FreeBSD をさわってみよう　　c シェル２

たくさんのデータの仲から、あるパターンに合致する行を、ファイルからピックアップしたいとき、grep（grobal regulr expression print）を使います。

書式

grep [オプション] 文字列パターン[ファイル]

grep コマンドは、ファイル名を読み込んで、文字列パターンにマッチする部分の行を検索して、その行を表示します。

オプション

-v 文字列パターンに合致しない行を表示する。

-n 行番号を表示する。

-l 文字列パターンに合致したファイル名だけを表示する。

-i　英語の大文字、小文字の区別をしないで検索する。

-e 「-」で始まる文字列パターンを検索する時に使う。

まずは、東京都２３区の識別する番号を、cat コマンド　、または、vi エディターで、入力してみましょう。
ファイル名は、toukyou23　です。
vi エディターは、ここのシリーズの

▼BSDヒロヤン　はじめての　vi ->ここをクリック!!▼

を、参照してください。

cat > toukyou23 と入力し Enter キーをおして、13121　adachiku　と入力し Enter キーをおす。
を、13110　　meguroku と入力し Enter キーを押したら、最後に
Ctrl + D (Ctrl キーを押しながら d キーを押す。)を押す。

```
% cat toukyou23
13121    adachiku
13118    arakawaku
13119    itabashiku
13123    edogawaku
13111    o-taku
13122    katushikaku
13117    kitaku
13108    ko-to-ku
13109    shinagawaku
13113    shibuyaku
13104    shinjyukuku
13115    suginamiku
13107    sumidaku
13112    setagayaku
13106    taito-ku
13102    chu-o-ku
13101    chiyodaku
13116    toshimaku
13114    nakanoku
13120    nerimaku
13105    bunkyo-ku
13103    minatoku
13110    meguroku
%cat toukyou23
13121    adachiku
・　・
途中省略
・　・
・
13110    meguroku
%
```

まずは、先頭の３桁の共通の数字１３１を含む行を grep コマンドで、ピックアップしてみましょう。

```
% grep 131 toukyou23
13121    adachiku
13118    arakawaku
13119    itabashiku
13123    edogawaku
13111    o-taku
13122    katushikaku
13117    kitaku
13108    ko-to-ku
13109    shinagawaku
13113    shibuyaku
13104    shinjyukuku
13115    suginamiku
13107    sumidaku
13112    setagayaku
13106    taito-ku
13102    chu-o-ku
13101    chiyodaku
13116    toshimaku
13114    nakanoku
13120    nerimaku
13105    bunkyo-ku
13103    minatoku
13110    meguroku
%
```

23 区　すべて、揃ったでしょうか？　先頭の３桁の数字、１３１は　東京都を表しています。

次は、わたしの住む区の行を、grep コマンドで、番号や、区の名前で、検索します。

```
% grep 13108 toukyou23
13108    ko-to-ku
% grep ko-to-ku toukyou23
13108
%
```

先ほどの　grep 13108 toukyou23 は、ファイル toukyou23 の中の、13108 の文字のある行を、
探し出し、出力します。
次の　grep ko-to-ku toukyou23 は、ファイル toukyou23 の中の、ko-to-ku の文字のある行を、
探し出し、出力します。

次は、shi　を、含む行を検索して、表示させます。

```
% grep shi toukyou23
13119    itabashiku
13122    katushikaku
13109    shinagawaku
13113    shibuyaku
13104    shinjyukuku
13116    toshimaku
%
```

顧客リストを作ります。商品A,B,C のファイル、kyakuA,B,C あくまでも実名では、ありません。

```
% cat > kyakuA
setagayaku              sakurai          wakaba
ko-to-ku                suzuki           katumi
sumidaku                mita             yoshiko
setagayaku              kura             toshiko
% cat >kyakuB
o-taku          suzuki          toshiko
arakawaku               endo-            hiromi
shibuyaku               kura             toshiko
ko-to-ku        nishino         yoshiko
% cat > kyakuC
edogawaku               tukita           emi
shinagawaku             satou            yumi
katushikaku             sasaki           mari
shibuyaku               satou            mai
taito-ku        ueno            yoshiko
%
```

ko-to-ku (江東区)に、住んでいるお客様は商品A、B、Cの内どのような商品が売れているかを、知りたい、また、その、お客様の氏名がしりたい、などの時にも、grep コマンドを使えます。
{kyaku*,*23}――＞ファイル kyakuA,B,C　および　ファイル toukyou23 から検索と言う意味です。

```
% grep –l ko-to-ku {kyaku*,*23}
kyakuA
kyakuB
tokyo23
% grep ko-to-ku {kyakuA,kyakuB}
kyakuA:ko-to-ku suzuki      katumi
kyakuB:ko-to-ku   nishino   yoshiko
%
```

ko-to-ku(江東区)では、A，Bの商品が、売れています。
そして、ko-to-ku(江東区)では、お客さま
kyakuA:ko-to-ku suzuki　　katumi
kyakuB:ko-to-ku　nishino　yoshiko
suzuki 様、と nishino 様がお買い上げ頂いている事がわかります。

次は、文の頭にある文字を検索してその行を表示します。
grep ^　文の頭にある文字　ファイル名
tai(台東区 taito-ku)、seta(世田谷区 setagayaku)　arak(荒川区 arakawaku)

```
% grep ^tai {kyaku*,*23}
kyakuC:taito-ku　ueno　　yoshiko
% grep ^seta {kyaku*,*23}
kyakuA:setagayaku　　sakurai　wakaba
kyakuA:setagayaku　　kura　　toshiko
% grep ^arak {kyaku*,*23}
kyakuB:arakawaku　　endo-　　hiromi
%
```

次は、文のお尻にある文字を検索してその行を表示します。

grep　文のお尻にある文字$　　ファイル名

名前などを検索して、その行を表示してみましょう。

```
% grep　hiromi$ {kyaku*,*23}
kyakuB:arakawaku　　endo-　　hiromi
% grep toshiko$ toshiko {kyaku*,*23}
kyakuA:setagayaku　　kura　　toshiko
kyakuB:o-taku　　suzuki　　toshiko
% grep ko$ {kyaku*,*23}
kyakuA:sumidaku　　mita　　yoshiko
kyakuA:setagayaku　　kura　　toshiko
kyakuB:o-taku　　suzuki　　toshiko
kyakuB:shibuyaku　　nishino　　yoshiko
kyakuC:taito-ku　　ueno　　yoshiko
%
```

「^」は、行の先頭を表しましたが、[]角括弧のなかで、書くと　「以外」と言う意味になります。
例えば、[^k3] と書かれていれば、　k または、３以外と言う意味と解釈されます。

行の尻が、例えば、o 以外の行を検索して出力してみます。

```
% grep '[^o]$' kyaku*
kyakuA:setagayaku　　sakurai　　wakaba
kyakuA:ko-to-ku　　suzuki　　katumi
kyakuB:arakawaku　　endo-　　hiromi
kyakuC:edogawaku　　tukita　　emi
kyakuC:shinagawaku　　satou　　yumi
kyakuC:katushikaku　　sasaki　　mari
kyakuC:shibuyaku　　satou　　mai
%
```

コマンドライン：
画面上に命令の入力を促すプロンプトが表示され、ユーザがキーボードからコマンド(命令)を入力し、コンピュータに指示を与え、コンピュータはユーザの入力の次の行から処理過程や結果を出力し、再び入力が可能な状態になると改行してプロンプトを表示する過程。

ｃシェルは、入力したコマンドラインを、一つの仕事として、実行しています。一つのコマンドラインの事を、ジョブ (job) といいます。一つのコマンドラインで、複数のコマンドを実行する事ができます。FreeBSD は、プログラム実行の時に、一つのコマンドに対して、ひとのプロセスが割り当てられます。

例えば
ls | more

ls | more で、一つのジョブです。このなかの　ls は一つプロセスです。また、more も一つのプロセスです。

ジョブは、シェル（ｃシェル）からみた一つの仕事の単位です。
プロセスは、FreeBSD と言うＯＳ（オペレーティングシステム）からみた一つの仕事の単位です。

現時点の実行されているプロセスを、表示するために、ps（process status） コマンドがあります。
process status：プロセスの状態

ps コマンドを使って、現時点のプロセスを、見てみましょう。

```
% ps
PID    TT    STAT        TIME        COMMAND
1080   v0    S           0:00.15     -csh(csh)
1082   v0    R+          0:00.02     ps
%
```

PID:プロセス番号
TT:プロセスが実行されている端末
STAT:プロセスの状態
TIME:CPU を使った時間
COMMAND:実行したコマンド

より詳細な情報が欲しい時は、
ps auxww と　コマンドを打ってください。

```
% ps auxww
USER   PID   %CPU   %MEM     VSZ    RSS    TT    STAT  STARTED     TIME          COMMAND
root   10    98.1   0.0       0     8      ??    RL     3:31       3:04.27       [idle]
.
.
%
```

使用例

ユーザ hiro が使っているプロセスを表示する。
% ps auxw –U hiro

プロセス番号１５のプロセスの状態をファイル ffps に書き込む。
% ps auxw –p 15 > ffps

書式　PPID,PGID,JOBC,

ps [オプション]

-u　　USER, PID, %CPU, %MEM ,VSZ ,RSS ,TT , STAT,STATED, TIME, COMMAND を、表示する。

-a　自分のプロセスと自分以外のユーザのプロセスを表示する。

-x　制御端末を持たないプロセスを表示する。

-l　UID, PID, PPID, CPU,PRI,NI,VSZ, RSS,MWCHAN,STATE,TT,TIME,COMMAND を、表示する。

-w　１プロセスの表示列を１３２桁幅で表示する。

-j　　USER, PID,PPID,PGID,SID,JOBC, TT ,　TIME, COMMAND を、表示する。

-t　指定された端末デバイスにとりつけられたプロセスの情報を表示する。

　-t tty

-U　指定されたユーザ（複数を指定できる）のプロセス情報を表示する。

　-U　 user 名

-p　指定したプロセスＩＤ番号に、一致するプロセス情報を表示する。

　-p pid

USER:プロセスを使用しているユーザ

%CPU:CPU の使用率

%MEM:メモリ使用率

VSZ:仮想記憶メモリ使用量（Kbyte）

RSS:物理メモリ上の常駐サイズ（Kbyte）

STAT:プロセスの状態

D	プロセスはディスク（あるいは他の割り込み不可能な短期間の）待ち状態です。
I	プロセスは idle 状態（20 秒以上 sleep している）です。
L	プロセスはロック獲得を待っている状態です。
R	プロセスは 実行 状態です。
S	プロセスは 20 秒未満の sleep 状態です。
T	プロセスは stop している状態です。
W	スレッドは割り込みアイドル状態です。
Z	プロセスは死んでいる状態 です。

STARTED:プロセスを始めた時間

FLAGS:フラグ

NI:プロセス優先度

PPID:親プロセス番号

PGID:プロセスグループ番号

JOBC:ジョブコントロール数

ファイルをツリー構造のなかで検索するときに、再帰的に下の層に下ってファイルを探す時に
find コマンドを使います。

再帰: (回帰)　処理手続きや、規則の定義に、それ自身を繰り返し使う方法。
この find コマンドの場合、　下の層に下ってファイルを探すたびに、同じ探す処理をする事。

例
　/以降　下にある階層のファイル　toukyou23 を検索
% find /-name toukyou23 –print
/usr/home/hiro/toukyou23

文字列を探すために　–exec をつけて、grep コマンドと組あわせて検索する事ができます。

% find / -name toukyou23 –exec grep “ko-to-ku” {} ¥;
13108　　ko-to-ku

grep コマンドを呼び出して, “文字列”を検索して、表示するには、

　　-exec　コマンド　引数　{}　¥;

の形式にする事。

.(カレントディレクトリ)にある toukyou23 と言うファイルをけんさくして、
i ノード番号、ファイルサイズ、保護モード、ハードリンク数、ユーザ名グループ名、バイトサイズ、修正時刻、パス名を知りたい時には、

% find . –name toukyou23 –ls
630　　4 –rw-rw-r--　　1 hiro　　wheel 365 sep 14　17:06　./toukyou23

i ノード : i ノードとは、ファイルに関する情報パーミッション情報(保護ビットともいう)
ファイルタイプ 、ファイルサイズ、修正時刻 、更新時刻、ファイルポインタテーブル(ファイルそのものがある場所を指す)などが記録されているファイル。

ハードリンク:ハードリンクとはコンピュータのファイルシステム上のファイルやディレクトリ等の資源とその資源につけられた名前を結びつけること.

１日前以上にアクセスされてないファイルを検索
% find . –atime 1 -print

書式

find 検索を始めるディレクトリ　　検索条件　　処理の方法

検索条件

-name ファイル名　：　ファイル名で検索

-atime 時刻　：　最後にファイルにアクセスされた時刻まえで検索
-mtime 時刻　：　ファイルの中身が最後に修正された時刻まえで検索
-ctime 時刻　：　ファイルの中身または、属性が変更された時刻でまえで検索

-user ユーザ名　：ファイルのオーナ名で検索

処理の方法

-print：検索結果を表示。
-ls: 検索結果のファイルの ls コマンド処理の情報を表示。
-delete: 検索結果のファイルを削除。
-exec コマンド　：　検索結果に対してコマンドを実行。

例

ファイル名 toukyou23 を指定して、ディレクトリ/hiro 以下のファイルを検索。
find /hiro –name toukyou23 -print

.(カレントディレクトリ)以下の 2 日以上アクセスされなかった。ファイルを検索
find . –name “*” –atime +1 –print

.(カレントディレクトリ)以下の 3 日以内アクセスされた。ファイルを検索
find . –name “*” –atime -2 –print

ディレクトリ/以下からファイル toukyou23 から　ko-to-ku　の文字列を検索
% find / -name toukyou23 –exec grep “ko-to-ku” {} ¥;

書式

sleep 秒

例
１０秒間実行を停止してみます。

```
% sleep 10
%
```

10 秒たったら、プロンプトが帰ってきます。

＆　(アンパサンド)

今までは、プロンプトが表示されているところから、普通にコマンドを実行していました。
端末に入力、出力を行う事をフォアグランドジョブと言います。
そのフォアグランドジョブに対して、バックグランドジョブと言うものがあります。

ｃシェルでは、ダウンロードなどの待機時間が長いコマンドを実行している時に、今までででしたら、
コマンド終了しなければ、プロンプトが戻って来ないので、次の作業まで待たなければ、なりませんでした。しかし、バックグランドジョブを使い、待機しないで、次の作業ができるのです。

例えば　sleep コマンドです。
% sleep 20　で　２０秒間実行を停止していたら、　２０秒間なにも、できません。プロンプトが帰ってこないのですから、

しかし、＆　をコマンドの　最後に付けると　sleep コマンドは、バックグランドで実行され、すぐにプロンプトが帰ってきます。

```
% sleep 20 &
[1] 759
%
```

「1」　の事を、ジョブ番号と言い、
759 の事を、プロセス番号と言います。

カレント:（current）通用している。現行の。いまの、現代の
カレントジョブ:今のジョブ

バックグランドで実行したプログラムや、

Ctrl＋Z（Ctrl キーを押しながら Z キーを押す）をおして、一時中断した時
または、ジョブ番号を忘れた時に jobs コマンドを使い
ジョブ番号、カレントジョブ、ジョブの状態　、プログラム名を、表示する。

例
sleep コマンドをバックグランドで実行してから、jobs コマンドを実行してみましょう。

```
% sleep 20
[1] 1157
% jobs
[1]   + Running                    sleep20
%
```

jyobs コマンドを実行してから表示される

[1]は、ジョブ番号
＋ は、カレントジョブ　（現在のジョブ）
Running は、ジョブの状態

書式

jobs [オプション]

オプション
-l プロセス番号を表示

カレントジョブ
＋ 現在のジョブ
-　 以前のジョブ

ジョブの状態
Runninig　　実行中
Suspended　一時中断
Done　　終了
Terminated　強制終了

例
% jobs -l

バックグランドで実行したジョブをフォアグランドで実行する時や、
Ctrl＋Z（Ctrl1 キーを押しながら z キーを押す）で、一時中断したジョブをまた、元に戻して、実行したい

時に、fg コマンドがあります。

sleep 20　&　　のバックグランドでのコマンドを　フォアグランドで実行させます。

```
% sleep 20 &
[1] 1021
% fg
sleep 20
%
```

次は dmesg | more で、システムのメッセージを表示して、途中で Ctrl + Z (Ctrl1 キーを押しながら z キーを押す)で、一時中断します。
そして、jobs コマンドで確認します。

```
%dmesg | more
Copyright © 1992-2008 The FreeBSD Project.
Copyright © 1979, 1980, 1983, 1986, 1989, 1991, 1992, 1993, 1994
        The Regents of the University of California.  All rights reserved.
FreeBSD RELEASE #0: Web Jan 16 04:18:52 UTC 2000
root@dessler.cse.buffalw.ecu:usr

―――――――――――――――――――――途中省略―――――――――――――――――――――

ata1:<ATAchannel 0> on atapci0

Suspeded
%jobs
[1]   +  Done                    dmesg |
         Suspeded                more
#
```

次にジョブを　また、fg コマンドでフォアグランドで、実行させます。

```
% fg
Copyright © 1992-2008 The FreeBSD Project.
Copyright © 1979, 1980, 1983, 1986, 1989, 1991, 1992, 1993, 1994
        The Regents of the University of California.  All rights reserved.
FreeBSD RELEASE #0: Web Jan 16 04:18:52 UTC 2000
root@dessler.cse.buffalw.ecu:usr




----------途中省略----------



ata1:<ATAchannel 0> on atapci0
```

書式

fg [%ジョブ番号]

ジョブ番号[1]を　フォアグランドで実行させる。
% fg %1

カレントジョブをフォアグランドで実行させる。
% fg

fg コマンドは、引数を指定しなければ、カレントジョブを選択する。

Ctrl＋Z（Ctrl1 キーを押しながら z キーを押す)で、中断した。ジョブをバックグランドで、実行するには、

bg コマンドをつかいます。

例
まず、sleep 50 コマンドで 50 秒間実行を停止して、
すぐに Ctrl + Z (Ctrl1 キーを押しながら z キーを押す)を押す。
そして　bg コマンドを押す。
次に　jobs コマンドで、sleep 50 コマンドが実行中なのを、確認します。
次に 50 秒すぎてから、jobs コマンドで、sleep 50 コマンドが終了なのを、確認します

```
%　sleep 50
^Z
% bg
[1]　　sleep 50 &
% jobs
[1]　　Running　　sleep 50
% jobs
[1]　　Done　　sleep 50
%
```

書式

bg [%ジョブ番号]

例
カレントジョブをバックグランドで、実行する。
% bg

ジョブ番号２のジョブをバックグランドで、実行する。
% bg %2

bg コマンドは、引数を指定しなければ、カレントジョブを選択する。

プロセスやジョブを終了させたり、プロセスにシグナルを送るときに、kill コマンドを使います。

sleep 60 &コマンドを実行して、ジョブ番号とプロセス番号を確認します。

すぐに、jobs コマンドを実行します。
最後に kill コマンドにプロセス番号を入力して、プロセスを終了させます。
次に jobs コマンドでプロセスが終了している事を確認します。

```
% sleep 60 &
[1]  774
% jobs
[1]    + Running                                    sleep 60
% kill 774
[1]       Terminated                                sleep 60
% jobs
%
```

Terminated:終わらせた

また、先ほどと、同じように sleep 60 &コマンドを実行して、ジョブ番号とプロセス番号を確認します。
次に、60 秒後にもう一度 jobs コマンドを実行します。
Done と表示されて、sleep 60 & が、終了した事を確認します。
次に、kill 809 コマンドを実行して、No such pross の表示を確認して、プロセスがない事を確認します。

```
% sleep 60 &
[1] 809
% jobs
[1]      Done                                       sleep 60
% kill 809
809: No such pross
%
```

例
ジョブ番号１，２，３ のジョブを殺す。
kill %1 %2 %3

先ほどは、プロセス番号を指定して、プロセスを、終了させましたが、ジョブを指定して、ジョブを
kill %1 で終了させます。

```
% sleep 60 &
```

```
[1] 889
% kill %1
[1]        Terminated                                    sleep 60
% jobs
%
```

kill コマンドは、プロセス番号のプロセスにシグナルを送ります。他のユーザのプロセスにシグナルを送ることができるのは、スーパユーザです。
kill コマンドを実行する時に、シグナルの指定をしないと、TERM シグナルが送られる。
TERM は、プロセスに終了を知らせます。

kill n	プロセス番号ｎのプロセスを指定する。
kill %	カレントジョブを指定する。
kill %%	カレントジョブを指定する。
kill %+	一つ前のカレントジョブを指定する。
kill %n	ジョブ番号[ｎ]のジョブを指定する。

書式

kill [オプション]　[プロセス番号ｎ又は、ジョブ番号]

オプション
-シグナル：　指定したシグナルをプロセスに送る。
-１　：　シグナルのリストを表示する。

シグナル番号とシグナル名
1　HUP：　(hang up) プロセスに再起動を知らせる。
2　INT：　(interrupt) 端末から Ctrl +C を入力した時に発生するシグナルで、シグナルを受け取ったプロセスは死ぬ。
3　QUIT：　(quit) プロセスを中止する。
端末から Ctrl + ¥を入力した時に発生して、core ファイルをはいて死ぬ。
6　ABRT：　(abort)
9　KILL　: (non-catchable, non-gnorable kill) プロセスに強制終了を知らせる
14　ALRM：(alarm clock)
15　TERM　: (software termination signal) プロセスに終了を知らせる。kill のデフォルトシグナル

abort：失敗する。実をむすばない
alarm clock:目覚まし時計(警告)