディレクトリがカレントディレクトリです。



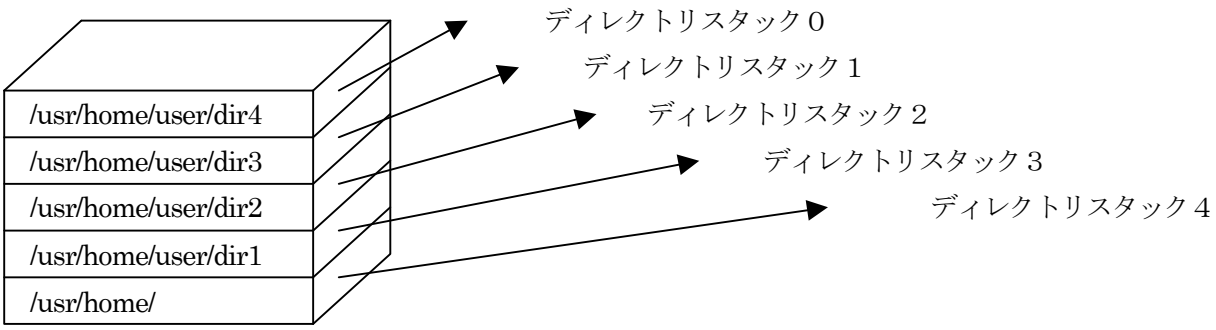

スタック：後入れ先出し方式でデータを操作する際に利用されるメモリー領域のこと。
データを入れるプッシュ命令とスタックからデータを取り出すポップ命令を用いて行う。

次は、dirs コマンドで現在のスタックに積まれているディレクトリをみます。
pushd コマンドで引数なしを実行します。
そして、dirs コマンドで、スタックに積まれているディレクトリをみます。
スタックに積まれているディレクトリが、一番上と上から２番目が入れ替わっている事が、確認できます。
もう一度 pushd コマンドで引数なしを実行すると、一番上と上から２番目が入れ替わり、元にもどります。

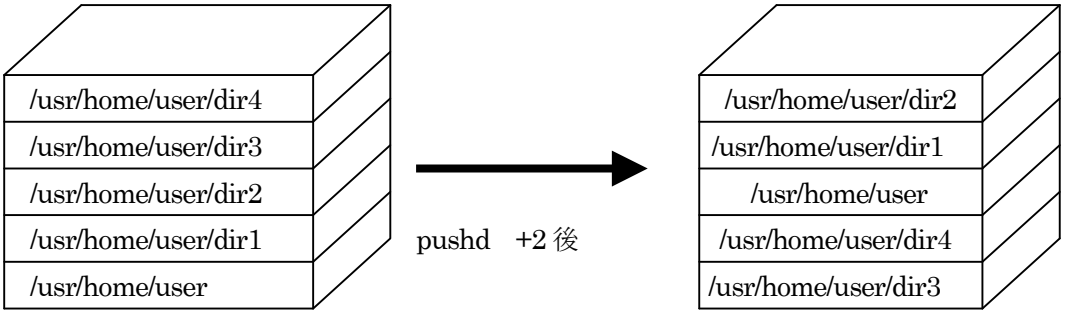
```
% dirs
~ /dir4 ~/dir3 ~ /dir2 ~ /dir1 ~
% pushd
~ /dir3 ~/dir4~ /dir2 ~ /dir1 ~
% dirs
~ /dir3 ~/dir4 ~ /dir2 ~ /dir1 ~
% pushd
~ /dir4 ~/dir3 ~ /dir2 ~ /dir1 ~
```



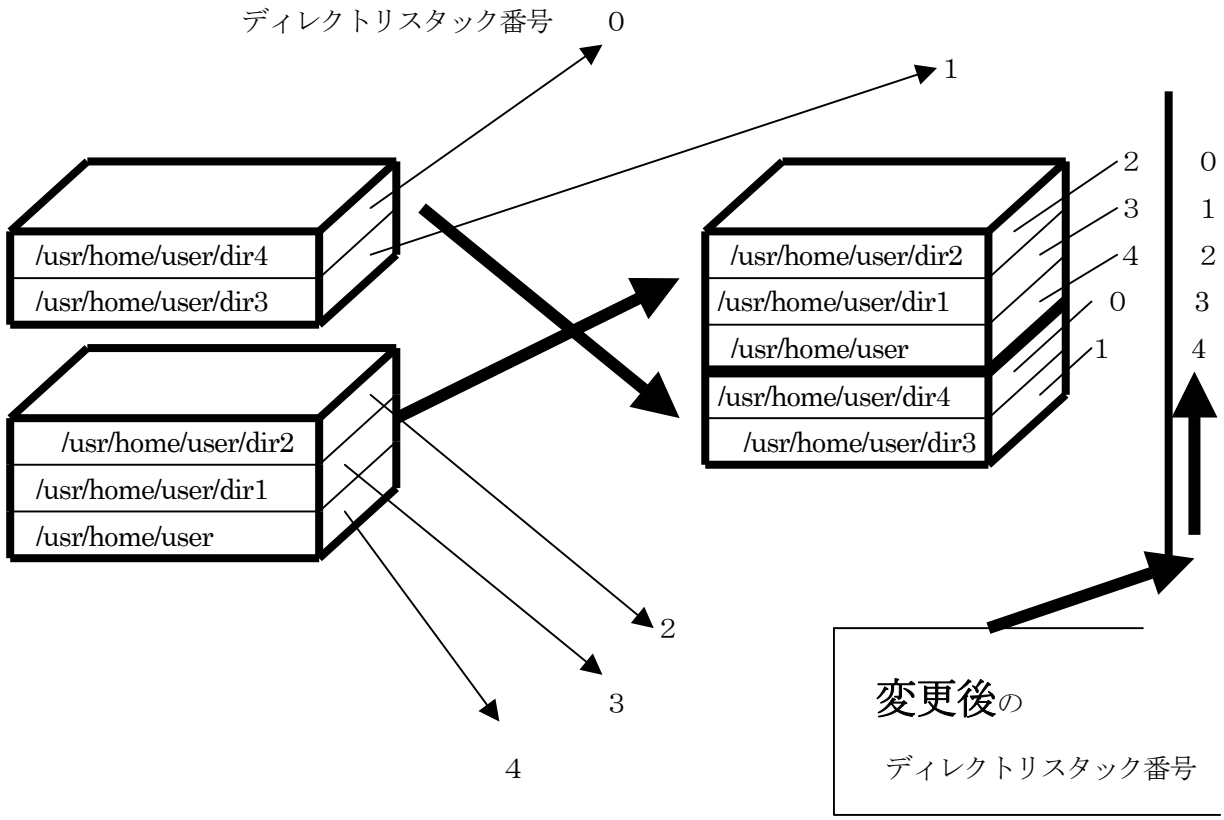

ディレクトリスタック番号
カレントディレクトリ（下記の図のように一番上のディレクトリ）を０番として　下に行くごとに、＋１増やして数えます。




pushd +ディレクトリスタック番号　を実行すると、指定された番号のディレクトリが、カレントディレクトリになります。



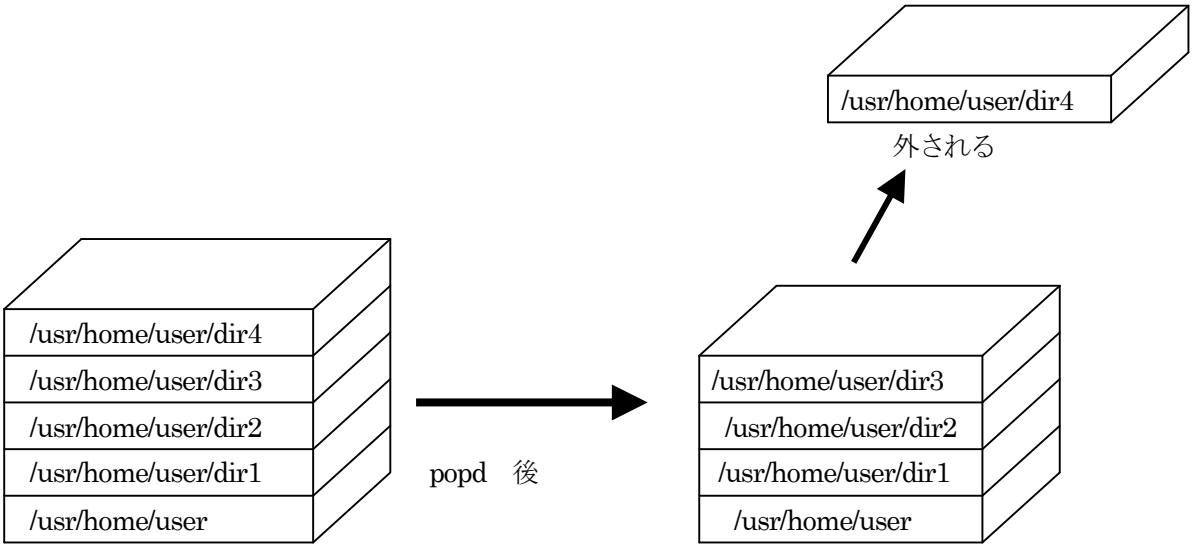
先程の図を説明しますと、




pushd +ディレクトリスタック番号で
pushd ＋2では、ディレクトリスタック番号２より小さい番号（ディレクトリスタック番号０、１）の、ブロックが、 指定された、ディレクトリスタック番号２以上のブロック（ディレクトリスタック番号２，３、４）の下に移動します。

```
% dirs
~ /dir4 ~/dir3 ~ /dir2 ~ /dir1 ~
% pushd +2
~ /dir2 ~/dir1 ~ ~ /dir4 ~/dir3 ~
% pushd +3
~/dir3 ~ /dir2 ~ /dir1 ~ ~ /dir4
% pushd +4
~ /dir4 ~/dir3 ~ /dir2 ~ /dir1 ~
%
```

次は、popd コマンドを使います。popd コマンドは、ディレクトリスタックに積まれた、ディレクトリを上から、外していきます。その時に、ディレクトリスタック番号の０番がカレントディレクトリになる。




popd コマンドでディレクトリスタックからディレクトリを１づつ外していきます。

```
% dirs
~ /dir4 ~/dir3 ~ /dir2 ~ /dir1 ~
% pwd
/usr/home/user/dir4
% popd
~/dir3 ~ /dir2 ~ /dir1 ~
% pwd
/usr/home/user/dir3
% popd
~ /dir2 ~ /dir1 ~
% pwd
/usr/home/user/dir 2
% popd
~ /dir1 ~
% pwd
/usr/home/user/dir 1
% popd
~
% pwd
/usr/home/user
%popd
popd: Directory stack empty.
%
```

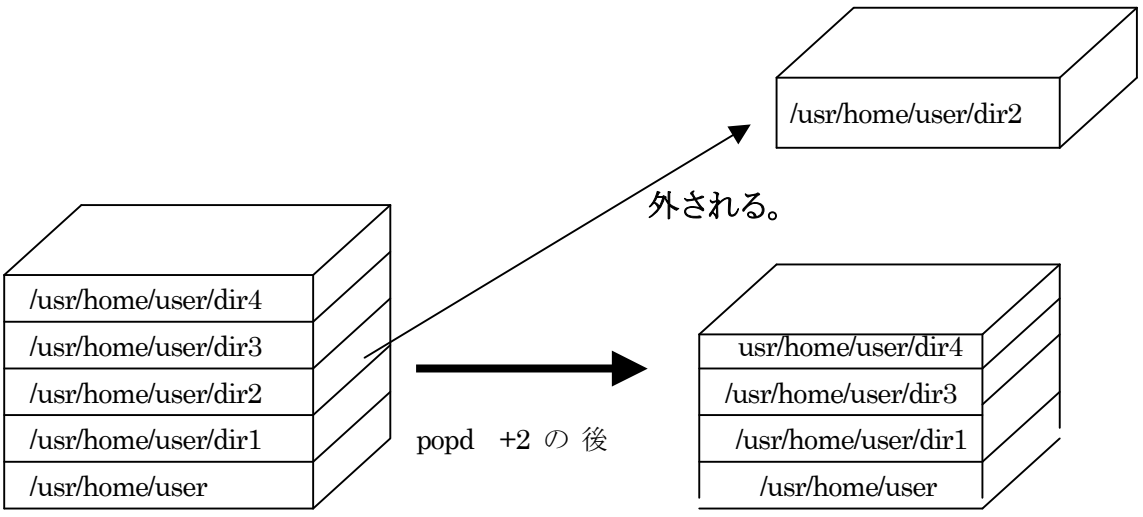
まずは、ディレクトリスタックに、元のとおりに、します。

```
% pwd
/usr/home/user
% dirs
~
% pushd /usr/home/user/dir1
~ /dir1 ~
% pushd /usr/home/user/dir2
~ /dir2 ~ /dir1~
% pushd /usr/home/user/dir3
~/dir3 ~ /dir2 ~ /dir1 ~
% pushd /usr/home/user/dir4
~ /dir4 ~/dir3 ~ /dir2 ~ /dir1 ~
%
```

popd +ディレクトリスタック番号 を実行します。

例えば　popd +2 コマンドを実行します。

指定されたディレクトリスタック番号のディレクトリが外されます。



```
% dirs
~ /dir4 ~/dir3 ~ /dir2 ~ /dir1 ~
% pwd
/usr/home/user/dir4
%
% popd +2
~ /dir4 ~/dir3 ~ /dir1 ~
% popd +1
~ /dir4 ~/dir1 ~
% popd +2
~ /dir4 ~/dir1
% popd +1
~ /dir4
%
```

書式

pushd [ディレクトリ名 | +ディレクトリスタック番号]

表示された、左端から、ディレクトリスタック番号 0, 1,2.....n に、なる
　引数がなければ、ディレクトリスタックの番号０とディレクトリスタックの番号１をに入れ換えます。
　引数なしの pushd は cd のように `pushd カレントディレクトリ' を行います。
(+) ディレクトリ をつけると、現在の作業ディレクトリをディレクトリスタックに積んで
name に移動します。
(+) `+n' として番号をつけると、ディレクトリスタック番号の n 番目のエントリがトップにくるようにスタックを回転します。
　pushd +n' を行うと n 番目のディレクトリが展開されて、スタックのトップに移動されます。
　dirs と同じようにディレクトリスタックの最終的な内容を表示します。

例

カレントディレクトリを/home に移動して、ディレクトリスタック番号０を、/home をする。
% pushd /home

ディレクトリスタック番号２番のディレクトリに移動する。
% pushd +2

書式

popd [+ディレクトリスタック番号]

引数がなければ、ディレクトリスタックの積まれた上のディレクトリスタック０番をはずして、ディレクトリスタック番号１番のディレクトリに移動する。
+ディレクトリスタック番号　のように数値を与えると、ディレクトリスタック番号のエントリを破棄します。
また、すべての形式の popd は dirs のようにディレクトリスタックの最後のエントリを表示します。
表示された、左端から、ディレクトリスタック番号 0, 1,2.....n に、なる

例

ディレクトリスタックの番号０をはずして、ディレクトリスタックの番号１を番号０にする。
% popd

ディレクトリスタック番号３　を外す。
% popd +3

書式

## dirs

スタック先頭のディレクトリは現在のディレクトリになります。
表示された、左端から、ディレクトリスタック番号 0, 1,2.....n に、なる

ディレクトリスタックを表示する。
dirs

ここでは、前のコマンドが成功したら、次のコマンドを実行する　を見ていきます。
その時は、　「&&」を使います。
まずは、ファイル ffand を cat コマンドを使って、つくります。「成功するか？」
ls ffand の後に　「&&」を入れて、そして、cat ffand コマンドを使い、中身をみます。
これは、ls ffand　コマンドが、**成功**して、ffand があれば、cat ffand コマンドで、その中身を見る　と言う事です。
次に、存在しないファイル ffnono を、ls ffnono && cat ffnono を実行して「&&」を使用します。

```
% cat > ffand
seikousuruka ?
% ls ffand && cat ffand
ffand
seikou suruka?
% ls ffnono && cat ffnono
ls : ffnono: No such file or directory
%
```

ls ffnono コマンドが、失敗のため、「&&」の後ろの cat ffnono　のコマンドの部分が実行されずに、
ls : ffnono: No such file or directory　とエラーメッセージが表示されました。

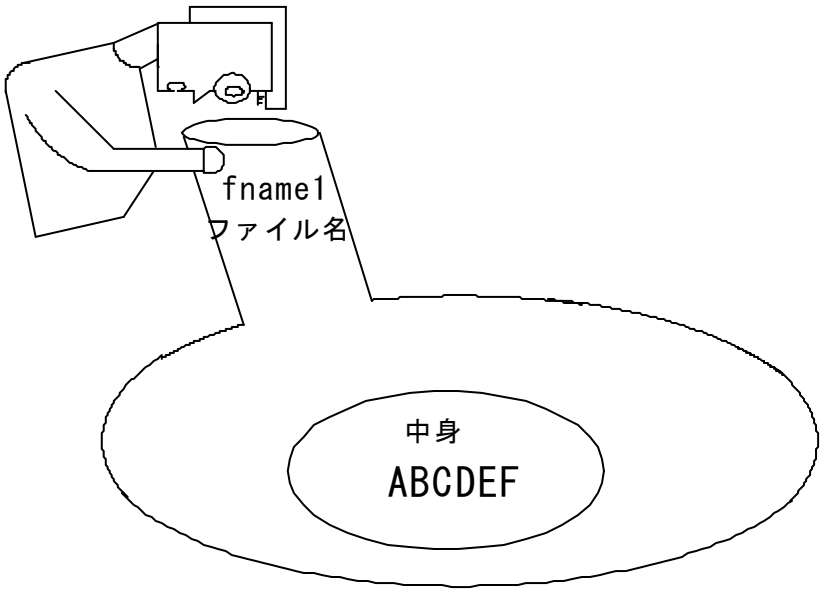
次は、前のコマンドが、失敗したら、次のコマンドを実行する。　を、見ていきます。
その時は、　「| |」を使います。
先程のファイル ffand と　ffnono　を使います。

ls ffand の後に　「| |」を入れて、そして、cat ffand コマンドを使い、中身をみます。
これは、ls ffand　コマンドが、**失敗**して、ffand があれば、cat ffand コマンドで、その中身を見る　と言う事です。
次に、存在しないファイル ffnono を、ls ffnono && cat ffnono を実行して「&&」を使用します。

```
% ls ffand || cat ffand
ffand
% ls ffnono || cat ffand
ls : ffnono: No such file or directory
seikou suruka?
%
```

「| |」より前の　ls ffand コマンドが成功すれば、「| |」の後の cat ffand コマンドは、実行しない。
「| |」より前が失敗すれば、エラーメッセージ ls : ffnono: No such file or directory を出してから、
「| |」より後の cat ffand コマンドが実行されました。

ファイルに別の名前をつけるには、ln コマンドを使います。
ある、ファイルの名前を、別のファイルの名前で、実行したり、参照したり、ファイルの置いてある所ではない、別のディレクトリから、直接実行、参照する時に、ln コマンドを使用します。
cp で、ファイルをコピーして、名前を変えても、良いのですが、限りある、ディスクの資源を、有効利用するためにも、ln コマンドを使い、コピーを行わないで、参照できるようにします。
まず、下記の図は、ファイル fname1 という　名のファイルです。ファイルに書かれている中身は、ABCDEF と書かれています。
ファイル名とは、下記の図の筒の事だと思ってください。その筒（ファイル名 ffname!）からは、中身のＡＢＣＤＥＦがあることが、わかります。

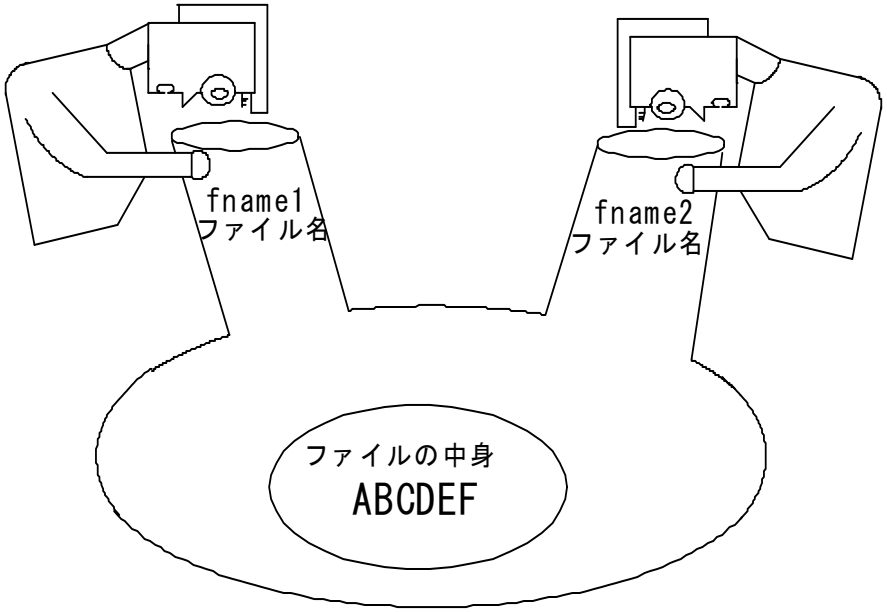



まず、ファイル fname1 を、cat コマンドで作ります。そして、ls –l コマンドをみてみます。

```
% cat > fname1
ABCDEF
% cat fname1
ABCDEF
%   ls –l fname1
-rw-r--r--   1 hiro wheel   7 Oct 19 23:49 fname1
%
```

-rw-r--r--　1 hiro wheel　7 Oct 19 23:49 fname1
　　↑
　　この１に注意をしてください。ハードリンクの数です。

次は、ハードリンクと呼ばれるものを、考えて見たいと思います。
筒の名前がファイルの名前ならば、ln コマンドで、もう一つ、筒を増やして、その筒の名前（ファイル名）を、fname2 とします。
下記の図では、増やされたファイル名 ffname2 からも、ファイル名 ffname1 から見える。同じファイルの中身 ABCDEF が、見えます。



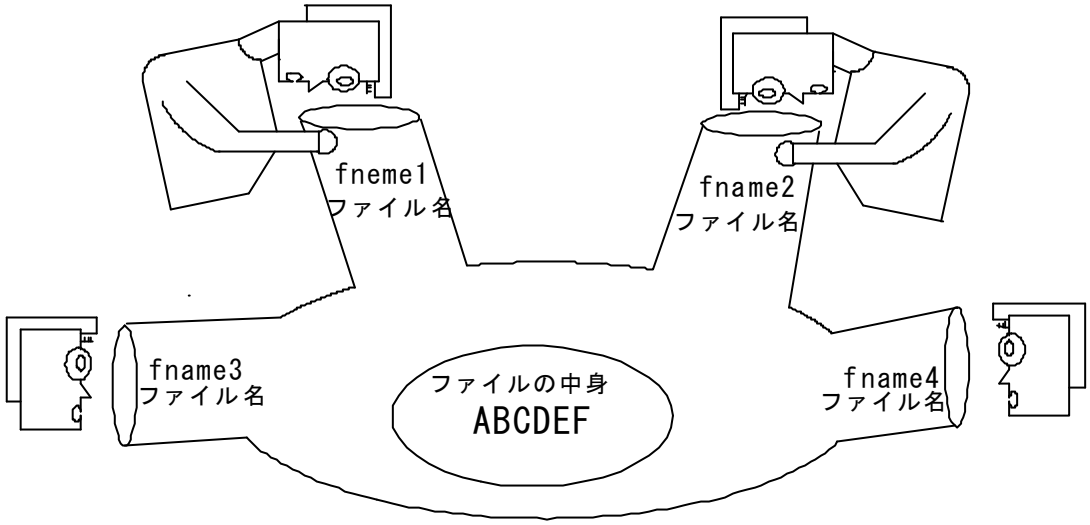

まず、ln コマンドで、ファイル名 fname2 を作ります。そして、cat コマンドで、ファイルの中身を確認します。
つぎに、　ls –l　コマンドで、どのような結果が出るかを、見ていきます。

```
%　ln fname1 fname2
% cat fname2
ABCDEF
% ls –l fname[1-2]
-rw-r--r--　2 hiro wheel　7 Oct 19 23:49 fname1
-rw-r--r--　2 hiro wheel　7 Oct 19 23:49 fname2
%
```

-rw-r--r--　2 hiro wheel　7 Oct 19 23:49 fname1
-rw-r--r--　2 hiro wheel　7 Oct 19 23:49 fname2

先程の　-rw-r--r--　1 hiro wheel　7 Oct 19 23:49 fname2　の 1 と比べてみてください。
先程は、ファイル名が一つしかないので、1　しかし、ln コマンドで、ファイル名が一つ増えたので
同じ共通の中身のファイル名が２つですよ、と教えてくれていますので、fname1,fname2 の、
ハードリンクの数が 2 になる。

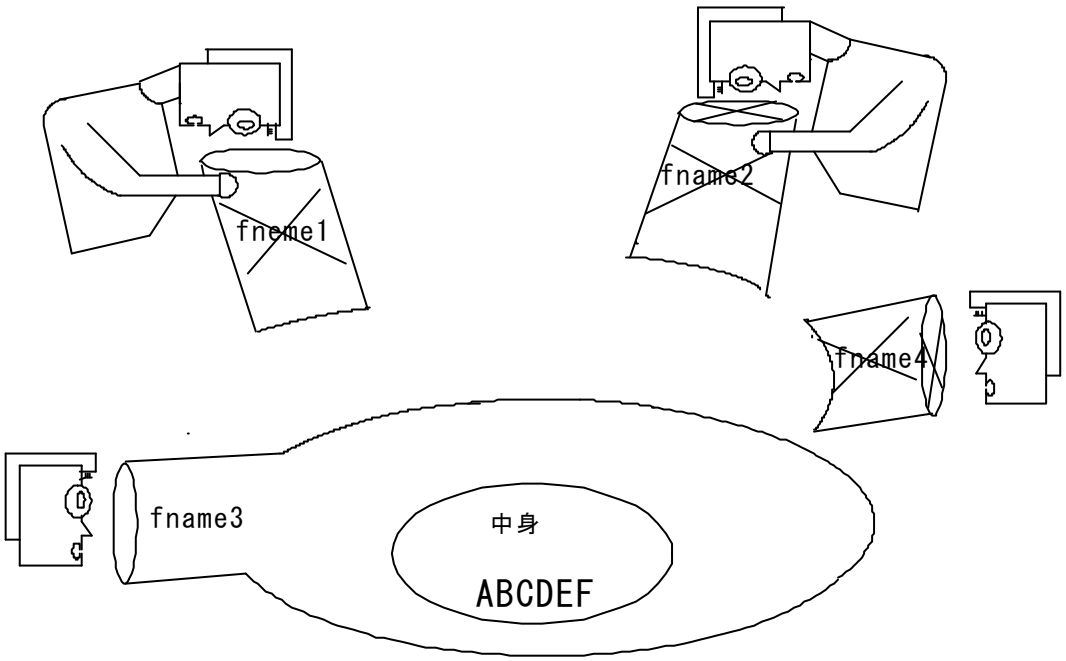
もう２つ増やして、fname2 をオリジナルファイルとしてもう一度みてみましょう。先程の数字が 4 になっている事でしょう。また、cat コマンドで、fname4 だけを、みてみましょう。




```
% ln fname2 fname3
% ln fname2 fname4
% ls –l fname[1-4]
-rw-r--r--   4 hiro wheel   7 Oct 19 23:49 fname1
-rw-r--r--   4 hiro wheel   7 Oct 19 23:49 fname2
-rw-r--r--   4 hiro wheel   7 Oct 19 23:49 fname3
-rw-r--r--   4 hiro wheel   7 Oct 19 23:49 fname4
% cat fname4
ABCDEF
%
```

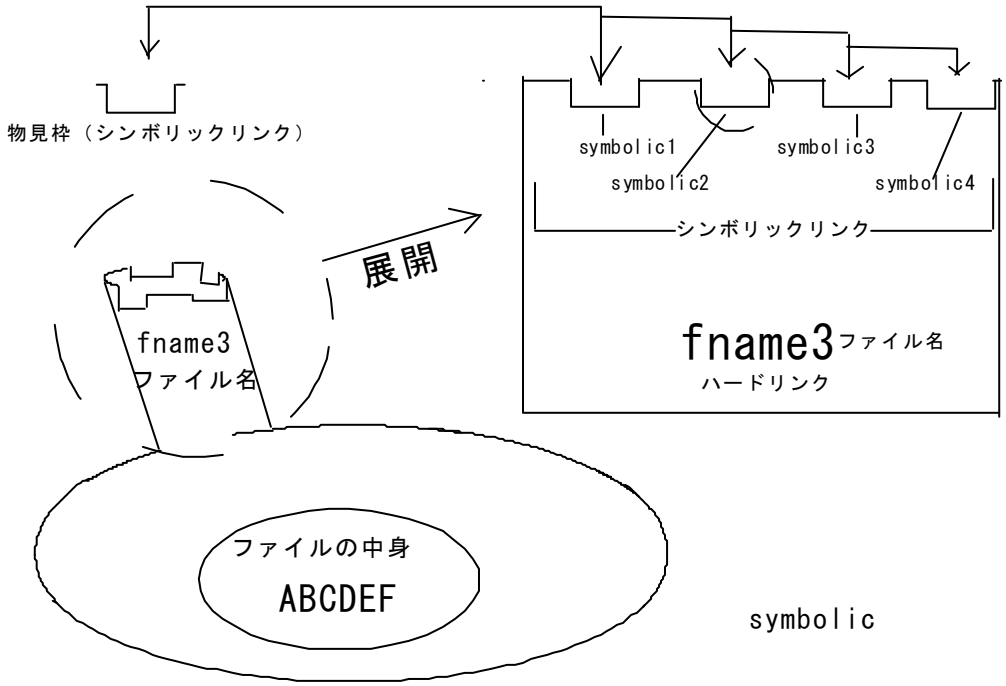
見事にハードリンクの数が 4 になってます。ファイル名が違っても、作られた時間が、全て同じです。また、fname4 の中身も ABCDEF と同じですね。

下記の図のように、ハードリンクは、最初のオリジナルの fname1 が、消去され、さらに他のファイル fname2, fname4 が、消去されても　fname3 があるかぎり、実体の中身 ABCDEF は残っています。
fname3 まで、消してしまえば、ファイルの中身は、なくなってしまいます。




実際に、上図のように、ファイル、fname1, fname2, fname4 を削除してファイル fname3 の中身が存在しているか、みてみましょう。
また、fname3 のハードリンクの数が 1 であることも、注意してください。

```
% ls –l fname[1-4]
-rw-r--r--   4 hiro wheel   7 Oct 19 23:49 fname1
-rw-r--r--   4 hiro wheel   7 Oct 19 23:49 fname2
-rw-r--r--   4 hiro wheel   7 Oct 19 23:49 fname3
-rw-r--r--   4 hiro wheel   7 Oct 19 23:49 fname4
% rm fname[1-2,4]
% ls –l fmame*
-rw-r--r--   1 hiro wheel   7 Oct 19 23:49 fname3
% cat fname3
ABCDEF
%
```




次は、ハードリンクに対して、ソフトリンクです。または、シンボリックリンクです。

ソフトリンク（シンボリックリンク）は、上記の図でいうと、ハードリンクの別名（物見枠の名前みたいなもの）です。
ハードリンクの上でソフトリンクは、成り立っています。上記の図では、ハードリンクのファイル名は、fname3 です。fname3（この図でいえば、筒）の別名です。
fname3 の筒から、ファイルの実体の中身を見るのに、simbolik3 の場所で、見るのか、simbolik４の場所で見るのか？等の違いです。ようは、シンボリックから、ファイルの中身を見る事が出来るのです。

しかし、ハードリンクと違い、上記の図で言えば、リンクする前のオリジナルファイルの fname3 を消してしまうと、ファイルの中身は、見る事ができません。筒を消してしまうのですから、筒の上にあった。物見枠は、当然役には、立たないのです。windows のショートカットと同じです。
ソフトリンクを張るには、ln –s コマンドを使います。まず、ファイル fname3 のソフトリンクを張ります。

ls –F コマンドで、ファイルの後ろに @ が付いている事に、注目してください。これは、ファイルがソフトリンク（シンボリックリンク）である事を、示しています。

そして、最後に作った。symbolic4 を、cat コマンドで中身を、確認します。

次に、オリジナルのファイル fname3 を消します。
そして、symbolic4 を、cat コマンドで中身を、確認します。

```
% ln-s fname3     symbolic1
% ln-s fname3     symbolic2
% ln-s fname3     symbolic3
% ln-s fname3     symbolic4
% ls –F symbolic*
symbolic1@          symbolic2@          symbolic3@          symbolic4@
% cat symbolic4
ABCDEF
% rm fname3
% cat symbolic4
% cat : symbolic4 : No such file or directory
%
```

オリジナルファイル fname3 を消してしまうと、symbolic4 が、無いと言われてしまいました。

次に、新しく、fname3 を、cat コマンドで作って、リンクが戻るのかを、見てみます。

```
% ls symbolic*
symbolic1           symbolic2           symbolic3           symbolic4
% cat > fname3
new
ABC
% cat symbolic4
new
ABC
%
```

しっかりと、リンクが戻っていました。

書式

ln [オプション] オリジナルファイル　リンクファイル名

オプション

-s： ソフトリンク（シンボリックリンク）で、別の名前のファイルを作る。

例

ファイル fname にハードリンクファイルをつくり、ファイル名 fname1 をつくる。
% ln fname fname1

ファイル fz に、に、ソフトリンク（シンボリックリンク）ファイルをつくり、
ファイル名　fzsymbolic をつくる。
% ln –s fz fzsymbolic

---

一休み

二つ以上のコマンドを()で囲んで実行させる事が出来ます。
例えば、カレントディレクトリを、ホームディレクトリにしておいて、
( cd / ; pwd) ; pwc コマンドを実行します。
()で囲んでは、サブシェルで、実行されます。数学の括弧と同じく、最初に実行されます。
()で囲んだところが、処理を終えれば、次に、括弧以外が、処理をされます。

```
% cd
/usr/home/hiro
% ( cd / ; pwd ) ; pwd
/
/usr/home/hiro
%
```

わざわざ、ディレクトリを、移動して戻らなくても、いいです。

ｃシェルの環境変数の設定をしたいとおもいます。

まずは、set コマンドを使います。行数が少し多いので set | more コマンドで、実行します。

```
% set | more
-

addsuffix
argv    ()
csubstnonl
cwd     /home/hiro
dirstack        /home/hiro
echo_style      bsd
edit
filec
gid     0
group   wheel
history 100
home    /home/hiro
killring        30
loginsh
mail    /var/mail/hiro
owd
path    (/sbin/bin /usr/sbin /usr/bin /usr/games /usr/local/sbin /usr/local/bin
 /usr/X11R6/bin /home/hiro/bin)
prompt  %
prompt2 %R?
prompt3 Correct>%R (y|n|e|a)?
savehist        100
shell   /bin/csh
shlvl   1
status  0
tcsh    6.14.00
term    cons25
tty     ttyv0
uid     1001
user    hiro
version tcsh 6.14.00 (Astron) 2005-03-25 (i386-intel-FreeBSD) options wide,nls,
dl,al,kan,rh,color,filec
%
```

環境変数:アプリケーション（OS 上で動作するタスク）が共通に提供される機能で、OS から、外部からデータを受け取りタスクの動作・設定を変更する。

変数には、シェル変数と環境変数があります。シェル変数は、変数をチューニングしたシェルにたいして、有効に働きますが、環境変数は、そのシェルから実行されるプロセスの動作・設定を変更する。
動作・設定を変更する為の、変数の事を、組み込み変数と呼ぶ。

**********

組み込み変数（シェル変数）

addsuffix :ファイル名またはディレクトリ名、が補完の時に一致したら、、ディレクトリの場合には末尾に `/’ を付け、 通常のファイルの場合には末尾にスペースを加える。

argv: シェルスクリプトの引数
例えば$1, $2, $3, ...はシェルの引数に置換され、関数への引数は, シェル変数 argv で与えられ、argv[1], argv[2], argv[3] ...が第一引数、第二引数、...を表し、$argv[1], $argv[2], $argv[3] ...が参照され,$1, $2, $3, ... は略式記号.

csubstnonl:改行とキャリッジリターンはコマンド置換によって空白文字に置き換えられる。デフォルトで設定。

キャリッジリターン : 復帰、復改とも呼ばれる、カーソルを文頭へ戻すことを表す制御コードのこと。
制御コード:ディスプレイやプリンタを制御する特別な文字コード

cwd:カレントディレクトリのフルパス名

dirstack: ディレクトリスタックに積まれているディレクトリ

echo_style :echo 組み込みコマンドのスタイル。

edit:コマンドラインエディタの設定対話型シェルではデフォルトで設定。

filec:デフォルト設定の tcsh では補完は常に行われ、この変数は無視され、edit が未設定である場合、csh の補完が使用され、csh で設定されている場合は、ファイル名の補完が使われる。

gid:ユーザの実グループ ID

group:ユーザのグループ名

history :最初の単語は、記録しておくべきヒストリイベント数。オプションである 2 番目の単語 (+) は、ヒストリがどういう形式で表示されるかを示し,これが与えられていなければ、`%h\t%T\t%R\n' が使われる。フォーマットシーケンスは、prompt 下に記述され、`%R' の意味が変わることの注意がされ、デフォルトでは 100。

`%h\t%T\t%R\n' : %h= %!, ! 現在のヒストリイベント番号です。
\t=タブコード
%T=現在時刻
%n=ユーザ名
\n=改行

home:起動したユーザのホームディレクトリに初期化され、ファイル名での `~ の展開には、この変数が参照される。

killring: 何個のキルされた文字列をメモリ中に保持するかを示し、デフォルトで `30' にセットされる。セットしないか、`2' より小さい値を設定すると、最近キルした文字列をシェルは保持する。

loginsh:シェルがログインシェルである場合に設定されシェル実行中でこの変数を設定したり設定を解除しても何の効力もない

mail: 届けられたメールをチェックするファイルあるいはディレクトリ名。

owd :前の作業ディレクトリで、cd が使う -（ハイフン）および pushd と等価.

path:実行可能なコマンドを探すディレクトリのリスト。null 文字はカレントディレクトリを示す。レントディレクトリを示す。path 変数がない場合、フルパス名での指定で実行される。

prompt :端末からコマンドを読み込む前に表示される文字列

prompt2: while ループや foreach ループの中で、また `\' で終った行の次の行で、プロンプトとして用いられる文字列

prompt3 :自動スペル訂正の確定時のプロンプト文字列。`%R' の意味が変わることに注意。

%R = prompt2 の中ではパーサの状態。prompt3 の中では修正された文字列。history の中では履歴文字列。

savehist:シェルは終了する前に `history -S' を行う最初の単語を数字に設定すると、その個数までの行が保存され履歴リストをファイル名に保存する。
2 番目の単語を `merge'にすると、履歴ファイルが存在する場合に、置換ではなく追加を行いタイムスタンプによってソートを行い、最近のイベントを残す。

shell:シェルのファイル。シェルをフォークして、実行ビットが設定されているがシステムによる実行が不可能なファイルを実行するために用いられる。初期値は シェルの置き場所。
fork :（フォーク）は、UNIX 系システムコールのひとつ、プロセスのコピーを生成するもの

shlvl:入れ子になったシェルの数。ログインシェルでは 1 にリセットされる。

status ： 最後のコマンドによって返された状態。コマンドが異常終了した場合には 0200 が加えられる。
組み込みコマンドは、失敗すると終了状態 `1' を返す。
その他の場合は、すべての組み込みコマンドは状態 `0' を返す。

シェル組み込みコマンド：実行中のシェルプロセス内で実行されるコマンド

tcsh ：`R.VV.PP' 形式のシェルのバージョン番号。
`R' はメジャーリリース番号、VV' はカレントバージョン、`PP' はパッチレベル

term：端末の種類。通常は ~/.login で設定される。

tty：tty の名前。端末にアタッチされていない場合は空。

attach：（アタッチ）付ける，取り付ける，張り付ける。

uid ：ユーザの実ユーザ ID

user ：ユーザのログイン名

version：バージョン ID スタンプ。シェルのバージョン番号 (tcsh を参照) 、配布元、リリース日、
ベンダー、オペレーティングシステム、マシン (VENDOR, OSTYPE, MACHTYPE を参照)、
コンパイル時に設定されたオプションをカンマで区切ったリストからなる。
ディストリビューションのデフォルトとしてセットされたオプションが記録されている

autolist：あいまいな補完を行った後、可能性のあるものを表示する。
`ambiguous' が設定されている場合、可能 性 のあるものをリストするのは、補完によって何の文字も追加されなかった場合に限られる。

ignoreeof：空文字列あるいは `0' に設定されており、入力デバイスが端末で あ る場合に、end-of-file コマンド (通常は、ユーザが空行に`~D' を打つことで生成される) を入力すると、シェルは終了してしま う 代わりに `Use "exit" to leave tcsh.' と表示する。シェルが kill されてしまうのを防ぐ。
無限ループを避けるため、この設定は 26 回の連続し た EOF の後に終了する。
数値 n を設定している時は、シェ ル は n - 1 回連続した end-of-file を無視し、 n 回目の end-of-file があればそのときに終了する。
(+) これが設定されていない場合には、`1' が使われる。つまり、シェルは `^D' 1 回で終了する。

noglob ： ファイル名置換および、ディレクトリスタック置換 が禁止される。

noclobber：出力リダイレクションに制限がおかれるようになり、入出力の切り替えなどでファイルを 壊

さないように、また、`>>' リダイレクションが存在するファイルを指すようにする。

implicitcd ：シェルは、コマンド入力されたディレクトリ名を、そのディレクトリへ移動すると解釈する。
verbose の設定では、ディレクトリの移動が行われることが標準出力にエコーされるようになる。非対話的なシェルスクリプト、 2 語以上あるコマンド行では禁止されている。
ディレクトリを移動するのは、ディレ クトリ名のような名前を持ったコマンドを実行するよりも優先され、エイリアスの置換よりは後になる。チルダおよび変数 の展開も動作する。

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

環境変数

EDITER:コマンドラインエディタの設定対話型シェルではデフォルトで設定。

GROUP:ユーザのグループ名

HOME:起動したユーザのホームディレクトリに初期化され、ファイル名での `~ の展開には、
この変数が参照される。

HOST: シェルが実行されているマシンのタイプで初期化、gethostname(2)のシステムコールできまる。

PATH:実行可能なコマンドを探すディレクトリのリスト。null 文字はカレントディレクトリを示す。
レントディレクトリを示す。path 変数がない場合、フルパス名での指定で実行される。

LANG: 優先で、固有言語システムの文字環境を得る。

LC_CTYPE: ctype キャラクタのみ変更。

MACHTYPE: コンパイルの時に決まったマシンのタイプ（マイクロプロセッサ　または、マシンモデル）

OSTYPE: コンパイルの時に決まった OS(オペレーションシステム)

REMOTEHOST: ユーザがどのホストからログインしているのかを、示す。

SHLVL:入れ子になったシェルの数。ログインシェルでは 1 にリセットされる。

TERM：端末の種類。通常は ~/.login で設定される。

USER：ユーザのログイン名

PWD: カレントディレクトリのフルパス名 シェル変数 cwd に似ているが、シェル変数とは同期しない。
実際のディレクトリ変更が行われたあとでだけアップデート さ れ る。

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

変数に設定されている設定を調べるには、echo コマンドです。
変数に設定されている設定を参照するには、変数の頭に ＄ をつける。
$変数　と表示するには、\＄変数　と＄変数　のまえにバックスラッシュ\　をつける。

```
% echo $USER
user
% echo \$USER
\$USER
```

書式

# echo[-n]文字列[$変数]['文字列'][“文字列＄変数名　“]

引数の文字列を、標準出力で表示する。

echo　は、引数の間を、１つの空白文字で区切って、最後に改行文字を付けた文字列を
標準出力で表示する。

オプション
-n：文字列の最後に改行を表示しない。（改行文字の手前までを表示する。）

例
ohayou と標準出力
% echo ohayou
ohayou

ｎで終わるファイルを標準出力
% echo *n
mikan　　odenn

シェル変数 home に設定されている設定を表示する。
% echo $home

環境変数 HOME に設定されている設定を表示する。
% echo　$HOME

“$HOME”という文字列を表示する。
% echo \$HOME
\\

有効な環境変数を表示するには、printenvコマンドです。
printenvコマンドを実行して、次に、個別の環境変数 SHELL を表示する

```
%printenv
TERM=cons25
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/games:/usr/local/sbin:/usr/local/bin:/usr/
/X11R6/bin:/home/user/bin
MAIL=/var/mail/user
BLOCKSIZE=k
FTP_PASSIVE_MODE=YES
SHELL=/bin/csh
HOME=/home/user
LOGNAME=user
USER=user
HOSTTYPE=FreeBSD
VENDER=intel
OSTYPE=FreeBSD
MACHTYPE=i386
SHLVL=1
PWD=/home/user
GROUP=wheel
HOST=
EDITER=vi
PAGER=more
%
%printenv SHELL
/bin/csh
%
```

書式

## printenv[環境変数名]

有効な環境変数を表示する。
% printenv

環境変数 EDITER を表示する。
% printenv EDITER

シェル変数に設定値を設定するには、set コマンドを使います。
シェル変数は、アルファベットと　_(アンダースコアー)ではじめる。
設定値は、数字や文字列を使う事が出来る。
set コマンドで、変数名・設定値を　決めずに　set だけで、Enter キーを押せば、シェル変数の一覧が表示されます。

set コマンドで設定された設定値を解除するには、unset コマンドを使います。

set コマンドで、所在地　｛変数名（shozaiti)｝　江東区　｛設定値（ko-to)｝をセットします。
そして、echo コマンドを使いセットされているかをみてみます。

次に、unset コマンドで、設定値を解除します。解除されたかを、echo コマンドを使いセットが、
解除されているかをみてみます。

```
% set shozaiti = ko-to
% echo $shozaiti
ko-to
%unset shozaiti
%echo $shozaiti
shozaiti: Undefined variable.
%
```

shozaiti: Undefined variable　と　未定義の変数だと言われてしまいました。

つぎに、set コマンドで、変数 user　変数の設定値を　/usr/home/user (自分のカレントディレクトリ)を、セットします。
そして、cd コマンドを使い、移動して、pwd コマンドで、現在のカレントディレクトリを確認します。

```
% cd /
%pwd
/
% set user = /usr/home/user
% cd $user
% pwd
/usr/home/user
%
```

書式

## set[オプション][シェル変数名　=設定値]

例

シェル変数 user をつくり、設定値　/usr/home/user を設定する。
set user = /usr/home/user

設定値が設定されている変数の値を展開するには、”（ダブルコーテーション）でくくる。

例
設定値が設定されている変数 user　の設定値を　変数 hana を設定して、展開する。
この花は　/usr/home/user にある。
そして、echo コマンドでみる。

```
% set user = /usr/home/user
% set hana =　“kono hana ha　$user　ni aru”
% echo $hana
kono hana ha /usr/home/user ni aru
%
```

書式

## unset シェル変数名

例
シェル変数　user を、無効にする。
unset user

環境変数を設定するには、setenv コマンドをつかう。
環境変数は、シェルとシェルから起動されたコマンドに有効になる。

```
% cd /
% pwd
/
% setenv USERGO " cd /usr/home/user"
% $USERGO
% pwd
/usr/home/user
%
```

書式

setenv [変数名　設定値]

例
環境変数　AO に　設定値　123 を設定する。
% setenv AO 123
**********

書式

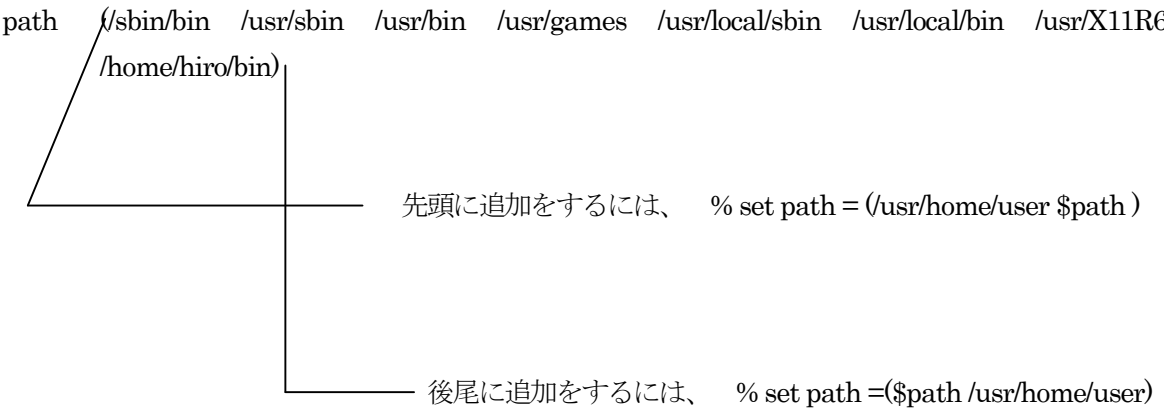
## unsetenv 変数名

環境変数を無効にするには、unsetenv コマンドをつかう。

例
環境変数 BVB を無効にする。
% unsetenv BVB
**********

パスを、設定するためには、シェル変数 path を設定します。
変数 path は、プログラムの本体が置かれているデレクトリまでのパスを設定します。
プログラムの検索は、パスの設定値を先頭から読み込みます。従って、先頭にある。デレクトリでコマンド名が見つかると、そちらを優先的に実行するので、注意をしてください。

例えば、デレクトリ/usr/home/user/work に、パスの設定に追加したければ




まず、echo コマンドで現在の path を確認します。
つぎに、set コマンドで、path にシェル変数の設定値 /usr/home/user/work を後尾に追加をします。
echo コマンドで追加後の path を確認します。
また、set コマンドで、path にシェル変数の設定値 / を 先頭に追加して、echo コマンドで追加後の path を確認します。

```
% echo $path
PATH=/sbin /bin /usr/sbin /usr/bin /usr/games /usr/local/sbin /usr/local/bin /usr/
/X11R6/bin /home/user/bin
% set path = (/usr/home/hiro $path)
% echo $path
PATH=/sbin /bin /usr/sbin /usr/bin /usr/games /usr/local/sbin /usr/local/bin /usr/
/X11R6/bin /home/user/bin /usr/home/user
% set path = (/ $path)
% echo $path
PATH=/ /sbin /bin /usr/sbin /usr/bin /usr/games /usr/local/sbin /usr/local/bin /usr/
/X11R6/bin /home/user/bin /usr/home/user
%
```

シェル変数 prompt を設定すると、% だけでなく、自分の好みのプロンプトを、表示する事が出来ます。
“ “(ダブルクォーテーション)で囲むと空白文字などもプロンプトにする事ができます。
例
自分の名前（user 名）user を　プロンプトにします。

```
% echo $prompt
%
% set prompt = “user%”
user% echo $prompt
user%
user%
```

プロンプトに現在の時間を表示するために、set prompt = %t を実行して
プロンプトとコマンドがくっついて見にくいので、
set prompt = “%t “ と” “ で括り、%t の後ろに空白文字を一ついれます。
次にカレントディレクトリを表示するために、set prompt = %/　を実行します。

```
% set prompt = %t
8:57am
8:57amecho $prompt
%t
8:57amset prompt = “%t “
8:57am
8:57am setprompt =　“%/ “
/usr/home/hiro
/usr/home/hiro echo $prompt
%/
/usr/home/user
```

prompt で使用できる表示

| | |
|---|---|
| %/ | カレントディレクトリ |
| %M | ホスト名 |
| %t , %@ | AM/PM の１２時間表記 |
| %T | M/PM の２４時間表記 |
| %p | %t に秒を追加した１２時間表記 |
| %P | %T に秒を追加した２４時間表記 |
| %n | ユーザ名 |
| %j | ジョブ数 |
| %d | Day 形式の曜日 |
| %D | dd 形式の曜日 |
| %w | Mon 形式の月 |
| %W | mm 形式の月 |
| %y | yy 形式の年 |
| %Y | yyyy 形式の年 |
| %l | シェルの tty |