

# Open usp Tukubaiコマンドマニュアル

2013年 4月 17日

ユニバーサル・シェル・プログラミング研究所

### cgi-name(1)

#### 【名前】

cgi-name : CGI POSTメソッドで受け渡されるデータをネーム形式に変換

#### 【書式】

```
Usage : cgi-name [-d<c>][-i<string>] <param_file>
Option :          [--template <html>]
Version : Thu Dec 15 13:46:38 JST 2011
```

#### 【説明】

cgi-nameはWEBサーバのCGI POSTメソッドで受け渡されるデータをネーム形式へ変換する。

#### 【例1】

Webサーバよりplace=tokyo&country=japanという文字列が渡される場合、キー値と値をネーム形式に変更して出力する。

```
$ dd bs=$CONTENT_LENGTH | cgi-name
place tokyo
country japan
$
```

#### 【例2】

改行(%0D%0A)はnに変換される。

```
$ echo 'place=%E6%9D%B1%E4%BA%AC%0D%0A%E5%A4%A7%E9%98%AA&country=japan' | cgi-name
place 東京n大阪
country japan
$
```

#### 【例3】

-dオプションを指定すると半角空白は削除あるいは指定文字に変換される。

```
$ echo 'place=tokyo osaka&country=japan' | cgi-name
place tokyo osaka
country japan
$
```

```
$ echo 'place=tokyo osaka&country=japan' | cgi-name -d_
place tokyo_osaka
country japan
$
```

```
$ echo 'place=tokyo osaka&country=japan' | cgi-name -d
place tokyoosaka
country japan
$
```

#### 【例4】

-iオプションを指定することで、返される文字列がヌルの場合の初期値を指定できる。

```
$ echo 'place=&country=japan' | cgi-name
place
country japan
$

$ echo 'place=&country=japan' | cgi-name -i somewhere
place somewhere
country japan
$
```

#### 【例5】

--templateオプションを指定することで、指定したHTMLテンプレートより、radioboxとcheckboxの名称を取り出し、その名称でデータが出力されなければ、出力をその名称で補完する。radioboxやcheckboxは選択しないでsubmitすると値そのものが出力されない。何も選択しなかったというデータを生成するためにこのオプションを使用する。

```
$ cat html
-----省略
<input type="radiobox" name="XXXX" value="A" />
<input type="radiobox" name="XXXX" value="B" />
<input type="radiobox" name="XXXX" value="C" />
<input type="checkbox" name="YYYY" value="a" />
<input type="checkbox" name="YYYY" value="b" />
<input type="checkbox" name="YYYY" value="c" />
-----省略
$

$ dd bs=$CONTENT_LENGTH | cgi-name -i_ --template html
-----省略
XXXX _
YYYY _
-----省略
$
```

#### 【例6】

radiobox checkboxがhtmlテンプレートのmojihame -lコマンド対象部分にある場合においても、タグ名\_%数字を認識して選択値がなくてもタグ名を出力する。

```
$ cat html
-----省略
<!-- MOJIHAME -->
<input type="radiobox" name="X_%1" value="%2" />
<!-- MOJIHAME -->
-----省略
$

$ dd bs=$CONTENT_LENGTH | cgi-name -i_ --template html
-----省略
X _
-----省略
$
```

**【備考】**

CGI POSTメソッドで受け渡されるデータには改行がない。通常のファイルとして使用するには、ファイル末に改行コードを付加しておくほうが扱いやすい。cgi-nameコマンドはファイル末の改行があってもなくても正しく値を取り出す。

**【関連項目】**

mime-read(1)、nameread(1)、ネーム形式(5)

### check\_attr\_name(1)

#### 【名前】

check\_attr\_name : ネームファイルのデータ種類を行ごとに判定

#### 【書式】

```
Usage          : check_attr_name <check_file> <name_file>
Option        : --through <string> --ngword <ng_file>
Attribute     : n N (0以上整数)
               : s S (符号つき整数)
               : f F (小数)
               : v V (符号つき小数)
               : e E (英字)
               : a A (英数字)
               : h H (半角文字)
               : z Z (全角文字)
               : k K (全角カタカナ)
               : x X (文字)
               : c C (チェックディジット)
               : o O (英大文字)
Version       : Thu Dec 15 13:46:38 JST 2011
               : UTF-8
```

#### 【説明】

<check\_file>に記述されているタグ名、桁数+属性の指定にしたがって<name\_file>のデータを判定する。データが大文字なら桁数に等しい値を、小文字なら指定桁数以下の値を要求する。<name\_file>のタグは、タグ名\_数値であれば、\_数値部分を取り除いたタグ名のみでチェックをする。エラーがあった場合は、コマンドはエラー終了し標準出力にタグ名と桁数+属性を出力する。

--through <string>オプションは、<name\_file>のデータが<string>のうちいずれかに等しい場合はチェックを実施しない。<string>のデフォルトは\_となる。このオプションは繰り返し使って複数の<string>を指定することができる。

--ngword <ng\_file>オプションは、Z X K などの日本語文字チェックのときに、<ng\_file>に含まれてるマルチバイト文字をエラーとする。

【例1】通常のチェック

```
$ cat check
A N3          ←3桁整数
B n4          ←4桁以下の整数
C x3          ←5桁以下の文字
D X6          ←6桁の文字
$

$ cat data
A 200
B 12345
C abcde
D_001 xxxxxx ←"D 6X" としてチェックされる
D_002 xxxxx
D_003 xxxx
$

$ check_attr_name check data
B n4
C x3
D_002 X6
D_003 X6
$ echo $?
1
$
```

【例2】--through オプションでチェックしない値を指定する

```
$ cat data2
A 200
B _
C _
D_001 xxxxxx
D_002 _
D_003 _
$

$ check_attr_name --through _ check data2
$ echo $?
0
$
```

【関連項目】

ネーム形式(5)

### check\_need\_name(1)

#### 【名前】

check\_need\_name : ネームファイルの整合性判定

#### 【書式】

```
Usage   : check_need_name <check_file> <name_file>
Option  : --blank <string>
Version : Thu Dec 15 13:46:38 JST 2011
```

#### 【説明】

<check\_file>に記述されているフィールド値に基づいて、<name\_file>ネームファイルの整合性を判断する。<check\_file>に記述されている第1フィールドの値が<name\_file>の第1フィールドのどこにも存在しなかったり、<name\_file>の第2フィールドの値がない、または\_だった場合にコマンドはエラー終了し、標準出力にそのフィールド値を出力する。

<check\_file>ファイルは2フィールドあっても動作する。詳しくは例1を参照のこと。<check\_file>は、フィールド値+(以外の文字列)でも正しく認識する。

--blank <string>オプションは、<name\_file>のデータの値が<string>の場合はデータがないとみなす。つまり、<check\_file>に指定されていて値が<string>である行はエラーになる。<string>のデフォルトは\_となる。

#### 【例1】 通常のチェック

```
$ cat check_file
A
B
C
D
$

$ cat check_file2          ←checkとcheck2は同等
A need
B need
C need
D need
E _
F _
$

$ cat name_file
A _
B 12345
C abcde
D_001 _
D_002 xxx
$

$ check_need_name check_file name_file
A
D_001
$ echo $?
1
$
```

【例2】 --blank オプションでヌルデータを指定する。

```
$ cat data2  
A @  
B 1  
C 2  
D 3  
$
```

```
$ check_need_name --blank @ check data  
A  
$ echo $?  
1  
$
```

【関連項目】

ネーム形式(5)



## cjoin0(1)

### 【名前】

cjoin0 : マスタファイルにキーフィールドが一致するトランザクションファイルの行を抽出

### 【書式】

```
Usage : cjoin0 [+ng[<fd>]] key=<n> <master> <tran>
Version : Fri Jul 27 20:09:40 JST 2012
```

### 【説明】

tranのkey=<n>で指定したキーフィールドがmasterの第1フィールド(キーフィールド)と一致した行のみtranから抽出して出力する。

masterに-を指定すると標準入力をマスタファイルとする。tranが無指定かまたは-が指定されている場合には、標準入力がトランザクションファイルとなる。

masterの第1フィールドは必ず昇順で整列されていることが条件となる。さらに、masterについてはキーフィールド(第1フィールド)について、各行がユニークでなければならない(第1フィールドが同じ値をもつ行が複数あつてはならない)。tranの第<n>フィールドは整列されていなくてもよく、同じ値を持つキーフィールド(第<n>フィールド)はいくつあつても構わない。

キーに選択するフィールドは複数指定することもできる。たとえばkey=3/5のように指定した場合、tranの第3、第4、第5フィールドを意味するようになる。またこの場合、masterのキーは第1、第2、第3フィールドとなる。いずれも、指定したkeyについてmasterもtranも整列されている必要がある。

+ng オプションをつけると、一致した行を標準出力ファイルへ、一致しなかった行をファイルデスクリプタ<fd>のファイルへ出力する。<fd>を省略した場合は標準エラー出力ファイルへ出力する。

tranのキーが整列済みであればcjoin0(1)ではなくjoin0(1)で処理できる。cjoin0(1)はtranのキーが整列されていないものも処理できるが、masterをすべてメモリに読み込むため、メモリ確保エラーが発生する可能性がある。cjoin0(1)はmasterが小さくtranが大容量な場合に効果的に使用できる。

### 【例1】基本パターン

成績ファイルkekkaからmasterに登録されている4人のデータを抽出する。

```
$ cat master
0000003 杉山_____ 26 F
0000005 崎村_____ 50 F
0000007 梶川_____ 42 F
$
```

```
$ cat kekka
0000005 82 79 16 21 80
0000001 46 39 8 5 21
0000004 58 71 20 10 6
0000009 60 89 33 18 6
0000003 30 50 71 36 30
0000007 50 2 33 15 62
$
```

kekkaの第1フィールドがmasterファイルに存在する行のみ抽出する。

```
$ cjoin0 key=1 master kekka
0000005 82 79 16 21 80
0000003 30 50 71 36 30
0000007 50 2 33 15 62
$
```

### 【例2】+ngオプションの使い方

```
$ cjoin0 +ng key=1 master kekka > ok 2> ng
$ cat ng
0000001 46 39 8 5 21
0000004 58 71 20 10 6
0000009 60 89 33 18 6
$
```

### 【例3】標準入力の使い方

```
$ cat kekka | cjoin0 +ng key=1 master > ok 2> ng
$ cat kekka | cjoin0 +ng key=1 master - > ok 2> ng
$ cat master | cjoin0 +ng key=1 - tran > ok 2> ng
```

### 【関連項目】

cjoin0(1)、cjoin1(1)、cjoin2(1)、join1(1)、join2(1)、loopj(1)、loopx(1)、up3(1)、マスタファイル(5)、トランザクションファイル(5)

## cjoin1(1)

### 【名前】

cjoin1 : トランザクションファイルにマスタファイルを連結(一致した行のみ連結)

### 【書式】

```
Usage : cjoin1 [+ng[<fd>]] key=<n> <master> <tran>
Version : Fri Jul 27 20:25:12 JST 2012
```

### 【説明】

tranのkey=<n>で指定したキーフィールドがmasterの第1フィールド(キーフィールド)と一致した行のみtranから抽出して、masterの情報を連結して出力する。連結はtranのキーフィールドの直後にmasterの内容を挿入連結する形で実施される。

masterに-を指定すると標準入力をマスタファイルとする。tranが無指定かまたは-が指定されている場合には、標準入力がトランザクションファイルとなる。

masterの第1フィールドは必ず昇順で整列されていることが条件となる。さらに、masterについてはキーフィールド(第1フィールド)について、各行がユニークでなければならない(第1フィールドが同じ値をもつ行が複数あつてはならない)。tranの第<n>フィールドは整列されていなくてもよく、同じ値を持つキーフィールド(第<n>フィールド)はいくつあつても構わない。

キーに選択するフィールドは複数指定することもできる。たとえばkey=3/5のように指定した場合、tranの第3、第4、第5フィールドを意味するようになる。またこの場合、masterのキーは第1、第2、第3フィールドとなる。いずれも、指定したkeyについてmasterもtranも整列されている必要がある。

+ng<fd>オプションを使うことで、一致した行を標準出力ファイルへ、一致しなかった行をファイルデスクリプタ<fd>のファイルへ出力することが可能。<fd>を省略した場合は標準エラー出力へ出力する。

tranのキーが整列済みであればcjoin1(1)ではなくjoin1(1)で処理できる。cjoin1(1)はtranのキーが整列されていないものも処理できるが、masterをすべてメモリに読み込むため、メモリ確保エラーが発生する可能性がある。cjoin1(1)はmasterが小さくtranが大容量な場合に効果的に使用できる。

### 【例1】基本パターン

```
$ cat master
0000003 杉山_____ 26 F
0000005 崎村_____ 50 F
0000007 梶川_____ 42 F
$

$ cat tran
0000005 82 79 16 21 80
0000001 46 39 8 5 21
0000004 58 71 20 10 6
0000009 60 89 33 18 6
0000003 30 50 71 36 30
0000007 50 2 33 15 62
$

$ cjoin1 key=1 master tran > ok
$ cat ok
0000005 崎村_____ 50 F 82 79 16 21 80
0000003 杉山_____ 26 F 30 50 71 36 30
0000007 梶川_____ 42 F 50 2 33 15 62
$
```

**【例2】 +ngオプションの使い方**

```
$ cjoin1 +ng key=1 master tran > ok 2> ng
$ cat ng
0000001 46 39 8 5 21
0000004 58 71 20 10 6
0000009 60 89 33 18 6
$
```

**【例3】 標準入力の使い方**

```
$ cat tran | cjoin1 +ng key=1 master > ok 2> ng
$ cat tran | cjoin1 +ng key=1 master - > ok 2> ng
$ cat master | cjoin1 +ng key=1 - tran > ok 2> ng
```

**【関連項目】**

cjoin0(1)、cjoin1(1)、cjoin2(1)、join0(1)、join2(1)、loopj(1)、loopx(1)、up3(1)、マスタファイル(5)、トランザクションファイル(5)

## cjoin2(1)

### 【名前】

cjoin2: トランザクションファイルにマスタファイルを連結(一致しなかった行はダミーデータへ置換)

### 【書式】

```
Usage : cjoin2 [-d<string>] [+<string>] key=<key> <master> <tran>
Version : Fri Jul 27 21:03:24 JST 2012
```

### 【説明】

tranのkey=<n>で指定したキーフィールドがmasterの第1フィールド(キーフィールド)と一致した行をtranから抽出し、masterの情報を連結して出力する。一致しない行はダミーデータ\*を連結して出力する。ダミーデータは-dオプションにて指定できる。

masterに-を指定すると標準入力をマスタファイルとする。tranが無指定かまたは-が指定されている場合には、標準入力がトランザクションファイルとなる。

masterの第1フィールドは必ず昇順で整列されていることが条件となる。さらに、masterについてはキーフィールド(第1フィールド)について、各行がユニークでなければならない(第1フィールドが同じ値をもつ行が複数あつてはならない)。tranの第<n>フィールドは整列されていなくてもよく、同じ値を持つキーフィールド(第<n>フィールド)はいくつあつても構わない。

キーに選択するフィールドは複数指定することもできる。たとえばkey=3/5のように指定した場合、tranの第3、第4、第5フィールドを意味するようになる。またこの場合、masterのキーは第1、第2、第3フィールドとなる。いずれも、指定したkeyについてmasterもtranも整列されている必要がある。

tranのキーが整列済みであればcjoin2(1)ではなくjoin2(1)で処理できる。cjoin2(1)はtranのキーが整列されていないものも処理できるが、masterをすべてメモリに読み込むため、メモリ確保エラーが発生する可能性がある。cjoin2(1)はmasterが小さくtranが大容量な場合に効果的に使用できる。

### 【例1】基本パターン

```
$ cat master
0000003 杉山_____ 26 F
0000005 崎村_____ 50 F
0000007 梶川_____ 42 F
$

$ cat tran
0000005 82 79 16 21 80
0000001 46 39 8 5 21
0000004 58 71 20 10 6
0000009 60 89 33 18 6
0000003 30 50 71 36 30
0000007 50 2 33 15 62
$

$ cjoin2 key=1 master tran > ok
$ cat ok
0000005 崎村_____ 50 F 82 79 16 21 80
0000001 ***** ** * 46 39 8 5 21
0000004 ***** ** * 58 71 20 10 6
0000009 ***** ** * 60 89 33 18 6
0000003 杉山_____ 26 F 30 50 71 36 30
0000007 梶川_____ 42 F 50 2 33 15 62
$
```

【例2】 +dオプションの使い方

```
$ cjoin2 -d@@ key=1 master tran > ok
$ cat ok
0000005 崎村_____ 50 F 82 79 16 21 80
0000001 @@ @@ @@ 46 39 8 5 21
0000004 @@ @@ @@ 58 71 20 10 6
0000009 @@ @@ @@ 60 89 33 18 6
0000003 杉山_____ 26 F 30 50 71 36 30
0000007 梶川_____ 42 F 50 2 33 15 62
$
```

【例3】 標準入力の使い方

```
$ cat tran | cjoin2 +ng key=1 master
$ cat tran | cjoin2 +ng key=1 master -
$ cat master | cjoin2 +ng key=1 - tran
```

cjoin0(1)、cjoin1(1)、cjoin2(1)、join0(1)、join1(1)、loopj(1)、loopx(1)、up3(1)、マスタファイル(5)、トランザクションファイル(5)

### comma(1)

#### 【名前】

comma : 指定したフィールドに3桁または4桁のカンマを追加

#### 【書式】

```
Usage : comma [+<n>h][-4] <f1> <f2> ... <file>
       : comma -d[4] <string>
```

Version : Thu Dec 15 13:46:38 JST 2011

#### 【説明】

引数のファイルまたは標準入力のテキストデータの数字フィールドに3桁または4桁のカンマを追加する。カンマを追加するフィールドは引数で指定できる。

#### 【例1】

```
$ cat data
20060201 296030 6710000
20060202 1300100 3130000
20060203 309500 20100
20060204 16300 300100
20060205 41000 210000
20060206 771100 400000
$
```

```
$ comma 2 3 data
20060201 296,030 6,710,000
20060202 1,300,100 3,130,000
20060203 309,500 20,100
20060204 16,300 300,100
20060205 41,000 210,000
20060206 771,100 400,000
$
```

```
$ comma -4 2 3 data ←4桁カンマ
20060201 29,6030 671,0000
20060202 130,0100 313,0000
20060203 30,9500 2,0100
20060204 1,6300 30,0100
20060205 4,1000 21,0000
20060206 77,1100 40,0000
$
```

## 【例2】

+hオプションを使用すると1行目の行を除いてカンマを追加する。先頭行が項目名などの場合に使用する。+2hや+3hのように数字を指定すると、2行目または3行目以降の行にカンマを追加するようになる。

```
$ cat data          ←元データ：先頭行に項目(ヘッダー)がついている
年月日 売上高 発注高
20060201 296030 6710000
20060202 1300100 3130000
20060203 309500 20100
20060204 16300 300100
20060205 41000 210000
20060206 771100 400000
$
```

```
$ comma +h 2 3 data      ←先頭行を除いてカンマ変換する
年月日 売上高 発注高
20060201 296,030 6,710,000
20060202 1,300,100 3,130,000
20060203 309,500 20,100
20060204 16,300 300,100
20060205 41,000 210,000
20060206 771,100 400,000
$
```

## 【例3】

数字の文字列を直接引数に指定してカンマを追加することができる。

```
$ comma -d 1234567
1,234,567
$
```

```
$ comma -d4 1234567
123,4567
$
```

## 【ワンポイント】

データファイルの数値にcommaでカンマを追加してしまうと、それ以降はsm2(1)やdivsen(1)などでの数値計算はできなくなる。カンマの追加はすべての計算を終えた最終成形の段階で使用する。

## 【ワンポイント】

小数桁についてはカンマを追加しない。負数も正しくカンマ処理が実施される。

## 【関連項目】

keta(1)



## count(1)

### 【名前】

count : 同じキーの行数をカウント

### 【書式】

Usage : count <k1> <k2> <file>  
Version : Thu Dec 15 13:46:38 JST 2011

### 【説明】

引数のファイルまたは標準入力のテキストデータの指定したキーフィールドの値が同じ行(行)の数を出力する。キーとなるフィールドは開始となるフィールドkey1から、終了となるフィールドkey2への形で指定する。

### 【例1】

```
$ cat data
01 埼玉県 01 さいたま市 91 59 20 76 54
01 埼玉県 02 川崎市 46 39 8 5 21
01 埼玉県 03 熊谷市 82 0 23 84 10
02 東京都 04 新宿区 30 50 71 36 30
02 東京都 05 中央区 78 13 44 28 51
02 東京都 06 港区 58 71 20 10 6
02 東京都 07 千代田区 39 22 13 76 08
02 東京都 08 八王子市 82 79 16 21 80
02 東京都 09 立川市 50 2 33 15 62
03 千葉県 10 千葉市 52 91 44 9 0
03 千葉県 11 市川市 60 89 33 18 6
03 千葉県 12 柏市 95 60 35 93 76
04 神奈川県 13 横浜市 92 56 83 96 75
04 神奈川県 14 川崎市 30 12 32 44 19
04 神奈川県 15 厚木市 48 66 23 71 24
04 神奈川県 16 小田原市 45 21 24 39 03
$

$ count 1 2 data          ←県ごとの市の数を数えて出力
01 埼玉県 3
02 東京都 6
03 千葉県 3
04 神奈川県 4
$
```

### 【関連項目】

gyo(1)、juni(1)、rank(1)、retu(1)

## ctail(1)

### 【名前】

ctail : ファイルの末尾n行を削って出力

### 【書式】

```
Usage  : ctail -n <file>
        : ctail n <file>
        : ctail -<n>c <file>
Version : Thu Dec 15 13:46:38 JST 2011
UTF-8
```

### 【説明】

引数のファイルまたは標準入力のテキストデータから、引数-n(またはn)で指定した行数の末尾行を除いて標準出力に出力する。

【例】 ファイルの末尾3行を削って表示する。

```
$ cat data
001 北海道
002 東北
003 関東
004 中部
005 近畿
006 中部
007 四国
008 九州
009 沖縄
$
```

```
$ ctail -3 data > data2
$ cat data2
001 北海道
002 東北
003 関東
004 中部
005 近畿
006 中部
$
```

【例2】 ファイルの末尾nバイトを削る

```
$ echo -n abcde | ctail -1c
abcd
$
```

```
$ echo -n abcde | ctail -2c
abc
$
```

### 【関連項目】

getfirst(1)、getlast(1)、tcatt(1)、ycatt(1)

## delf(1)

### 【名前】

delf : 指定したフィールドを除いて出力

### 【書式】

```
Usage   : delf f1 f2 ... file
         : delf -d f1 f2 ... string
Caution : supposed f1 < f2 < f3 < ..
Version  : Thu Dec 15 13:46:38 JST 2011
```

### 【説明】

fileから、指定したフィールドだけを除いて出力する(delf=delete field)。self(1)とは逆の動作となる。ファイル名指定なし、あるいは、ファイル名が-の時は標準入力を期待する。

### 【注意】

f1 < f2 < f3 < ...という関係である必要がある。

```
$ delf 1 3 5 data      ← OK
$ delf 3 1 5 data      ← NG
```

### 【オプション】

-d : **ダイレクトモード**  
文字列<string>にdelf を適用する。

### 【例1】

```
$ cat data          ←元データ
0000000 浜地_____ 50 F
0000001 鈴田_____ 50 F
0000003 杉山_____ 26 F
0000004 白土_____ 40 M
0000005 崎村_____ 50 F
0000007 梶川_____ 42 F
0000008 角屋_____ 50 F
0000009 米原_____ 68 F
0000010 柳本_____ 50 F
0000011 安武_____ 55 F
$
```

```
$ delf 2 data       ←第2フィールドを除いて出力する。
0000000 50 F
0000001 50 F
0000003 26 F
0000004 40 M
0000005 50 F
0000007 42 F
0000008 50 F
0000009 68 F
0000010 50 F
0000011 55 F
$
```

【例2】

```
$ delf NF data
0000000 浜地_____ 50
0000001 鈴田_____ 50
0000003 杉山_____ 26
0000004 白土_____ 40
0000005 崎村_____ 50
0000007 梶川_____ 42
0000008 角屋_____ 50
0000009 米原_____ 68
0000010 柳本_____ 50
0000011 安武_____ 55
$
```

【例3】

```
$ delf 2/3 data
0000000 F
0000001 F
0000003 F
0000004 M
0000005 F
0000007 F
0000008 F
0000009 F
0000010 F
0000011 F
$
```

【注意】

入力ファイルが改行文字で終了していない場合、つまり行が尻切れになっている場合は、ファイル末尾に改行文字が追加されて完結した行になる。

【バグ】

数字名のファイルを使用してはならない。もし10という名前のファイルが存在すると「delf 1 10」が10であるファイルから第1フィールドを削除するという動作をして、標準入力ファイルから第1、10フィールドを削除するという動作をしなくなる。ただし、2のように数字1文字だけを指定した場合は、そのファイルが存在しても、それは無視してフィールドとして扱う。

【関連項目】

self(1)

## divsen(1)

### 【名前】

divsen : 指定フィールドを1,000で除算

### 【書式】

Usage : divsen [-s] <f1> <f2> ... <file>  
Version : Thu Dec 15 13:46:38 JST 2011

### 【説明】

引数のファイルまたは標準入力のテキストデータの数値を1,000で除算し出力する。千単位の表を作成する場合などに使用する。引数において1,000で除算する各フィールドを順に指定する。なお、小数点以下の数値は四捨五入される。-sオプションを指定すると四捨五入せず、小数点の値をそのまま出力する。

### 【例1】

```
$ cat data
20060201 296030 6710000
20060202 1300100 3130000
20060203 309500 20100
20060204 16300 300100
20060205 41000 210000
20060206 771100 400000
$
```

```
$ divsen 2 3 data          ←2フィールドと3フィールドを1,000で除算
20060201 296 6710
20060202 1300 3130
20060203 310 20
20060204 16 300
20060205 41 210
20060206 771 400
$
```

-sを指定して小数点まで表示させる。こうしたデータは以降でmarume(1)などを使って加工する。

```
$ divsen -s 2 3 data      ←小数点まで表示
20060201 296.030 6710.000
20060202 1300.100 3130.000
20060203 309.500 20.100
20060204 16.300 300.100
20060205 41.000 210.000
20060206 771.100 400.000
$
```

### 【ワンポイント】

divsenされた結果をさらにパイプでつないでdivsenを実行すると、100万単位のデータを作成することができる。

### 【関連項目】

marume(1)

## filehame(1)

### 【名前】

filehame : テンプレートにファイルをはめ込み

### 【書式】

```
Usage : filehame -l<string> <template> <data>
Version : Thu Aug 2 14:07:28 JST 2012
```

### 【説明】

<template>ファイルに含まれる文字列<string>が含まれる行を<data>ファイルの中身に置き換える。

### 【例1】

```
$ cat template
<html>
<meta>
<!-- label1 -->
</meta>
<body>
<!-- label2 -->
</body>
</html>
$

$ cat data1
<script type="javascript/text">
  function sb(i) {
    submit(getElementById(i))
  }
</script>
$

$ cat data2
<input id="123" type="button" onclick=sb('123') value="push" />
$

$ filehame -llabel1 template file1 | filehame -llabel2 - file2
<html>
<meta>
<script type="javascript/text">
  function sb(i) {
    submit(getElementById(i))
  }
</script>
</meta>
<body>
<input id="123" type="button" onclick=sb('123') value="push" />
</body>
</html>
$
```

### 【関連項目】

formhame(1)、mojihame(1)

## getfirst(1)

### 【名前】

getfirst : 同一キーの最初の行を出力

### 【書式】

Usage : getfirst [+ng[<fd>]] <n1> <n2> <file>  
Version : Thu Dec 15 13:46:38 JST 2011

### 【説明】

引数のファイルまたは標準入力のテキストデータに対し、指定されたキーフィールドの値が同一の行が複数ある場合に、キーフィールドごとに最初の行のみを抽出して出力する。

### 【例1】

```
$ cat date                ←元データ 商品コード 商品名 販売日 売れ数
0000007 セロリ 20060201 117
0000007 セロリ 20060202 136
0000007 セロリ 20060203 221
0000017 練馬大根 20060201 31
0000017 練馬大根 20060202 127
0000017 練馬大根 20060203 514
0000021 温州みかん 20060201 90
0000021 温州みかん 20060202 324
0000021 温州みかん 20060203 573
0000025 プリンスメロン 20060201 129
0000025 プリンスメロン 20060202 493
0000025 プリンスメロン 20060203 391
0000030 ジャガ芋 20060201 575
0000030 ジャガ芋 20060202 541
0000030 ジャガ芋 20060203 184
$

$ getfirst 1 2 data        ←同一商品コード/商品名の最初の行のみ出力
0000007 セロリ 20060201 117
0000017 練馬大根 20060201 31
0000021 温州みかん 20060201 90
0000025 プリンスメロン 20060201 129
0000030 ジャガ芋 20060201 575
$
```

【例2】

"+ng"オプションを指定すると、同じキーフィールドの値を持つ行の最初の行以外を標準エラー出力に出力する。

```
$ getfirst +ng 1 2 data > /dev/null 2> data2
$ cat data2
0000007 セロリ 20060202 136
0000007 セロリ 20060203 221
0000017 練馬大根 20060202 127
0000017 練馬大根 20060203 514
0000021 温州みかん 20060202 324
0000021 温州みかん 20060203 573
0000025 プリンスメロン 20060202 493
0000025 プリンスメロン 20060203 391
0000030 ジャガ芋 20060202 541
0000030 ジャガ芋 20060203 184
$
```

【例3】

+ng オプションに<fd>を指定すると、標準エラー出力のかわりに<fd>への出力が実施される。

```
$ getfirst +ng4 1 2 data > /dev/null 4> data2
$ cat data2
0000007 セロリ 20060202 136
0000007 セロリ 20060203 221
0000017 練馬大根 20060202 127
0000017 練馬大根 20060203 514
0000021 温州みかん 20060202 324
0000021 温州みかん 20060203 573
0000025 プリンスメロン 20060202 493
0000025 プリンスメロン 20060203 391
0000030 ジャガ芋 20060202 541
0000030 ジャガ芋 20060203 184
$
```

【注意】

フィールド分割において連続する空白は1つの空白とみなされる。また、行頭の空白は無視される。例えば、キーフィールド指定が1と2の場合、次の2行は同じキーフィールドを持つとみなされる。

```
0000007 セロリ 20060202 136
0000007 セロリ 20060203 221
```

【関連項目】

ctail(1)、getlast(1)、tcat(1)、ycat(1)



## getlast(1)

### 【名前】

getlast : 同一キーの最後の行を出力

### 【書式】

Usage : getlast [+ng] <n1> <n2> <file>  
Version : Thu Dec 15 13:46:38 JST 2011

### 【説明】

引数のファイルまたは標準入力のテキストデータに対し、指定されたキーフィールドの値が同一の行が複数ある場合に、キーフィールドごとに最後の行のみを抽出して出力する。

### 【例1】

```
$ cat data          ←元データ 商品コード 商品名 販売日 売れ数
0000007            セロリ 20060201 117
0000007            セロリ 20060202 136
0000007            セロリ 20060203 221
0000017            練馬大根 20060201 31
0000017            練馬大根 20060202 127
0000017            練馬大根 20060203 514
0000021            温州みかん 20060201 90
0000021            温州みかん 20060202 324
0000021            温州みかん 20060203 573
0000025            プリンスメロン 20060201 129
0000025            プリンスメロン 20060202 493
0000025            プリンスメロン 20060203 391
0000030            ジャが芋 20060201 575
0000030            ジャが芋 20060202 541
0000030            ジャが芋 20060203 184
$
```

```
$ getlast 1 2 data
0000007            セロリ 20060203 221
0000017            練馬大根 20060203 514
0000021            温州みかん 20060203 573
0000025            プリンスメロン 20060203 391
0000030            ジャが芋 20060203 184
$
```

【例2】

"+ng"オプションを指定すると、同じキーフィールドの値を持つ行の最後の行以外を標準エラー出力に出力する。

```
$ getlast +ng 1 2 data > /dev/null 2> data2
$ cat data2
0000007          セロリ 20060201 117
0000007          セロリ 20060202 136
0000017          練馬大根 20060201  31
0000017          練馬大根 20060202 127
0000021          温州みかん 20060201  90
0000021          温州みかん 20060202 324
0000025          プリンスメロン 20060201 129
0000025          プリンスメロン 20060202 493
0000030          ジャが芋 20060201 575
0000030          ジャが芋 20060202 541
$
```

【関連項目】

ctail(1)、getfirst(1)、tcat(1)、ycat(1)

## gyo(1)

### 【名前】

gyo : 行をカウント

### 【書式】

Usage : gyo [-f] <file> ...  
Version : Thu Dec 15 13:46:38 JST 2011

### 【説明】

引数のファイルまたは標準入力のテキストデータの行数(行数)をカウントして出力する。

### 【例1】

```
$ cat data
0000000 浜地_____ 50 F 91 59 20 76 54
0000001 鈴田_____ 50 F 46 39 8 5 21
0000003 杉山_____ 26 F 30 50 71 36 30
0000004 白土_____ 40 M 58 71 20 10 6
0000005 崎村_____ 50 F 82 79 16 21 80
0000007 梶川_____ 42 F 50 2 33 15 62
$

$ gyo data
6
$
```

### 【例2】

複数のファイルの行数を一度にカウントできる。

```
$ cat data1
1 file1
2 file1
3 file1
$

$ cat data2
1 file2
2 file2
$

$ cat data3
1 file3
2 file3
3 file3
4 file3
$

$ gyo data1 data2 data3
3
2
4
$
```

**【例3】**

"-f"オプションを使用するとファイル名と行数をそれぞれ表示する。

```
$ gyo -f data1 data2 data3
data1 3
data2 2
data3 4
$
```

**【備考】**

ファイル名に - を指定すると標準入力ファイルを期待する。

```
$ cat data2 | gyo -f data1 - data3
data1 3
data2 2
data3 4
$
```

**【関連項目】**

count(1)、juni(1)、rank(1)、retu(1)

## han(1)

### 【名前】

han : 半角へ変換

### 【書式】

```
Usage : han f1 f2 .. file
       : han -d string
Version : Thu Dec 15 13:46:38 JST 2011
       : UTF-8
```

### 【説明】

引数のファイルまたは標準入力のテキストデータの、全角スペース、全角英数およびカタカナの部分すべてを半角に変換して出力する。

### 【例1】

引数のファイルの指定したフィールドの中身を半角に変換する。

```
$ cat data
これは データ である。
T h i s i s d a t a
1 2 3 4 5 6 7 8 9 0
$
```

```
$ han 1 2 3 data
これは データ である。
This is data
1234 567 890
$
```

### 【例2】

フィールドを指定しないと行全体を半角にする。全角スペースは半角スペースになる。

```
$ cat data2
これはデータである。
全角 スペース データも変換する。
1 2 3 456 7 8 9
$
```

```
$ han data2
これはデータである。
全角 スペース データも変換する。
123456789
```

### 【例3】

-dオプションを使用すると、引数で指定した文字列の全角部分を半角に変えて出力させることができる。

```
$ han -d カタカナABC 1 2 3
かたかABC123
$
```

### 【関連項目】

zen(1)



## join0(1)

### 【名前】

join0 : マスタファイルにキーフィールドが一致するトランザクションファイルの行を抽出

### 【書式】

```
Usage : join0 [+ng[<fd>]] key=<n> <master> <tran>
Version : Thu Dec 15 13:46:38 JST 2011
```

### 【説明】

tranのkey=<n>で指定したキーフィールドがmasterの第1フィールド(キーフィールド)と一致した行のみtranから抽出して出力する。

masterに-を指定すると標準入力をマスタファイルとする。tranが無指定かまたは-が指定されている場合には、標準入力がトランザクションファイルとなる。

masterの第1フィールドおよびtranの第<n>フィールドは必ず昇順で整列されていることが条件となる。また、masterのキーフィールド(第1フィールド)はユニークでなければならない。(第1フィールドが同じ値をもつ行が複数あつてはならない)。tranについてはこの制約はなく、キーフィールド(第<n>フィールド)が同じ値を行はいくつあつても構わない。

キーに選択するフィールドは複数指定することもできる。たとえばkey=3/5のように指定した場合、tranの第3、第4、第5フィールドを意味するようになる。またこの場合、masterのキーは第1、第2、第3フィールドとなる。いずれも、指定したkeyについてmasterもtranも整列されている必要がある。

+ngオプションをつけると、一致した行を標準出力ファイルへ、一致しなかった行をファイルデスク립タ<fd>のファイルへ出力する。<fd>を省略した場合は標準エラー出力ファイルへ出力する。

tranでキーが整列されていないものはjoin0(1)では処理できない。その場合は一旦tranを整列してからjoin0(1)で処理するか、join0(1)のかわりにcjoin0(1)を使って処理すればよい。

【例1】基本パターン

成績ファイルkekkaからmasterに登録されている4人のデータを抽出する。

```
$ cat master
0000003 杉山_____ 26 F
0000005 崎村_____ 50 F
0000007 梶川_____ 42 F
0000010 柳本_____ 50 F
$
```

```
$ cat kekka
0000000 91 59 20 76 54
0000001 46 39 8 5 21
0000003 30 50 71 36 30
0000004 58 71 20 10 6
0000005 82 79 16 21 80
0000007 50 2 33 15 62
0000008 52 91 44 9 0
0000009 60 89 33 18 6
0000010 95 60 35 93 76
0000011 92 56 83 96 75
$
```

kekkaの第1フィールドがmasterファイルに存在する行のみ抽出する。

```
$ join0 key=1 master kekka
0000003 30 50 71 36 30
0000005 82 79 16 21 80
0000007 50 2 33 15 62
0000010 95 60 35 93 76
$
```



【例2】

左から順に複数のフィールドに連続するキーを指定できる。例えばmasterファイルに第2フィールドと第3フィールドの順にフィールドキーが存在する行のみ抽出する場合、次のようになる。

```
$ cat master
A 000003 杉山_____ 26 F
A 000005 崎村_____ 50 F
B 000007 梶川_____ 42 F
C 000010 柳本_____ 50 F
$
```

```
$ cat kekka
1 A 000000 91 59 20 76 54
2 A 000001 46 39 8 5 21
3 A 000003 30 50 71 36 30
4 A 000004 58 71 20 10 6
5 A 000005 82 79 16 21 80
6 B 000007 50 2 33 15 62
7 B 000008 52 91 44 9 0
8 C 000009 60 89 33 18 6
9 C 000010 95 60 35 93 76
10 C 000011 92 56 83 96 75
$
```

```
$ join0 key=2/3 master kekka > data
3 A 000003 30 50 71 36 30
5 A 000005 82 79 16 21 80
6 B 000007 50 2 33 15 62
9 C 000010 95 60 35 93 76
$
```

左から順に連続していない複数のフィールドをキーに指定することも可能。この場合は複数のキーフィールドを@でつなげて指定する。

```
$ join0 key=3@1 master tran
```

### 【例3】 +ngオプション

masterとキーが一致しない"tran"の行を抽出することも可能。キーに一致する行は標準出力に、一致しない行は標準エラー出力に出力される。

```
$ cat master
0000003 杉山_____ 26 F
0000005 崎村_____ 50 F
0000007 梶川_____ 42 F
0000010 柳本_____ 50 F
$
```

```
$ cat kekka
0000000 91 59 20 76 54
0000001 46 39 8 5 21
0000003 30 50 71 36 30
0000004 58 71 20 10 6
0000005 82 79 16 21 80
0000007 50 2 33 15 62
0000008 52 91 44 9 0
0000009 60 89 33 18 6
0000010 95 60 35 93 76
0000011 92 56 83 96 75
$
```

成績ファイルkekkaからmasterに存在する4人のデータと、その他のデータとをそれぞれ抽出する。

```
$ join0 +ng key=1 master kekka > ok-data 2> ng-data
$ cat ok-data          ←一致したデータ
0000003 30 50 71 36 30
0000005 82 79 16 21 80
0000007 50 2 33 15 62
0000010 95 60 35 93 76
$ cat ng-data         ←一致しなかったデータ
0000000 91 59 20 76 54
0000001 46 39 8 5 21
0000004 58 71 20 10 6
0000008 52 91 44 9 0
0000009 60 89 33 18 6
0000011 92 56 83 96 75
$
```

### 【例4】

ファイル名を-にすることにより標準入力からmasterやtranを読むことができる。join1(1)やjoin2(1)も同様。tranの-を省略した場合も標準入力からtranを読み込む。

```
$ cat master | join0 key=1 - tran
$ cat tran | join0 key=1 master -
$ cat tran | join0 key=1 master    ←-を省略可能
```

### 【コラム1】 +ngオプションのコメント -join1(1)も同様

不一致のデータをパイプで次のコマンドへつなぐ場合は次のように記述する。

```
$ join0 +ng key=1 master tran 2>&1 1> ok-data | 次のコマンド
```

不一致のデータのみ出力したい場合には、一致データを/dev/nullへ出力させれば良い。

```
$ join0 +ng key=1 master tran > /dev/null 2> ng-data
```

### 【コラム2】

join0(1)およびjoin1(1)はmasterと同じキーフィールドを持つtranの行を出力するが、パイプの目詰まりを起こさないようにmasterを読み終わったあと読み残したtranについても最後まで読むように動作する。

```
$ cat bigfile | join0 key=1 master > ok-data
```

### 【備考】

keyのフィールドは合わせて128フィールドが最大で、1キーフィールドあたり最大256[byte]まで指定できる。

### 【関連項目】

cjoin0(1)、cjoin1(1)、cjoin2(1)、join1(1)、join2(1)、loopj(1)、loopx(1)、up3(1)、マスタファイル(5)、トランザクションファイル(5)

## join1(1)

### 【名前】

join1 : トランザクションファイルにマスタファイルを連結(一致した行のみ連結)

### 【書式】

```
Usage : join1 [+ng[<fd>]] key=<n> <master> <tran>
Version : Thu Dec 15 13:46:38 JST 2011
```

### 【説明】

tranのkey=<n>で指定したキーフィールドがmasterの第1フィールド(キーフィールド)と一致した行のみtranから抽出して、masterの情報を連結して出力する。連結はtranのキーフィールドの直後にmasterの内容を挿入連結する形で実施される。

masterに-を指定すると標準入力をマスタファイルとする。tranが無指定かまたは-が指定されている場合には、標準入力がトランザクションファイルとなる。

masterの第1フィールドおよびtranの第<n>フィールドは必ず昇順で整列されていることが条件となる。また、masterのキーフィールド(第1フィールド)はユニークでなければならない。(第1フィールドが同じ値をもつ行が複数あってはならない)。tranについてはこの制約はなく、キーフィールド(第<n>フィールド)が同じ値を行はいくつあっても構わない。

masterの第1フィールドおよびtranの第<n>フィールドは必ず昇順で整列されていることが条件となる。また、masterのキーフィールド(第1フィールド)はユニークでなければならない。(第1フィールドが同じ値をもつ行が複数あってはならない)。tranについてはこの制約はなく、キーフィールド(第<n>フィールド)が同じ値を行はいくつあっても構わない。

+ng<fd>オプションを使うことで、一致した行を標準出力ファイルへ、一致しなかった行をファイルデスクリプタ<fd>のファイルへ出力することが可能。<fd>を省略した場合は標準エラー出力へ出力する。

tranでキーが整列されていないものはjoin1(1)では処理できない。その場合は一旦tranを整列してからjoin1(1)で処理するか、join1(1)のかわりにcjoin1(1)を使って処理すればよい。

【例1】基本パターン

経費ファイルkeihiからmasterに登録されている4人のデータを抽出し、マスタファイルの内容を連結する。

```
$ cat master
0000003 杉山_____ 26 F
0000005 崎村_____ 50 F
0000007 梶川_____ 42 F
0000010 柳本_____ 50 F
$
```

```
$ cat keihi
20070401 0000001 300
20070403 0000001 500
20070404 0000001 700
20070401 0000003 200
20070402 0000003 400
20070405 0000003 600
20070401 0000005 250
20070402 0000005 450
20070402 0000007 210
20070404 0000007 410
20070406 0000007 610
$
```

keihiの第2フィールドをキーにしてmasterを結合する。masterに存在するキーをもつ行にしか連結を行わない。存在しない行は捨てられる。keihiは同じキーを持つ行が複数行あっても構わない。

```
$ join1 key=2 master keihi
20070401 0000003 杉山_____ 26 F 200
20070402 0000003 杉山_____ 26 F 400
20070405 0000003 杉山_____ 26 F 600
20070401 0000005 崎村_____ 50 F 250
20070402 0000005 崎村_____ 50 F 450
20070402 0000007 梶川_____ 42 F 210
20070404 0000007 梶川_____ 42 F 410
20070406 0000007 梶川_____ 42 F 610
$
```

【例2】

左から順に連続した複数のフィールドをキーに指定する場合は次のように実行する。kekkaファイルの第2フィールド、第2フィールドをキーにして(第2、第3フィールド順に整列されている)masterファイル(第1、第2フィールドの順に整列されている)に存在する行のみ抽出して、masterの内容を連結して出力している。

```
$ cat master
A 000003 杉山_____ 26 F
A 000005 崎村_____ 50 F
B 000007 梶川_____ 42 F
C 000010 柳本_____ 50 F
$
```

```
$ cat kekka
1 A 000000 91 59 20 76 54
2 A 000001 46 39 8 5 21
3 A 000003 30 50 71 36 30
4 A 000004 58 71 20 10 6
5 A 000005 82 79 16 21 80
6 B 000007 50 2 33 15 62
7 B 000008 52 91 44 9 0
8 C 000009 60 89 33 18 6
9 C 000010 95 60 35 93 76
10 C 000011 92 56 83 96 75
$
```

```
$ join1 key=2/3 master kekka
3 A 000003 杉山_____ 26 F 30 50 71 36 30
5 A 000005 崎村_____ 50 F 82 79 16 21 80
6 B 000007 梶川_____ 42 F 50 2 33 15 62
9 C 000010 柳本_____ 50 F 95 60 35 93 76
$
```

左から順に連続していない複数のフィールドをキーに指定することも可能。この場合は複数のキーフィールドを@でつなげて指定する。なお次の例の場合には、tranは第4、第2フィールドの順に整列され、masterは第3、第1フィールドの順に整列されていることが必要となる。

```
$ join1 key=4@2 master tran
```

### 【例3】 "+ng" オプション

masterとキーに一致しない行を抽出することも可能。キーに一致する行は標準出力に、一致しない行は標準エラー出力に出力される。この場合、一致する行はマスタファイルと連結されるが、一致しない行はマスタファイルに存在しないため連結されずそのまま出力されることになる。

```
$ cat master
0000003 杉山_____ 26 F
0000005 崎村_____ 50 F
0000007 梶川_____ 42 F
0000010 柳本_____ 50 F
$
```

```
$ cat kekka
0000000 91 59 20 76 54
0000001 46 39 8 5 21
0000003 30 50 71 36 30
0000004 58 71 20 10 6
0000005 82 79 16 21 80
0000007 50 2 33 15 62
0000008 52 91 44 9 0
0000009 60 89 33 18 6
0000010 95 60 35 93 76
0000011 92 56 83 96 75
$
```

成績ファイルkekkaからmasterに登録されている4名のデータとその他のデータはそれぞれ次のように抽出できる。

```
$ join1 +ng key=1 master kekka > ok-data 2> ng-data
```

```
$ cat ok-data          ← 一致したデータ
0000003 杉山_____ 26 F 30 50 71 36 30
0000005 崎村_____ 50 F 82 79 16 21 80
0000007 梶川_____ 42 F 50 2 33 15 62
0000010 柳本_____ 50 F 95 60 35 93 76
$
```

```
$ cat ng-data          ← 一致しなかったデータ
0000000 91 59 20 76 54
0000001 46 39 8 5 21
0000004 58 71 20 10 6
0000008 52 91 44 9 0
0000009 60 89 33 18 6
0000011 92 56 83 96 75
$
```

### 【関連項目】

cjoin0(1)、cjoin1(1)、cjoin2(1)、join0(1)、join2(1)、loopj(1)、loopx(1)、up3(1)、マスタファイル(5)、トランザクションファイル(5)

## join2(1)

### 【名前】

join2 : トランザクションファイルにマスタファイルを連結(一致しなかった行はダミーデータへ置換)

### 【書式】

```
Usage : join2 [-d<string>] [+<string>] key=<key> <master> <tran>
Version : Thu Dec 15 13:46:38 JST 2011
```

### 【説明】

tranのkey=<n>で指定したキーフィールドがmasterの第1フィールド(キーフィールド)と一致した行をtranから抽出し、masterの情報を連結して出力する。一致しない行はダミーデータ\*を連結して出力する。ダミーデータは-dオプションにて指定できる。

masterに-を指定すると標準入力をマスタファイルとする。tranが無指定かまたは-が指定されている場合には、標準入力がトランザクションファイルとなる。

masterの第1フィールドおよびtranの第<n>フィールドは必ず昇順で整列されていることが条件となる。また、masterはキーフィールド(第1フィールド)がユニークでなければならない(第1フィールドが同じ値を持つ行が複数あってはならない)。tranについてはこの制約はなく、キーフィールド(第<n>フィールド)が同じ値の行はいくつあっても構わない。

masterの第1フィールドおよびtranの第<n>フィールドは必ず昇順で整列されていることが条件となる。また、masterのキーフィールド(第1フィールド)はユニークでなければならない。(第1フィールドが同じ値をもつ行が複数あってはならない)。tranについてはこの制約はなく、キーフィールド(第<n>フィールド)が同じ値を行はいくつあっても構わない。

tranでキーが整列されていないものはjoin2(1)では処理できない。その場合は一旦tranを整列してからjoin2(1)で処理するか、join2(1)のかわりにcjoin2(1)を使って処理すればよい。



【例1】基本パターン

```
$ cat master
0000003 杉山_____ 26 F
0000005 崎村_____ 50 F
0000007 梶川_____ 42 F
0000010 柳本_____ 50 F
$
```

```
$ cat kekka
0000000 91 59 20 76 54
0000001 46 39 8 5 21
0000003 30 50 71 36 30
0000004 58 71 20 10 6
0000005 82 79 16 21 80
0000007 50 2 33 15 62
0000008 52 91 44 9 0
0000009 60 89 33 18 6
0000010 95 60 35 93 76
0000011 92 56 83 96 75
$
```

masterに存在しない行は次のように\*で補完して出力される。

```
$ join2 key=1 master kekka
0000000 ***** ** * 91 59 20 76 54
0000001 ***** ** * 46 39 8 5 21
0000003 杉山_____ 26 F 30 50 71 36 30
0000004 ***** ** * 58 71 20 10 6
0000005 崎村_____ 50 F 82 79 16 21 80
0000007 梶川_____ 42 F 50 2 33 15 62
0000008 ***** ** * 52 91 44 9 0
0000009 ***** ** * 60 89 33 18 6
0000010 柳本_____ 50 F 95 60 35 93 76
0000011 ***** ** * 92 56 83 96 75
$
```

【例2】

左から順に連続した複数のフィールドをキーに指定する場合次のようになる。第2、第3フィールドでキーが一致するかどうかをチェックしている。

```
$ cat master
A 000003 杉山_____ 26 F
A 000005 崎村_____ 50 F
B 000007 梶川_____ 42 F
C 000010 柳本_____ 50 F
$
```

```
$ cat kekka
1 A 000000 91 59 20 76 54
2 A 000001 46 39 8 5 21
3 A 000003 30 50 71 36 30
4 A 000004 58 71 20 10 6
5 A 000005 82 79 16 21 80
6 B 000007 50 2 33 15 62
7 B 000008 52 91 44 9 0
8 C 000009 60 89 33 18 6
9 C 000010 95 60 35 93 76
10 C 000011 92 56 83 96 75
$
```

```
$ join2 key=2/3 master kekka
1 A 000000 ***** ** * 91 59 20 76 54
2 A 000001 ***** ** * 46 39 8 5 21
3 A 000003 杉山_____ 26 F 30 50 71 36 30
4 A 000004 ***** ** * 58 71 20 10 6
5 A 000005 崎村_____ 50 F 82 79 16 21 80
6 B 000007 梶川_____ 42 F 50 2 33 15 62
7 B 000008 ***** ** * 52 91 44 9 0
8 C 000009 ***** ** * 60 89 33 18 6
9 C 000010 柳本_____ 50 F 95 60 35 93 76
10 C 000011 ***** ** * 92 56 83 96 75
$
```

### 【例3】

+<string>はダミーデータとして補完する文字を指定するオプション。+の後に補完で使用する文字を指定する。

```
$ cat master
0000003 杉山_____ 26 F
0000005 崎村_____ 50 F
0000007 梶川_____ 42 F
0000010 柳本_____ 50 F
$
```

```
$ cat kekka
0000000 91 59 20 76 54
0000001 46 39 8 5 21
0000003 30 50 71 36 30
0000004 58 71 20 10 6
0000005 82 79 16 21 80
0000007 50 2 33 15 62
0000008 52 91 44 9 0
0000009 60 89 33 18 6
0000010 95 60 35 93 76
0000011 92 56 83 96 75
$
```

```
$ join2 +@ key=1 master kekka
0000000 @ @ @ 91 59 20 76 54
0000001 @ @ @ 46 39 8 5 21
0000003 杉山_____ 26 F 30 50 71 36 30
0000004 @ @ @ 58 71 20 10 6
0000005 崎村_____ 50 F 82 79 16 21 80
0000007 梶川_____ 42 F 50 2 33 15 62
0000008 @ @ @ 52 91 44 9 0
0000009 @ @ @ 60 89 33 18 6
0000010 柳本_____ 50 F 95 60 35 93 76
0000011 @ @ @ 92 56 83 96 75
$
```

### 【備考】

masterが0バイトの時は何も連結せずtranそのものを出力する。

```
$ : > master
$ join2 key=1 master tran > data
$ cmp tran data ←同じデータが出力される
```

masterのフィールドを保証したい時は、例えば次のようなコードを書く。

```
$ [ ! -s master ] && echo 0000000 0 > master
$ join2 key=1 master tran
```

### 【関連項目】

cjoin0(1)、cjoin1(1)、cjoin2(1)、join0(1)、join1(1)、loopj(1)、loopx(1)、up3(1)、マスタファイル(5)、トランザクションファイル(5)

## juni(1)

### 【名前】

juni : 同一キー内での順位を出力

### 【書式】

```
Usage : juni [<k1> <k2>] <file>
       : juni -h [<k1> <k2>] <file>
Version : Thu Dec 15 13:46:38 JST 2011
```

### 【説明】

第<k1>フィールドから第<k2>フィールドまでをキーフィールドとして、同一キーの行どうしにおける順位を行の先頭に挿入する。<k1> <k2>を指定しないときは、すべての行が同一キーとみなすものとする。この場合は単純に行の先頭に行番号を挿入するのと同じになる。

-hオプションを指定した場合、各キーフィールドの順位を階層的に出力する。<file>を指定しない、あるいは、-が指定された時には標準入力からのデータ入力を期待する。

### 【例1】

行番号を付与する。

```
$ cat data
0000007 セロリ 100
0000017 練馬大根 95
0000021 温州みかん 80
0000025 プリンスメロン 70
0000030 ジャガ芋 30
$

$ juni data
1 0000007 セロリ 100
2 0000017 練馬大根 95
3 0000021 温州みかん 80
4 0000025 プリンスメロン 70
5 0000030 ジャガ芋 30
$
```

【例2】

キーが関係する場合、キーの開始フィールドと終了フィールドを指定して、同一キーを持つ行どうしにおける順位を行頭に付与する。

```
$ cat data
関東 パスタ 100
関東 おにぎり 90
関東 パン 40
九州 おにぎり 150
九州 パン 140
九州 パスタ 100
$
```

```
$ juni 1 1 data
1 関東 パスタ 100
2 関東 おにぎり 90
3 関東 パン 40
1 九州 おにぎり 150
2 九州 パン 140
3 九州 パスタ 100
$
```

【例3】

-hオプションを指定した場合、各キーフィールドの順位を階層的に出力する。

```
$ cat data
A A1 A11 A111
A A1 A11 A112
A A1 A12 A121
A A1 A12 A122
A A2 A21 A211
A A2 A21 A212
A A2 A22 A221
A A2 A22 A222
B B1 B11 B111
B B1 B11 B112
B B1 B12 B121
B B1 B12 B122
B B2 B21 B211
B B2 B21 B212
B B2 B22 B221
B B2 B22 B222
$
```

```
$ juni -h 1 3 data
1 1 1 A A1 A11 A111
1 1 1 A A1 A11 A112
1 1 2 A A1 A12 A121
1 1 2 A A1 A12 A122
1 2 1 A A2 A21 A211
1 2 1 A A2 A21 A212
1 2 2 A A2 A22 A221
1 2 2 A A2 A22 A222
2 1 1 B B1 B11 B111
2 1 1 B B1 B11 B112
2 1 2 B B1 B12 B121
2 1 2 B B1 B12 B122
2 2 1 B B2 B21 B211
2 2 1 B B2 B21 B212
2 2 2 B B2 B22 B221
2 2 2 B B2 B22 B222
$
```

**【例4】**

-hオプションでキーを指定しない場合には、第1フィールドから最終フィールドまでをキーフィールドとみなす。

```
$ juni -h data
1 1 1 1 A A1 A11 A111
1 1 1 2 A A1 A11 A112
1 1 2 1 A A1 A12 A121
1 1 2 2 A A1 A12 A122
1 2 1 1 A A2 A21 A211
1 2 1 2 A A2 A21 A212
1 2 2 1 A A2 A22 A221
1 2 2 2 A A2 A22 A222
2 1 1 1 B B1 B11 B111
2 1 1 2 B B1 B11 B112
2 1 2 1 B B1 B12 B121
2 1 2 2 B B1 B12 B122
2 2 1 1 B B2 B21 B211
2 2 1 2 B B2 B21 B212
2 2 2 1 B B2 B22 B221
2 2 2 2 B B2 B22 B222
```

**【関連項目】**

count(1)、gyo(1)、rank(1)、retu(1)

## kasan(1)

### 【名前】

kasan : 累計

### 【書式】

```
Usage : kasan [+r] [ref=<ref>] key=<n> <file>
Version : Thu Dec 15 13:46:38 JST 2011
```

### 【説明】

引数のファイルまたは標準入力のテキストデータの指定したフィールドの値を、各行ごとに加算して指定したフィールドの次のフィールドに挿入する。累計値を求める場合などに使用する。

### 【例1】

店別の累計の仕入れ値を算出する。ここでは4フィールド目の値を加算して5フィールド目に挿入する。

```
$ cat data
a店 1日目 103 62          ←店名 日付 売れ数 仕入数
a店 2日目 157 94
a店 3日目 62 30
a店 4日目 131 84
a店 5日目 189 111
a店 6日目 350 20
a店 7日目 412 301
$

$ kasan key=4 data
a店 1日目 103 62 62
a店 2日目 157 94 156
a店 3日目 62 30 186
a店 4日目 131 84 270
a店 5日目 189 111 381
a店 6日目 350 20 401
a店 7日目 412 301 702
$
```



【例2】

あるフィールドの値が変わった場所で加算をリセットする場合は、そのリセットのキーとなるフィールドをref=で指定する。たとえば店別の累計仕入数を算出する場合には、次のように1フィールド目に変更されるごとに加算をリセットする。

```
$ cat data
a店 1日目 103 62          ←店名 日付 売れ数 仕入数
a店 2日目 157 94
a店 3日目 62 30
a店 4日目 131 84
b店 1日目 210 113
b店 2日目 237 121
b店 3日目 150 82
b店 4日目 198 105
c店 1日目 81 52
c店 2日目 76 49
c店 3日目 38 21
c店 4日目 81 48
$
```

```
$ kasan ref=1 key=4 data
a店 1日目 103 62 62
a店 2日目 157 94 156
a店 3日目 62 30 186
a店 4日目 131 84 270
b店 1日目 210 113 113 ←累計加算をリセット
b店 2日目 237 121 234
b店 3日目 150 82 316
b店 4日目 198 105 421
c店 1日目 81 52 52 ←累計加算をリセット
c店 2日目 76 49 101
c店 3日目 38 21 122
c店 4日目 81 48 17
$
```

【例3】

元データのフィールドをその加算値と置き換えて出力する場合は+rオプションを使用する。たとえば仕入れ数を累計仕入れ数に置き換えるのであれば、この場合であれば4フィールド目の加算値を4フィールド目に出力するようにすれば良い。

```
$ cat data          ←店名 日付 売れ数 仕入数
a店 1日目 103 62
a店 2日目 157 94
a店 3日目 62 30
a店 4日目 131 84
a店 5日目 189 111
a店 6日目 350 20
a店 7日目 412 301
$
```

```
$ kasan +r key=4 data
a店 1日目 103 62
a店 2日目 157 156
a店 3日目 62 186
a店 4日目 131 270
a店 5日目 189 381
a店 6日目 350 401
a店 7日目 412 702
$
```

**【例4】**

小数点のあるデータを加算する場合、それまでの行にある最大小数桁数に合わせて出力するようになる。

```
$ cat data
A 1
A 1.2
A 1.23
B 2
B 2.34
B 3
$
```

```
$ kasan ref=1 key=2 data
A 1 1
A 1.2 2.2
A 1.23 3.43
B 2 2
B 2.34 4.34
B 3 7.34
$
```

**【例5】**

key=<pos>で指定できるのは数値、範囲(/)、列挙(@)、NF NF-<n>である。

```
$ kasan key=1 data
```

第2から第3フィールドの各フィールドを加算し、加算値をそれぞれのフィールドの直後に挿入するには次のようにコマンドを実行する。

```
$ cat data
A 1 2
A 3 4
A 5 6
$
```

```
$ kasan key=2/3 data
A 1 1 2 2
A 3 4 4 6
A 5 9 6 12
$
```

第2フィールドと第4フィールドの各フィールドを加算し、加算値をそれぞれのフィールドの直後に挿入するには次のようにコマンドを実行する。

```
$ kasan key=2@4 data
```

次のようにNFも指定できる。

```
$ kasan key=NF data
```

```
$ kasan key=NF-2 data
```

**【関連項目】**

plus(1)、ratio(1)、sm2(1)、sm4(1)、sm5(1)、ysum(1)

## keta(1)

### 【名前】

keta : テキストファイルの桁そろえ

### 【書式】

```
Usage : keta n1 n2 .. <filename>
       : keta -v <filename>
       : keta [--] <filename>
Version : Thu Dec 15 13:46:38 JST 2011
       : UTF-8
```

### 【説明】

引数のファイルまたは標準入力のテキストデータの各フィールドの桁数をそろえて出力する。各フィールドの桁数を自動的に判断して出力できるほか、フィールド毎の桁を指定して出力することが可能。

### 【例1】

引数で指定したファイルの各フィールド毎に最大桁数を判断して桁数を自動出力する。

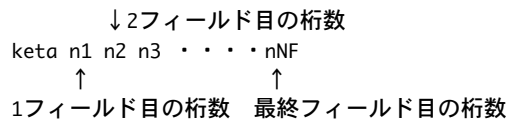
```
$ cat data
01 埼玉県 01 さいたま市 91 59 20 76 54
01 埼玉県 02 川越市 46 39 8 5 21
01 埼玉県 03 熊谷市 82 0 23 84 10
02 東京都 04 新宿区 30 50 71 36 30
02 東京都 06 港区 58 71 20 10 6
04 神奈川県 13 横浜市 92 56 83 96 75
$

$ keta data ←右詰め
01 埼玉県 01 さいたま市 91 59 20 76 54
01 埼玉県 02 川越市 46 39 8 5 21
01 埼玉県 03 熊谷市 82 0 23 84 10
02 東京都 04 新宿区 30 50 71 36 30
02 東京都 06 港区 58 71 20 10 6
04 神奈川県 13 横浜市 92 56 83 96 75
$

$ keta -- data ←左詰め
01 埼玉県 01 さいたま市 91 59 20 76 54
01 埼玉県 02 川越市 46 39 8 5 21
01 埼玉県 03 熊谷市 82 0 23 84 10
02 東京都 04 新宿区 30 50 71 36 30
02 東京都 06 港区 58 71 20 10 6
04 神奈川県 13 横浜市 92 56 83 96 75
$
```

【例2】

各フィールド毎の桁数を指定して出力する。引数としてファイルの各フィールドの1番目から最終フィールドまでのそれぞれの桁数を順に指定する。



なお、桁数は半角での文字数で指定する。大文字の場合は半角2文字分の指定が必要である。keta 4は半角では4文字分、全角では2文字分になる。次のコマンドで例1と同じ出力が得られる。

```
$ keta 2 8 2 10 2 2 2 2 2 data
```

【例3】

通常は桁は右詰めで出力されるが、左詰めで出力する場合は桁数に-をつけて指定すれば良い。例えば第2フィールドと第4フィールドを左詰めにして出力するには次のようにコマンドを実行する。

```

$ keta 2 -8 2 -10 2 2 2 2 2 data
01 埼玉県   01 さいたま市  91 59 20 76 54
01 埼玉県   02 川越市     46 39  8  5 21
01 埼玉県   03 熊谷市     82  0 23 84 10
02 東京都   04 新宿区     30 50 71 36 30
02 東京都   06 港区       58 71 20 10  6
04 神奈川県 13 横浜市     92 56 83 96  7
$

```

【例4】

同じ桁数を連続したフィールドに指定する場合は<桁数x連続するフィールド数>と表記することができる。keta 3x4はketa 3 3 3 3と同じ意味を持つ。フィールド数にはNF(1行のフィールド数)を使用しNF-xxと指定できる。次のコマンドで例3と同じ出力が得られる。

```

$ keta 2 -8 2 -10 2x5 data

$ keta 2 -8 2 -10 2xNF-4 data

```

【例5】

-vオプションを使用するとファイルの各フィールドの最大桁数を表示させることができる。

```

$ keta -v data
2 8 2 10 2 2 2 2 2
$

```

次のコマンドで例1と同じ出力が得られる。

```
$ keta $(keta -v data) data
```

【関連項目】

comma(1)

## keycut(1)

### 【名前】

keycut : キーでファイルを分割

### 【書式】

```
Usage : keycut [options] <filename> <file>
Options : -d : キーの削除
          : -a : ファイル追記
          : -z : 圧縮
Version : Thu Dec 15 13:46:38 JST 2011
(Version: zlib-1.2.3 )
```

### 【説明】

<file>を読み込み、分割先ファイル名<filename>で指定したキーフィールドの値が同一の行でファイルを分割する。例えば、第2フィールドが同じ値を持つ行でファイルを分割したいときは、data.%2のように<filename>を指定する。この時、分割先ファイル名はdata.(第2フィールドの値)となる。

なお、keycutを使用する場合にはキーフィールドの値は事前に整列されている必要がある(内部的にはキーフィールドが変化したところでファイルに出力している)。<filename>におけるキーフィールドは%(フィールド番号)と指定するが、%5.2 %5.1.3のようにサブストリング指定も可能である。

【例1】基本的な使用方法

```
$ cat data
01 埼玉県 03 熊谷市 82 0 23 84 10
01 埼玉県 01 さいたま市 91 59 20 76 54
02 東京都 04 新宿区 30 50 71 36 30
02 東京都 05 中央区 78 13 44 28 51
03 千葉県 10 千葉市 52 91 44 9 0
03 千葉県 12 柏市 95 60 35 93 76
04 神奈川県 13 横浜市 92 56 83 96 75
04 神奈川県 16 小田原市 45 21 24 39 03
$
```

```
$ keycut data.%1 data
```

```
$ ls -l data.*
-rw-r--r-- 1 usp usp 87 2009-02-19 11:14 data.01
-rw-r--r-- 1 usp usp 82 2009-02-19 11:14 data.02
-rw-r--r-- 1 usp usp 77 2009-02-19 11:14 data.03
-rw-r--r-- 1 usp usp 91 2009-02-19 11:14 data.04
$
```

←4つのファイルに分割

```
$ cat data.01
01 埼玉県 03 熊谷市 82 0 23 84 10
01 埼玉県 01 さいたま市 91 59 20 76 54
$
```

```
$ cat data.02
02 東京都 04 新宿区 30 50 71 36 30
02 東京都 05 中央区 78 13 44 28 51
$
```

```
$ cat data.03
03 千葉県 10 千葉市 52 91 44 9 0
03 千葉県 12 柏市 95 60 35 93 76
$
```

```
$ cat data.04
04 神奈川県 13 横浜市 92 56 83 96 75
04 神奈川県 16 小田原市 45 21 24 39 03
$
```

【例2】サブストリング指定

```
$ keycut data.%1.2.1 data
```

```
$ ls -l data.*
-rw-r--r-- 1 usp usp 87 2009-02-19 11:15 data.1
-rw-r--r-- 1 usp usp 82 2009-02-19 11:15 data.2
-rw-r--r-- 1 usp usp 77 2009-02-19 11:15 data.3
-rw-r--r-- 1 usp usp 91 2009-02-19 11:15 data.4
$
```

### 【例3】-aオプション使用例

-aオプションを指定すると分割先ファイルは追記されるようになる。分割先ファイルがないときは新たに作成される。このオプションを指定しない場合にはファイルは上書きとなる。

```
$ keycut data.%1 data

$ keycut -a data.%1 data

$ ls -l data.*
-rw-r--r-- 1 usp usp 174 2009-02-19 11:16 data.01
-rw-r--r-- 1 usp usp 164 2009-02-19 11:16 data.02
-rw-r--r-- 1 usp usp 154 2009-02-19 11:16 data.03
-rw-r--r-- 1 usp usp 182 2009-02-19 11:16 data.04
$

$ cat data.01
01 埼玉県 03 熊谷市 82 0 23 84 10
01 埼玉県 01 さいたま市 91 59 20 76 54
01 埼玉県 03 熊谷市 82 0 23 84 10
01 埼玉県 01 さいたま市 91 59 20 76 54
$
```

### 【例4】-dオプション使用例

-dオプションを指定すると、キーフィールドを除いた行を分割先ファイルに作成するようになる。キーフィールド指定が%1.2.1等のサブstring指定であっても、キーフィールド全体(この例だと第1フィールド全体)が取り除かれる。

```
$ keycut -d data.%1 data

$ ls -l data.*
-rw-r--r-- 1 usp usp 81 2009-02-19 13:13 data.01
-rw-r--r-- 1 usp usp 76 2009-02-19 13:13 data.02
-rw-r--r-- 1 usp usp 71 2009-02-19 13:13 data.03
-rw-r--r-- 1 usp usp 85 2009-02-19 13:13 data.04
$

$ cat data.01
埼玉県 03 熊谷市 82 0 23 84 10
埼玉県 01 さいたま市 91 59 20 76 54
$
```

### 【例5】-zオプション使用例

-zオプションを指定すると、出力ファイルがGzip圧縮形式になる。

```
$ keycut -z data.%1.gz data

$ ls -l data.*
-rw-r--r-- 1 usp usp 98 2009-02-19 13:17 data.01.gz
-rw-r--r-- 1 usp usp 94 2009-02-19 13:17 data.02.gz
-rw-r--r-- 1 usp usp 82 2009-02-19 13:17 data.03.gz
-rw-r--r-- 1 usp usp 100 2009-02-19 13:17 data.04.gz
$

$ gunzip < data.01.gz
01 埼玉県 03 熊谷市 82 0 23 84 10
01 埼玉県 01 さいたま市 91 59 20 76 54
$
```

**【備考1】**

keycut.cはzlibライブラリを使用している。コンパイル時は次のようにオプションが必要。

```
$ cckeycut.c -lz -o /home/T00Lkeycut
```

**【備考2】**

-aオプションと-zオプションを併用したときは、すでに存在する圧縮ファイルに圧縮ファイルが追記される。生成されるファイルはgunzip(1)コマンドで展開できるが、仕様の正しい動作かどうかは未調査。



### loopj(1)

#### 【名前】

loopj : 複数のテキストファイル全行を連結

#### 【書式】

```
Usage : loopj [-d<string>] num=<num> <file1> <file2> ..  
Version : Thu Dec 15 13:46:38 JST 2011
```

#### 【説明】

引数で指定した複数のテキストファイルを、第1フィールドからnum=<n>で指定したフィールドまでをキーにしてすべて連結する(loopj=loop of join)。各ファイルの中でキーフィールドが一致しない行についてはそれぞれ0でフィールドを補完する。

連結する各ファイルは0バイトより大きく、キーとなるフィールドは必ずユニークかつ昇順で整列されていることが条件となる。フィールド数がキーフィールド数に満たない場合エラーとなる。また、<file>が0バイトファイルの場合にもエラーとなる。

#### 【例1】 基本的な使い方

```
$ cat file1  
000003 杉山_____ A  
000005 崎村_____ B  
000007 梶川_____ C  
000010 柳本_____ D  
$
```

```
$ cat file2  
000000 50  
000003 26  
000004 40  
000009 68  
$
```

```
$ cat file3  
000000 F  
000003 F  
000004 M  
000005 F  
$
```

3つのファイルを連結するには次のようにコマンドを実行する。

```
$ loopj num=1 file1 file2 file3  
000000 0 0 50 F  
000003 杉山_____ A 26 F  
000004 0 0 40 M  
000005 崎村_____ B 0 F  
000007 梶川_____ C 0 0  
000009 0 0 68 0  
000010 柳本_____ D 0 0  
$
```

ファイル名を-にすることで、標準入力ファイルを連結することができる。

```
$ cat file2 | loopj num=1 file1 - file3
```

### 【例2】 -d オプション

-dオプションを使用することで補完する文字列を指定することができる。

```
$ loopj -d@@@ num=1 file1 file2 file3
0000000 @@@ @@@ 50 F
0000003 杉山_____ A 26 F
0000004 @@@ @@@ 40 M
0000005 崎村_____ B @@@ F
0000007 梶川_____ C @@@ @@@
0000009 @@@ @@@ 68 @@@
0000010 柳本_____ D @@@ @@@
$
```

### 【例3】 キーのみを持つファイルの連結

キーのみを持つファイルを連結した場合には値のみ補完が実施される。

```
$ cat file1
0001 1
0002 2
$
```

```
$ cat file2
0001
0002
0003
0004
$
```

```
$ loopj num=1 file1 file2
0001 1
0002 2
0003 0
0004 0
$
```

```
$ loopj num=1 file2 file2
0001
0002
0003
0004
$
```

**【注意】**

連結するファイルが0バイトの時はエラーになる。

```
$ : > data1
$ cat data2
a 1
b 2
c 3
$
```

```
$ loopj num=1 data1 data2
Errorloopj] : 0バイトファイル[1]は連結できません。
$
```

上記の例でdata1が、例えば3フィールドあることを保証するには、次のようにコマンドを実行すれば良い。

```
$ [ ! -s data1 ] && echo x 0 0 > data1
$ loopj num=1 data1 data2 | awk '$1~/x/'
a 0 0 1
b 0 0 2
c 0 0 3
$
```

**【関連項目】**

join0(1)、join1(1)、join2(1)、loopx(1)、up3(1)、マスタファイル(5)、トランザクションファイル(5)

### loopx(1)

#### 【名前】

loopx : 総掛けで連結

#### 【書式】

```
Usage  : loopx <file1> <file2> ...  
Version : Thu Jan 12 18:35:55 JST 2012
```

#### 【説明】

複数のファイルの各行に対して、総てのパターンの掛けあわせを作り連結して出力する。出力は指定ファイル順に行が総掛けされる。<file1>の各行に対し<file2>の行が総掛けされ、その結果の各行に対し<file3>の行が総掛けされるという順番に出力される。

#### 【例1】

ファイルdata1、data2、data3を総掛けすると、例えば次のようになる。

```
$ cat data1  
1 農業  
2 工業  
$  
  
$ cat data2  
A 東京  
B 大阪  
$  
  
$ cat data3  
晴  
雨  
$  
  
$ loopx data1 data2 data3  
1 農業 A 東京 晴  
1 農業 A 東京 雨  
1 農業 B 大阪 晴  
1 農業 B 大阪 雨  
2 工業 A 東京 晴  
2 工業 A 東京 雨  
2 工業 B 大阪 晴  
2 工業 B 大阪 雨  
$
```

#### 【関連項目】

join0(1)、join1(1)、join2(1)、loopj(1)、up3(1)、マスタファイル(5)、トランザクションファイル(5)

## map(1)

### 【名前】

map : 縦型ファイルを縦キー/横キー形式の表に変換

### 【書式】

```
Usage : map      [-<l>] num=<n>x<m> <file>
       : map +yarr [-<l>] num=<n>x<m> <file>
       : map +arr  [-<l>] num=<n>x<m> <file>
Option : -m<c> -n
Version : Sat Dec 10 21:04:17 JST 2011
       : OpenUSP (LINUX+FREEBSD/PYTHON2.4/UTF-8)
```

### 【説明】

引数のファイルまたは標準入力のテキストデータが縦キーフィールド<n>個、横キーフィールド<m>個(未指定時 m=1)、残りがデータフィールドであると仮定し、データを表に並べて出力する。表に展開することでデータの欠落が発生する場合は0で補完する。補完する文字は-mオプションで変更できる。

### 【例1】

縦型データをmap(1)するには次のようにコマンドを実行する。num=<n>の値は縦キーのフィールド数を表す。この例ではnumの値は縦キーの最後のフィールドを指し示す2を指定する必要がある。

```
$ cat data
001 a店 01/01 103
001 a店 01/02 157
002 b店 01/01 210
002 b店 01/02 237
$
$ map num=2 data | keta
* * 01/01 01/02
001 a店 103 157
002 b店 210 237
$
```

←縦キー 縦キー 横キー データ部

### 【例2】

データに欠落がある場合、データが存在しない場所に値0が補完される。

```
$ cat data
001 a店 01/01 103
002 b店 01/02 237
$
$ map num=2 data | keta
* * 01/01 01/02
001 a店 103 0
002 b店 0 237
$
```

【例3】

データ部が複数列ある場合には複数行にマッピングが実施される。この時、各行にはA B C ...と順にインデックスが追加される。

```

$ cat data
a店 1日目 103 62
a店 2日目 157 94
a店 3日目 62 30
a店 4日目 131 84
.
.
d店 6日目 98 69
d店 7日目 101 90
$

$ map num=1 data | keta
* * 1日目 2日目 3日目 4日目 5日目 6日目 7日目
a店 A 103 157 62 131 189 350 412
a店 B 62 94 30 84 111 20 301
b店 A 210 237 150 198 259 421 589
b店 B 113 121 82 105 189 287 493
c店 A 81 76 38 81 98 109 136
c店 B 52 49 21 48 61 91 110
d店 A 75 72 34 74 91 98 101
d店 B 48 42 19 43 51 69 90
$

```

【例4】

例2の結果をさらにsed(1)コマンドでAを売数、Bを客数に置換すると次のような表になる。

```

$ map num=1 data | sed -e 's/A/売数/1' -e 's/B/客数/1' | keta
* * 1日目 2日目 3日目 4日目 5日目 6日目 7日目
a店 売数 103 157 62 131 189 350 412
a店 客数 62 94 30 84 111 20 301
b店 売数 210 237 150 198 259 421 589
b店 客数 113 121 82 105 189 287 493
c店 売数 81 76 38 81 98 109 136
c店 客数 52 49 21 48 61 91 110
d店 売数 75 72 34 74 91 98 101
d店 客数 48 42 19 43 51 69 90
$

```

【例5】

+yarrオプションを指定することでデータ部の複数列を横に展開することができる。

```
$ map +yarr num=1 data | keta
* 1日目 1日目 2日目 2日目 3日目 3日目 -- 7日目 7日目
*   a    b    a    b    a    b --   a    b ←自動的に追加される
a店 103   62  157   94   62   30 --  412  301
b店 210  113  237  121  150   82 --  589  493
c店  81   52   76   49   38   21 --  136  110
d店  75   48   72   42   34   19 --  101   90
$
```

```
$ map +yarr num=1 data | sed -e '2s/a/売数/g' -e '2s/b/客数/g' | keta
* 1日目 1日目 2日目 2日目 3日目 3日目 -- 7日目 7日目
* 売数 客数 売数 客数 売数 客数 -- 売数 客数
a店 103   62  157   94   62   30 --  412  301
b店 210  113  237  121  150   82 --  589  493
c店  81   52   76   49   38   21 --  136  110
d店  75   48   72   42   34   19 --  101   90
$
```

横インデックスが不要な場合には+arrオプションを使用する。

```
$ map +arr num=1 data | keta
* 1日目 1日目 2日目 2日目 3日目 3日目 -- 7日目 7日目
a店 103   62  157   94   62   30 --  412  301
b店 210  113  237  121  150   82 --  589  493
c店  81   52   76   49   38   21 --  136  110
d店  75   48   72   42   34   19 --  101   90
$
```

【例6】

-mオプションで補完文字を指定できる。デフォルトは0。

```
$ cat data
A a 1
A b 2
B a 4
$

$ map -m@ num=1 data
* a b
A 1 2
B 4 @
$
```

【例7】

-<数値>オプションを妻うことで<数値>フィールド分ずつmap(1)を実施することができる。<数値>はデータ部のフィールド数の約数である必要がある。

```
$ cat data
X x 1 2 3 4 5 6      ←データ部を1 2 3と4 5 6というように3つずつmapしたい
X y 1 2 3 4 5 6
Y x 1 2 3 4 5 6
Y y 1 2 3 4 5 6
$
```

```
$ map -3 num=1 data      ←3フィールド分ずつmap(1)する
* * x x y y
* * a b a b
X A 1 4 1 4
X B 2 5 2 5
X C 3 6 3 6
Y A 1 4 1 4
Y B 2 5 2 5
Y C 3 6 3 6
$
```

```
$ map +yarr -3 num=1 data      ←+yarrの場合、3フィールド分ずつ折り返し
* * x x x y y y
* * a b c a b c
X A 1 2 3 1 2 3
X B 4 5 6 4 5 6
Y A 1 2 3 1 2 3
Y B 4 5 6 4 5 6
$
```



【例8】

num=<n>x<m>で縦キー<n>個、横キー<m>個にすることができる。横キーはヘッダー<m>行に展開される。

```
$ cat data
X1 Y1 Z1 1 8
X1 Y1 Z2 2 7
X1 Y2 Z1 3 6
X1 Y2 Z2 4 5
X2 Y1 Z1 5 4
X2 Y1 Z2 6 3
X2 Y2 Z1 7 2
X2 Y2 Z2 8 1
$
```

```
$ map num=1x2 data
* * Y1 Y1 Y2 Y2
* * Z1 Z2 Z1 Z2
X1 A 1 2 3 4
X1 B 8 7 6 5
X2 A 5 6 7 8
X2 B 4 3 2 1
$
```

←横ヘッダーが2行になる

```
$ map +yarr num=1x2 data
* Y1 Y1 Y1 Y1 Y2 Y2 Y2 Y2
* Z1 Z1 Z2 Z2 Z1 Z1 Z2 Z2
* a b a b a b a b
X1 1 8 2 7 3 6 4 5
X2 5 4 6 3 7 2 8 1
$
```

←横ヘッダーが2行になる

【例9】

-<数字>オプションとnum=<n>x<m>を組み合わせると次のように動作する。

```
$ cat data3
X1 Y1 Z1 1 8 4 5 6 7
X1 Y1 Z2 2 7 4 5 6 7
X1 Y2 Z1 3 6 4 5 6 7
X1 Y2 Z2 4 5 4 5 6 7
X2 Y1 Z1 5 4 4 5 6 7
X2 Y1 Z2 6 3 4 5 6 7
X2 Y2 Z1 7 2 4 5 6 7
X2 Y2 Z2 8 1 4 5 6 7
$
```

```
$ map -3 num=1x2 data3
* * Y1 Y1 Y1 Y1 Y2 Y2 Y2 Y2
* * Z1 Z1 Z2 Z2 Z1 Z1 Z2 Z2
* * a b a b a b a b
X1 A 1 5 2 5 3 5 4 5
X1 B 8 6 7 6 6 6 5 6
X1 C 4 7 4 7 4 7 4 7
X2 A 5 5 6 5 7 5 8 5
X2 B 4 6 3 6 2 6 1 6
X2 C 4 7 4 7 4 7 4 7
$
```

```
$ map +yarr -3 num=1x2 data3
* * Y1 Y1 Y1 Y1 Y1 Y1 Y2 Y2 Y2 Y2 Y2 Y2
* * Z1 Z1 Z1 Z2 Z2 Z2 Z1 Z1 Z1 Z2 Z2 Z2
* * a b c a b c a b c a b c
X1 A 1 8 4 2 7 4 3 6 4 4 5 4
X1 B 5 6 7 5 6 7 5 6 7 5 6 7
X2 A 5 4 4 6 3 4 7 2 4 8 1 4
X2 B 5 6 7 5 6 7 5 6 7 5 6 7
$
```

【例10】

-nオプションを指定すると、付加するインデックスが数値になる。-nを付けない場合にはアルファベットのwが使われるのでインデックス数が26個までという制限があるが、-nを指定した場合にはインデックスが26個に制限されず、データ部が多い場合もマップすることができる。

```
$ cat data4
X1 Y1 A1 A2 A3 A4 -- A99 A100
X1 Y2 B1 B2 B3 B4 -- B99 B100
X1 Y3 C1 C2 C3 C4 -- C99 C100
$
```

```
$ map -n num=1 data4
* * Y1 Y2 Y3
X1 1 A1 B1 C1
X1 2 A2 B2 C2
.
.
X1 29 A99 B99 C99
X1 30 A100 B100 C100
$
```

【関連項目】

unmap(1)、tateyoko(1)

### marume(1)

#### 【名前】

marume : 四捨五入、切り上げ、切り捨て

#### 【書式】

```
Usage : marume [+age| -sage] <f1.d1> <f2.d2> .. <file>
Version : Thu Dec 15 13:46:38 JST 2011
```

#### 【説明】

引数のファイルまたは標準入力のテキストデータの指定のフィールドを、指定された桁数にて四捨五入、切り上げ、切り捨てを実行して出力する。デフォルトの動作は四捨五入。+ageで切り上げ、-sageで切り下げを明示的に指定できる。小数点下に0を指定(n.0)すると該当フィールドを整数で出力する。また、桁数を0+桁数にすると、整数部の桁数以下を四捨五入、切り上げ、切り捨てする。

#### 【例1】

```
$ cat data
01 0.3418 1.5283 9.0023 7.1234 1234
02 3.1242 7.1423 6.5861 2.7735 1235
03 6.8254 2.6144 4.3234 3.4231 -1234
04 7.0343 3.3312 7.8678 1.3295 -1235
$
```

第2フィールドを整数へ四捨五入、第3フィールドを小数点下1桁で四捨五入、第4フィールドを小数点下2桁で四捨五入、第5フィールドを小数点下3桁で四捨五入、第6フィールドを1の位で四捨五入して出力すると次のようになる。

```
$ marume 2.0 3.1 4.2 5.3 6.01 data
01 0 1.5 9.00 7.123 1230
02 3 7.1 6.59 2.774 1240
03 7 2.6 4.32 3.423 -1230
04 7 3.3 7.87 1.330 -1240
$
```

#### 【例2】

例1をすべて四捨五入から切る上げに変更すると次のようになる。

```
$ marume +age 2.0 3.1 4.2 5.3 6.01 data
01 1 1.6 9.01 7.124 1240
02 4 7.2 6.59 2.774 1240
03 7 2.7 4.33 3.424 -1240
04 8 3.4 7.87 1.330 -1240
$
```

#### 【例3】

例1をすべて四捨五入から切り下げに変更すると次のようになる。

```
$ marume -sage 2.0 3.1 4.2 5.3 6.01 data
01 0 1.5 9.00 7.123 1230
02 3 7.1 6.58 2.773 1230
03 6 2.6 4.32 3.423 -1230
04 7 3.3 7.86 1.329 -1230
$
```

**【注意】**

負数は正数を四捨五入、切り上げ、切り捨てしたものにマイナス符号をつけた値となる。

**【関連項目】**

divsen(1)

## mdate(1)

### 【名前】

mdate : 日付、週、月の処理

### 【書式】

Usage :

#### DIRECT-MODE

日付	mdate -y <yyyymmdd>	: 曜日
	mdate -e <yyywwdd>/±<dif>	: dif 日先までの日付を連続出力
	mdate -e <yyyymmdd1> <yyyymmdd2>	: 日付の範囲を連続出力
	mdate <yyywwdd>/±<dif>	: dif 日先の日付
	mdate <yyyymmdd1> <yyyymmdd2>	: 日付の差
	mdate <yyyymm>m/±<dif>	: dif 月先の月
	mdate -e <yyyymm>m/±<dif>	: dif 月先までの月を連続出力
	mdate <yyyymm1>m <yyyymm2>m	: 月の差
	mdate -ly <yyyymm>m	: 前年月

#### FILTER-MODE

日付	mdate -f -y <f>	: 曜日
	mdate -f -e <f>/±<dif>	: dif 日先までの日付に展開
	mdate -f -e <f1> <f2>	: 日付間の展開
	mdate -f <f>/±<dif>	: dif 日先の日付
	mdate -f <f1> <f2>	: 日付の差
	mdate -f <f1> ±<f2>	: 日付の加算
	mdate -f -e <f1> ±<f2>	: 日付の加算または展開
	mdate -f -ly <f>	: 前年日
月次	mdate -f -d <f>m	: 日付を1ヶ月分出力
	mdate -f <f>m/±<dif>	: dif 月先の月
	mdate -f -e <f>m ±<dif>	: dif 月先の月まで展開
	mdate -f <f1>m <f2>m	: 月の差
	mdate -f -e <f1>m <f2>m	: 月の展開
	mdate -f <f>m ±<dif>	: 月の加算
	mdate -f -e <f>m ±<dif>	: 月の加算展開
	mdate -f -ly <f>m	: 前年月

Version : Wed Dec 28 15:32:10 JST 2011

### 【例1】 dif先の日付、月の情報を出力

```
$ mdate 20090912/+5
20090917
$
```

```
$ mdate 200909m/+5
201002
$
```

### 【例2】 dif先までの連続した日付、月の情報を出力

```
$ mdate -e 20090912/+5
20090912 20090913 20090914 20090915 20090916 20090917
$
```

```
$ mdate -e 200909m/+5
200909 200910 200911 200912 201001 201002
$
```

【例3】日付、月の差を出力

```
$ mdate 20090917 20090912
5
$
```

```
$ mdate 201002m 200909m
5
$
```

【例4】日付、月の範囲を連続出力

```
$ mdate -e 20090912 20090917
20090912 20090913 20090914 20090915 20090916 20090917
$
```

```
$ mdate -e 200909m 201002m
200909 200910 200911 200912 201001 201002
$
```

【例5】フィルターモード：dif先の日付、月の情報を挿入

```
$ cat date_data
A 20090901 B
A 20090902 B
A 20090903 B
$
```

```
$ mdate -f 2/+5 date_data
A 20090901 20090906 B
A 20090902 20090907 B
A 20090903 20090908 B
$
```

```
$ cat month_data
A 200909 B
A 200910 B
A 200911 B
$
```

```
$ mdate -f 2m/+5 month_data
A 200909 201002 B
A 200910 201003 B
A 200911 201004 B
$
```

【例6】フィルターモード：dif先までの連続した日付、週、月の情報を挿入

```
$ cat date_data
A 20090901 B
A 20090902 B
A 20090903 B
$
```

```
$ mdate -f -e 2/+5 date_data
A 20090901 20090902 20090903 20090904 20090905 20090906 B
A 20090902 20090903 20090904 20090905 20090906 20090907 B
A 20090903 20090904 20090905 20090906 20090907 20090908 B
$
```

```
$ cat month_data
A 200909 B
A 200910 B
A 200911 B
$
```

```
$ mdate -f -e 2m/+5 month_data
A 200909 200910 200911 200912 201001 201002 B
A 200910 200911 200912 201001 201002 201003 B
A 200911 200912 201001 201002 201003 201004 B
$
```

【関連項目】

calclock(1)、dayslash(1)、yobi(1)

## mime-read(1)

### 【名前】

mime-read : MIME形式のファイル読み込み

### 【書式】

```
Usage   : mime-read <name> <MIME-file>
         : mime-read -v <MIME-file>
Version : Thu Dec 15 13:46:38 JST 2011
```

### 【説明】

MIME形式のファイル<MIME-file>ファイルの各セクションの中から、name="<name>"の記述があるパートを見つけだし、その部分のデータを出力する。-v を指定した場合はすべてのname="..."を対象として名前の一覧を出力する。

### 【例】

mime-fileの中からabc.xlsで指定される名前のセクションを取り出す。

```
$ mime-read abc.xls mime-file > abc.xls
```

### 【関連項目】

cgi-name(1)、nameread(1)



## mojihame(1)

### 【名前】

mojihame : テンプレートに文字をはめ込み

### 【書式】

```
Usage : mojihame <template> <data>                (通常)
       : mojihame -l <label> <template> <data>    (行単位)
       : mojihame -h <label> <template> <data>    (階層データ)
Version : Sat Jan 24 11:09:49 JST 2009
```

```
Usage : mojihame-l [-l<label>][-dlt;c>] <template> <data>
Version : Sat Jan 24 13:31:13 JST 2009
```

```
Usage : mojihame-h -h<label> [-dlt;c>] <template> <data>
Version : Sat Jan 24 13:59:43 JST 2009
```

```
Usage : mojihame-p [-d<c>] <template> <data>
Version : Sat Jan 24 13:30:10 JST 2009
```

### 【説明】

引数に指定するtemplateファイルに、dataあるいは環境変数からデータを読んで値をはめ込む。次の3つの使い方がある。

1. dataのすべてのフィールドを順番にはめ込む(通常)
2. dataの行単位のフィールドをtemplateの指定ラベル間繰り返しはめ込む(行単位)
3. dataが階層データの場合、templateの指定ラベル間、階層的にはめ込む(階層データ)

### 【例1】 通常

templateの中にある%1 %2 %3 ...の部分に、dataのフィールド(左上から右下に順番に%1 %2 %3 ...に対応)を読んで置換する。

```
$ cat data
a b
c d
$
```

```
$ cat template
1st=%1
2nd=%2
3rd=%3 4th=%4
$
```

```
$ mojihame template data
1st=a
2nd=b
3rd=c 4th=d
$
```

## 【例2】 通常その2

デフォルトでは@というデータはヌル文字列として置換される。このデフォルトを変更するには-dオプションを使用する。-dオプションを単独で使用した場合には、ヌル文字列に置換する機能は無効になる。

```
$ cat data
a @
@ d
$
```

```
$ cat template
1st=%1
2nd=%2
3rd=%3 4th=%4
$
```

```
$ mojihame template data
1st=a
2nd=
3rd= 4th=d
$
```

```
$ mojihame -da template data
1st=
2nd=@
3rd=@ 4th=d
$
```

## 【例3】 行単位

dataは1行単位に読み込まれ、行の各フィールドが%1 %2 ...に対応する。次の行が読み込まれるときは再びtemplateが使用され、データがはめ込まれ出力される。この使い方は旧mojihameの-rと互換性がある。mojihame -rは将来廃止される。

```
$ cat data
a b c d
w x y z
$
```

```
$ cat template
1st=%1 2nd=%2
3rd=%3 4th=%4
$
```

```
$ mojihame -l template data
1st=a 2nd=b
3rd=c 4th=d
1st=w 2nd=x
3rd=y 4th=z
$
```

```
$ mojihame -r template data
1st=a 2nd=b
3rd=c 4th=d
1st=w 2nd=x
3rd=y 4th=z
$
```

#### 【例4】行単位その2

-lオプションで引数(ラベル)を指定した場合、templateのラベルで囲まれた行のみにデータが繰り返しはめ込まれる。このときラベルを含む行は出力されない。また、ラベルにはさまれていない前後の部分は何も変換されず、そのまま出力される。ラベルはtemplate内部に2回だけ記述可能で、それぞれはめ込み対象の開始行と終了行とみなされる。指定ラベルは部分一致でラベル行と判断するので注意が必要。

```
$ cat data
a b
y z
$
```

```
$ cat template
header %1
LABEL
1st=%1 2nd=%2
LABEL
footer %2
$
```

```
$ mojihame -lLABEL template data
header %1
1st=a 2nd=b
1st=y 2nd=z
footer %2
$
```

### 【例5】行単位その3

-lオプションを繰り返し使用することにより、プルダウンなどのHTMLを簡易に生成することができる。指定ラベルはtemplateに1組しか認められていないので、次の例のように内側のラベルから順番に値をはめていくという手順でコマンドを実行する。

```
$ cat member  
佐藤  
鈴木  
$
```

```
$ cat kbn1  
東京  
大阪  
横浜  
$
```

```
$ cat kbn2  
男  
女  
$
```

```
$ cat template  
MEMBER  
氏名=%1  
K1  
地名=%1  
K1  
K2  
性別=%1  
K2  
MEMBER  
$
```

```
$ mojihame -lK1 template kbn1 | mojihame -lK2 - kbn2 | mojihame -lMEMBER - member  
氏名=佐藤  
地名=東京  
地名=大阪  
地名=横浜  
性別=男  
性別=女  
氏名=鈴木  
地名=東京  
地名=大阪  
地名=横浜  
性別=男  
性別=女  
$
```

### 【例6】行単位その4

デフォルトでは@というデータはヌル文字列として置換される。このデフォルトを変更するには-dオプションを使用する。-dオプションは-lオプションの後に指定する必要がある。

```
$ mojihame -lLABEL -dx template data
```

### 【例7】階層データ

-hオプションは階層的にデータのはめ込みを実施する。templateにおいて階層的ラベルに囲まれた部分に現れる%数字がdataの階層的キーフィールドに対応する。

```
$ cat data
山田 東京 10:00
山田 大阪 20:00
山田 横浜 09:30
鈴木 東京 16:45
鈴木 神戸 15:30
$
```

次のtemplateの場合、LABEL-1に囲まれているのは%1、LABEL-1に囲まれている部分にさらにLABEL-2があり、LABEL-2で囲まれている部分に%2と%3がある。このとき、第1フィールドが変化する回数だけ、LABEL-1で囲まれている部分が繰り返し値がはめられ、LABEL-2で囲まれている部分は、同じ第1フィールドの値を持つ第2、第3フィールドの行の数だけ繰り返し値がはめられる。

```
$ mojihame -hLABEL template dat
表題 %1
氏名=山田
地名=東京 時刻=10:00
地名=大阪 時刻=20:00
地名=横浜 時刻=09:30
氏名=鈴木
地名=東京 時刻=16:45
地名=神戸 時刻=15:30
$
```

```
$ cat template
表題 %1
LABEL-1
氏名=%1
LABEL-2
地名=%2 時刻=%3
LABEL-2
LABEL-1
$
```

階層の深さに制約はない。ラベルは部分一致でラベル行と判定される。コマンドライン上は-hLABELとし、templateでは階層にあわせてLABEL-1、LABEL-2とするのが有効。このような階層ラベルはtemplateの中でそれぞれ1組ずつのみ認められている。

### 【注意1】

デフォルトでは@はヌル文字列として置換される。文字を変更するには-dオプションを使う。-dオプションは-hや-lオプションの後に指定しなくてはならない。

```
$ mojihame -dx template data

$ mojihame -lLABEL -dx template data

$ mojihame -hLABEL -dx template data
```

## 【注意2】 アンダースコアデータの扱い

データの中の\_mojihame実行後には\_に置換される。 エスケープのない\_は半角スペースに置換される。

```
$ cat template
<input type="text" value="%1" />
<input type="text" value="%2" />
<input type="text" value="%3" />
$

$ cat data
usp_lab
usp_lab
-----
$

$ mojihame template data
<input type="text" value="usp lab" />
<input type="text" value="usp_lab" />
<input type="text" value="  _" />
$
```

## 【備考】

1. ファイル名に-を指定したときは標準入力からのデータを期待する。templateもdataのいずれかを指定できる。
2. -dオプションは文字列を指定しない場合にはdataの値をそのまますべてtemplateにはめ込む。

## 【関連項目】

filename(1)、formhame(1)

## nameread(1)

### 【名前】

nameread : ネーム形式(5)のファイル読み込み

### 【書式】

```
Usage  : nameread [-el] <name> <namefile>
Option : -d<c>      空白を置換
        : -i<string> ヌルデータの初期化
Version : Thu Dec 15 13:46:38 JST 2011
```

### 【説明】

ネームファイル(5)からnameを指定して値を読み取る。<namefile>が-あるいは指定がないときは標準入力よりデータを受け取る。

-lオプションでnameも合わせて出力する。-eオプションで正規表現を指定でき、-dオプションで空白データを置換する文字を指定する(指定がない場合は空白データを削除する)。-iオプションでヌルデータの初期化を実施する。

【例】

```
$ cat namefile
A usp
B usp laboratory
C
$
```

通常の出力

```
$ nameread A namefile
usp
$
```

ネーム形式(5)のデータは空白を含む

```
$ nameread B namefile
usp laboratory
$
```

空白を変換したいときは-dオプションを使用

```
$ nameread -d_ B namefile
usp_laboratory
$
```

空白を削除したいときは-dオプションを引数なしで指定

```
$ nameread -d B namefile
usplaboratory
$
```

名前のみ存在している場合には空行を出力

```
$ nameread C namefile

$
```

マルチデータを初期化する場合は-iオプションを使用

```
$ nameread -i @@@ C namefile
@@@
$
```

名前が存在しない場合には何も出力しない

```
$ nameread D namefile
$
```

名前も一緒に出力するには-lオプションを使用

```
$ nameread -l A namefile
A usp
$
```

-eオプションを指定することで正規表現を利用可能



```
$ cat namefile2
```

```
A01 1
```

```
A02 2
```

```
A03 3
```

```
A10 5
```

```
$
```

```
$ nameread -e '^A0[0-9]$\ ' namefile2
```

```
1
```

```
2
```

```
3
```

```
$
```

```
$ nameread -e1 '^A0[0-9]$\ ' namefile2
```

```
A01 1
```

```
A02 2
```

```
A03 3
```

```
$
```

【関連項目】

cgi-name(1)、mime-read(1)、ネーム形式(5)

## plus(1)

### 【名前】

plus : 引数の足し算

### 【書式】

```
Usage  : plus v1 v2 ...
Version : Thu Dec 15 13:46:38 JST 2011
```

### 【説明】

v1 + v2 + v3 ...の値を出力する。

### 【例】

```
$ plus 1 2 3 4
10
$
```

```
$ plus 1.21 2.345 -2.524
1.031
$
```

### 【例】

パイプ処理の直後、パイプ各行のコマンドの結果ステータスを足し算して、0ならばすべて正常終了、異なればエラー終了である、といった判定処理に利用される。この使い方は\${PIPESTATUS[@]}が提供されているbashなどに限定される。

```
$ comman1 | command2 | command3 | ... > result
```

```
$ [ $plus ${PIPESTATUS[@]} -ne 0 ] && exit
```

### 【関連項目】

kasan(1)、ratio(1)、sm2(1)、sm4(1)、sm5(1)、ysum(1)

### rank(1)

#### 【名前】

rank : 行に階数を追加

#### 【書式】

```
Usage : rank ref=<ref> key=<key> <file>
Version : Thu Dec 15 13:46:38 JST 2011
```

#### 【説明】

fileの各行に階数を追加する。ref=<ref>で参照キーフィールドを指定した場合、参照キーフィールドが変化した段階で階数を1に初期化する。key=<key>で値フィールドを指定した場合、値が同じ間と同じ階数を追加する。

#### 【例1】基本パターン

引数を何も指定しないときは、単純に行番号を追加する。

```
$ cat data
JPN 杉山_____ 26
JPN 崎村_____ 27
JPN 梶川_____ 27
JPN 柳本_____ 30
USA BOB_____ 25
USA GEROGE_____ 29
USA SAM_____ 29
USA TOM_____ 35
$

$ rank data
1 JPN 杉山_____ 26
2 JPN 崎村_____ 27
3 JPN 梶川_____ 27
4 JPN 柳本_____ 30
5 USA BOB_____ 25
6 USA GEROGE_____ 29
7 USA SAM_____ 29
8 USA TOM_____ 35
$
```

#### 【例2】ref指定

ref=<ref>で参照フィールドを指定したときは、参照フィールドが変化した行で、階数を1に初期化する。ref=1/2 ref=3@2/5 などの連続・不連続のフィールドを指定することも可能。

```
$ rank ref=1 data
1 JPN 杉山_____ 26
2 JPN 崎村_____ 27
3 JPN 梶川_____ 27
4 JPN 柳本_____ 30
1 USA BOB_____ 25
2 USA GEROGE_____ 29
3 USA SAM_____ 29
4 USA TOM_____ 35
$
```

### 【例3】key指定

key=<key>で値フィールドを1つだけ指定できる。この場合、値が同じフィールドを持つ行には同じ階数が追加される。次の行には、はじめから数えた階数が追加される。

```
$ rank key=3 data
1 JPN 杉山_____ 26
2 JPN 崎村_____ 27
2 JPN 梶川_____ 27
4 JPN 柳本_____ 30
5 USA BOB_____ 25
6 USA GEROG_____ 29
6 USA SAM_____ 29
8 USA TOM_____ 35
$
```

### 【例4】refとkey指定

参照キーが変化するところで、行番号が初期化され、かつ同じ値フィールドを持つ行には同じ階数が追加される。

```
$ rank ref=1 key=3 data
1 JPN 杉山_____ 26
2 JPN 崎村_____ 27
2 JPN 梶川_____ 27
4 JPN 柳本_____ 30
1 USA BOB_____ 25
2 USA GEROG_____ 29
2 USA SAM_____ 29
4 USA TOM_____ 35
$
```

### 【注意】

入力ファイルがあるいは指定がない場合には、標準入力のデータが使用される。

同じ階数を追加するかどうかは、値が文字列として同じかどうかで判断しており、数字としては判定していない。

### 【関連項目】

count(1)、gyo(1)、juni(1)、retu(1)

## ratio(1)

### 【名前】

ratio : 構成比算出

### 【書式】

```
Usage : ratio key=<key> <file>
Option : ref=<ref>           ←参照キーの指定
        : -<s>               ←小数点以下の精度
        : +<n>h              ←先頭<n>行をヘッダーとみなす
Version : Thu Dec 15 13:46:38 JST 2011
```

### 【説明】

引数のファイルまたは標準入力のテキストデータに対し、key=で指定したフィールドに対して全行の合計に対する各フィールドの値の構成比を算出し、次のフィールドに挿入して出力する。

### 【例1】

```
$ cat data
a店 1日目 103 62           ←店名 日付 売数 客数
b店 1日目 210 113
c店 1日目 81 52
d店 1日目 75 48
e店 1日目 211 140
$
```

売数(第3フィールド)の店別構成比を求めて売数の次のフィールドに挿入するには、次のようにコマンドを実行する。

```
$ ratio key=3 data | keta
a店 1日目 103 15.1 62      ←第4フィールドが構成比になっている
b店 1日目 210 30.9 113
c店 1日目 81 11.9 52
d店 1日目 75 11.0 48
e店 1日目 211 31.0 140
$
```

【例2】 refオプション

ref=で指定したフィールドをキーフィールドとし、キーフィールドの値が同じ行の中で構成比を計算して出力する。なお、キーフィールドはあらかじめ整列されていることが条件となる。

```
$ cat data
a店 1日目 103 62          ←店名 日付 売数 客数
a店 2日目 157 94
a店 3日目 62 30
b店 1日目 210 113
b店 2日目 237 121
b店 3日目 150 82
c店 1日目 81 52
c店 2日目 76 49
c店 3日目 38 21
d店 1日目 75 48
d店 2日目 72 42
d店 3日目 34 19
e店 1日目 211 140
e店 2日目 149 91
e店 3日目 120 73
$
```

各日(第2フィールド)毎に店別の売数(第3フィールド)の構成比を求める。

```
$ sort -k2,2 -k1,1 data | ratio ref=2 key=3 | keta
a店 1日目 103 15.1 62      ←第4フィールドが1日目における構成比
b店 1日目 210 30.9 113
c店 1日目 81 11.9 52
d店 1日目 75 11.0 48
e店 1日目 211 31.0 140
a店 2日目 157 22.7 94      ←第4フィールドが2日目における構成比
b店 2日目 237 34.3 121
c店 2日目 76 11.0 49
d店 2日目 72 10.4 42
e店 2日目 149 21.6 91
a店 3日目 62 15.3 30      ←第4フィールドが3日目における構成比
b店 3日目 150 37.1 82
c店 3日目 38 9.4 21
d店 3日目 34 8.4 19
e店 3日目 120 29.7 73
$
```

【例3】 +hオプション

+<n>hオプションを付けると、先頭の<n>行を飛ばして構成比を計算する。先頭行に既に項目名などのヘッダーが付与されているデータを計算する場合などに使用する。新たに増える構成比のフィールドのヘッダーは@となる。

```
$ cat data
店 日付 売数 客数          ←項目名のヘッダー
a店 1日目 103 62
b店 1日目 210 113
c店 1日目 81 52
d店 1日目 75 48
e店 1日目 211 140
$
```

先頭行を飛ばして売数(第3フィールド)の店別構成比を求めて売数の次のフィールドに挿入する。

```
$ ratio +h key=3 data | keta
店 日付 売数 @ 客数      ←先頭行はそのまま(計算しない)
a店 1日目 103 15.1 62    ←第4フィールドが構成比になっている
b店 1日目 210 30.9 113
c店 1日目 81 11.9 52
d店 1日目 75 11.0 48
e店 1日目 211 31.0 140
$
```

【例4】 -<s>オプション

-<数値>で構成比の小数点以下の精度を指定できる。

```
$ cat data
a店 1日目 103 62          ←店名 日付 売数 客数
b店 1日目 210 113
c店 1日目 81 52
d店 1日目 75 48
e店 1日目 211 140
$
```

売数(第3フィールド)の店別構成比を小数点3桁まで求める。

```
$ ratio -3 key=3 data | keta
a店 1日目 103 15.147 62
b店 1日目 210 30.882 113
c店 1日目 81 11.912 52
d店 1日目 75 11.029 48
e店 1日目 211 31.029 140
$
```

【関連項目】

kasan(1)、plus(1)、sm2(1)、sm4(1)、sm5(1)、ysum(1)

## retu(1)

### 【名前】

retu : 列をカウント

### 【書式】

Usage : retu [-f] <file> ...  
Version : Fri Oct 21 11:26:06 JST 2011

### 【説明】

引数のファイルまたは標準入力のテキストデータの列数(フィールド数)をカウントして出力する。

### 【例1】

```
$ cat data
0000000 浜地_____ 50 F 91 59 20 76 54
0000001 鈴田_____ 50 F 46 39 8 5 21
0000003 杉山_____ 26 F 30 50 71 36 30
0000004 白土_____ 40 M 58 71 20 10 6
0000005 崎村_____ 50 F 82 79 16 21 80
0000007 梶川_____ 42 F 50 2 33 15 62
$

$ retu data
9
$
```



### 【例2】

複数のファイルの列数(フィールド数)を一度にカウントできる。同一ファイル内に違う列数の行があると、列数が変化した時点で列数を出力する。-fオプションをつけるとファイル名も出力される。

```
$ cat data1
1 file1
2 file1
3 file1
$
```

```
$ cat data2
1 file2
2
$
```

```
$ cat data3
1
2 file3
3
4 file3
$
```

```
$ retu data1 data2 data3
2
2
1
1
2
1
2
$
```

### 【例3】

-fオプションを使用するとファイル名と行数をそれぞれ表示する。

```
$ retu -f data1 data2 data3
data1 2
data2 2
data2 1
data3 1
data3 2
data3 1
data3 2
$
```

### 【備考】

ファイル名に-を指定すると標準入力からのデータ読み込みを期待する。

```
$ cat data2 | retu -f data1 - data3
```

### 【関連項目】

count(1)、gyo(1)、juni(1)、rank(1)

## self(1)

### 【名前】

self : 指定したフィールドのデータ取り出し

### 【書式】

```
Usage : self f1 f2 ... file
       : self -d f1 f2 ... string
Version : Wed Dec 14 15:20:01 JST 2011
       : OpenUSP (LINUX+FREEBSD/PYTHON2.4/UTF-8)
```

### 【説明】

fileから指定したフィールドのデータだけを取り出して出力する(self=select field)。fileの指定がないとき、あるいはファイル名が-の時は、標準入力よりデータを取り込む。-dオプションが指定された場合、引数の文字列に対して処理が適用される。

### 【例1】

テキストデータの第4フィールドと第2フィールドを取り出して出力する。

```
$ cat data
0000000 浜地_____ 50 F 91 59 20 76 54
0000001 鈴田_____ 50 F 46 39 8 5 21
0000003 杉山_____ 26 F 30 50 71 36 30
0000004 白土_____ 40 M 58 71 20 10 6
0000005 崎村_____ 50 F 82 79 16 21 80
$
```

```
$ self 4 2 data
F 浜地_____
F 鈴田_____
F 杉山_____
M 白土_____
F 崎村_____
$
```

【例2】

self(1)はフィールド中の部分を切り出して出力することもできる。処理としてはawk(1)のsubstr関数と同じ処理となる。

たとえば第1フィールドの4文字目以降を出力するには次のようにコマンドを実行する。

```
$ self 1.4 2 data
0000 浜地_____
0001 鈴田_____
0003 杉山_____
0004 白土_____
0005 崎村_____
$
```

第2フィールドの1文字目から半角4文字分(全角2文字分)出力する。

```
$ self 2.1.4 3 data
浜地 50
鈴田 50
杉山 26
白土 40
崎村 50
$
```

【例3】

0で行全体を指定できる。

```
$ self 0 data
0000000 浜地_____ 50 F 91 59 20 76 54
0000001 鈴田_____ 50 F 46 39 8 5 21
0000003 杉山_____ 26 F 30 50 71 36 30
0000004 白土_____ 40 M 58 71 20 10 6
0000005 崎村_____ 50 F 82 79 16 21 80
$

$ self 4 0 data
F 0000000 浜地_____ 50 F 91 59 20 76 54
F 0000001 鈴田_____ 50 F 46 39 8 5 21
F 0000003 杉山_____ 26 F 30 50 71 36 30
M 0000004 白土_____ 40 M 58 71 20 10 6
F 0000005 崎村_____ 50 F 82 79 16 21 80
$
```

【例4】

連続したフィールドも記述できる。

```
$ self 2/5 data
浜地_____ 50 F 91
鈴田_____ 50 F 46
杉山_____ 26 F 30
白土_____ 40 M 58
崎村_____ 50 F 82
$
```

【例5】

NF(現在の行のフィールド数)を使用することができる。

```
$ self 1 NF-3 NF data
0000000 59 54
0000001 39 21
0000003 50 30
0000004 71 6
0000005 79 80
$
```

【例6】

複数の空白があるフィールドを1つの空白にすることができる。

```
$ cat data2
a b
c d e
$

$ self 1/NF data2
a b
c d e
$
```

【例7】

ダイレクトモードを使用して文字列を加工することができる。

```
$ self -d 1.1.4 "20070401 12345"
2007
$
```

【例8】

selfの切り出し指定は画面表示長の半角文字分を1として扱う。

```
$ echo ｱｲｳｴｶｷｸｻ | self 1.3.4
ｳｴｶ
$
```

【注意】

入力ファイルが改行文字で終了していない場合、つまり行が尻切れになっている場合はファイル末尾に改行文字が追加されて完結した行になる。

#### 【バグ】

数字名のファイルを使用してはならない。もし10という名前のファイルが存在するとself 1 10はファイル名が10であるファイルから第1フィールドを抽出するという動作をして、標準入力ファイルから第1および第10フィールドを抽出するという動作をしなくなる。ただし2のように数字1文字だけを指定した場合は、そのファイルが存在しても、それは無視してフィールドとして扱う。

また、文字列切り出し指定において、切り取りや長さが文字の区切りに一致しないときにはエラーとなる。

```
$ echo U S P 研究所 | self 1.2.4
Errorself] : Rec 1. エラー
$
```

フィールドの幅より大きい開始位置を指定してもエラーとなる。

```
$ echo U S P 研究所 | self 1.20
Errorself] : Rec 1. エラー
$
```

#### 【関連項目】

delf(1)

sm2(1)

【名前】

sm2 : キー単位での値の集計

【書式】

Usage : sm2 [+count] <k1> <k2> <s1> <s2> <file>  
Version : Thu Dec 15 13:46:38 JST 2011

【説明】

<file>ファイルから、キーが同じ行の各フィールドの値を集計する。k1で指定したフィールドからk2で指定したフィールドまでをキーとして、s1で指定したフィールドからs2で指定したフィールドまでの各フィールドを集計する。同じキーをもつ行は1行に集計されて出力される。なお、キーまたは集計フィールドいずれの範囲にも指定していないフィールドについては出力はされない。

【例1】

```
$ cat data
0001 新橋店 20060201 91 59 20 76 54      ←店別番号 店名 日付 商品別売数
0001 新橋店 20060202 46 39 8 5 21
0001 新橋店 20060203 82 0 23 84 10
0002 池袋店 20060201 30 50 71 36 30
0002 池袋店 20060202 78 13 44 28 51
0002 池袋店 20060203 58 71 20 10 6
0003 新宿店 20060201 82 79 16 21 80
0003 新宿店 20060202 50 2 33 15 62
0003 新宿店 20060203 52 91 44 9 0
0004 上野店 20060201 60 89 33 18 6
0004 上野店 20060202 95 60 35 93 76
0004 上野店 20060203 92 56 83 96 75
$
```

店別の売れ数の合計を出力させる。第1フィールドから第2フィールドをキーとして第4フィールドから第8フィールドを集計する。

```
$ sm2 1 2 4 8 data
0001 新橋店 219 98 51 165 85
0002 池袋店 166 134 135 74 87
0003 新宿店 184 172 93 45 142
0004 上野店 247 205 151 207 157
$
```

【例2】

+countオプションはキー毎の集計すると同時に、元データの各キー毎の行の行数をキーの直後に挿入出力する。

```
$ cat data
1111 3
1111 5
1111 2
2222 3
2222 10
3333 4
3333 8
3333 9
3333 6
$
```

キーが同一の行を出力する。

```
$ sm2 +count 1 1 2 2 data
1111 3 10
2222 2 13
3333 4 27
$
```

【例3】

sm2(1)は小数点がある場合、各キー、各フィールド毎に一番精度の高い数に合わせて出力する。

```
$ cat data3
a 1.4 2.55
a 2 4
b 1.33 2.1
b 5.222 3.12
$
```

```
$ sm2 1 1 2 3 data3
a 3.4 6.55
b 6.552 5.22
$
```

【例4】

キーを0 0に指定すると、集計フィールドの総合計を出力する。

```
$ cat data4
a 1
b 2
c 3
$
```

```
$ sm2 0 0 2 2 data4
6
$
```

指定したディレクトリに存在するファイルのサイズを合計するには次のようにコマンドを実行する。

```
$ ls -l directory | sm2 0 0 5 5
```

【関連項目】

kasan(1)、plus(1)、ratio(1)、sm4(1)、sm5(1)、ysum(1)





## sm4(1)

### 【名前】

sm4 : 小計、中計

### 【書式】

```
Usage   : sm4 [+h] <k1> <k2> <d1> <d2> <s1> <s2> <file>
Version : Thu Dec 15 13:46:38 JST 2011
        : UTF-8
```

### 【説明】

<file>ファイルにおけるサブキーによる小計や中計を計算する。k1で指定したフィールドからk2で指定したフィールドまでを主キー、d1からd2までのフィールドがサブキー、s1からs2までのフィールドまでのフィールドを集計フィールドとして、主キーの値が同じキーをサブキーのフィールドを飛ばして集計フィールドを集計して各主キーの行の最終行に挿入する。

集計行におけるサブキーのフィールドは@で補完される。小計および中計を段階的に付与するためにsm4(1)を複数回使用する場合には、サブキーに@が含まれる行は集計から除外される。キーまたは集計フィールドいずれの範囲にも指定していないフィールドについては出力は実行されない。

【例1】

```
$ cat data
01 埼玉県 01 さいたま市 91 59 20 76 54
01 埼玉県 02 川崎市 46 39 8 5 21
01 埼玉県 03 熊谷市 82 0 23 84 10
02 東京都 04 新宿区 30 50 71 36 30
02 東京都 05 中央区 78 13 44 28 51
02 東京都 06 港区 58 71 20 10 6
02 東京都 07 八王子市 82 79 16 21 80
02 東京都 08 立川市 50 2 33 15 62
03 千葉県 09 千葉市 52 91 44 9 0
03 千葉県 10 市川市 60 89 33 18 6
03 千葉県 11 柏市 95 60 35 93 76
04 神奈川県 12 横浜市 92 56 83 96 75
04 神奈川県 13 川崎市 30 12 32 44 19
04 神奈川県 14 厚木市 48 66 23 71 24
$
```

←各都市での調査データ

県別に小計を出す。

```
$ sm4 1 2 3 4 5 NF data
01 埼玉県 01 さいたま市 91 59 20 76 54
01 埼玉県 02 川崎市 46 39 8 5 21
01 埼玉県 03 熊谷市 82 0 23 84 10
01 埼玉県 @@ @@@@ 173 59 43 160 64
02 東京都 04 新宿区 30 50 71 36 30
02 東京都 05 中央区 78 13 44 28 51
02 東京都 06 港区 58 71 20 10 6
02 東京都 07 八王子市 82 79 16 21 80
02 東京都 08 立川市 50 2 33 15 62
02 東京都 @@ @@@@ 248 213 151 95 167
03 千葉県 09 千葉市 52 91 44 9 0
03 千葉県 10 市川市 60 89 33 18 6
03 千葉県 11 柏市 95 60 35 93 76
03 千葉県 @@ @@@@ 207 240 112 120 82
04 神奈川県 12 横浜市 92 56 83 96 75
04 神奈川県 13 川崎市 30 12 32 44 19
04 神奈川県 14 厚木市 48 66 23 71 24
04 神奈川県 @@ @@@@ 170 134 138 211 118
$
```

【例2】

小計と中計を出す。

```
$ cat data
01 関東地区 01 埼玉県 01 さいたま市 91 59 20 76 54
01 関東地区 01 埼玉県 03 熊谷市 82 0 23 84 10
01 関東地区 02 東京都 04 新宿区 30 50 71 36 30
01 関東地区 02 東京都 05 中央区 78 13 44 28 51
01 関東地区 02 東京都 07 八王子市 82 79 16 21 80
02 関西地区 01 大阪府 01 大阪市 91 59 20 76 54
02 関西地区 01 大阪府 02 八尾市 46 39 8 5 21
02 関西地区 01 大阪府 03 川西市 82 0 23 84 10
02 関西地区 02 兵庫県 04 神戸市 30 50 71 36 30
02 関西地区 02 兵庫県 05 姫路市 78 13 44 28 51
02 関西地区 02 兵庫県 06 明石市 58 71 20 10 6
02 関西地区 02 兵庫県 07 加古川市 82 79 16 21 80
02 関西地区 02 兵庫県 08 芦屋市 50 2 33 15 62
02 関西地区 03 京都府 09 京都市 52 91 44 9 0
02 関西地区 03 京都府 10 舞鶴市 60 89 33 18 6
$

$ sm4 1 4 5 6 7 11 data > data2
$ cat data2
01 関東地区 01 埼玉県 01 さいたま市 91 59 20 76 54
01 関東地区 01 埼玉県 03 熊谷市 82 0 23 84 10
01 関東地区 01 埼玉県 @@ @@@@@@@@@@ 173 59 43 160 64
01 関東地区 02 東京都 04 新宿区 30 50 71 36 30
01 関東地区 02 東京都 05 中央区 78 13 44 28 51
01 関東地区 02 東京都 07 八王子市 82 79 16 21 80
01 関東地区 02 東京都 @@ @@@@@@@@@@ 190 142 131 85 161
02 関西地区 01 大阪府 01 大阪市 91 59 20 76 54
02 関西地区 01 大阪府 02 八尾市 46 39 8 5 21
02 関西地区 01 大阪府 03 川西市 82 0 23 84 10
02 関西地区 01 大阪府 @@ @@@@@@@@@@ 219 98 51 165 85
02 関西地区 02 兵庫県 04 神戸市 30 50 71 36 30
02 関西地区 02 兵庫県 05 姫路市 78 13 44 28 51
02 関西地区 02 兵庫県 06 明石市 58 71 20 10 6
02 関西地区 02 兵庫県 07 加古川市 82 79 16 21 80
02 関西地区 02 兵庫県 08 芦屋市 50 2 33 15 62
02 関西地区 02 兵庫県 @@ @@@@@@@@@@ 298 215 184 110 229
02 関西地区 03 京都府 09 京都市 52 91 44 9 0
02 関西地区 03 京都府 10 舞鶴市 60 89 33 18 6
02 関西地区 03 京都府 @@ @@@@@@@@@@ 112 180 77 27 6
$
```

```

$ sm4 1 2 3 6 7 11 data2
01 関東地区 01 埼玉県 01 さいたま市 91 59 20 76 54
01 関東地区 01 埼玉県 03 熊谷市 82 0 23 84 10
01 関東地区 01 埼玉県 @@ @@@@@@@@@@ 173 59 43 160 64
01 関東地区 02 東京都 04 新宿区 30 50 71 36 30
01 関東地区 02 東京都 05 中央区 78 13 44 28 51
01 関東地区 02 東京都 07 八王子市 82 79 16 21 80
01 関東地区 02 東京都 @@ @@@@@@@@@@ 190 142 131 85 161
01 関東地区 @@ @@@@@@ @@ @@@@@@@@@@ 363 201 174 245 225
02 関西地区 01 大阪府 01 大阪市 91 59 20 76 54
02 関西地区 01 大阪府 02 八尾市 46 39 8 5 21
02 関西地区 01 大阪府 03 川西市 82 0 23 84 10
02 関西地区 01 大阪府 @@ @@@@@@@@@@ 219 98 51 165 85
02 関西地区 02 兵庫県 04 神戸市 30 50 71 36 30
02 関西地区 02 兵庫県 05 姫路市 78 13 44 28 51
02 関西地区 02 兵庫県 06 明石市 58 71 20 10 6
02 関西地区 02 兵庫県 07 加古川市 82 79 16 21 80
02 関西地区 02 兵庫県 08 芦屋市 50 2 33 15 62
02 関西地区 02 兵庫県 @@ @@@@@@@@@@ 298 215 184 110 229
02 関西地区 03 京都府 09 京都市 52 91 44 9 0
02 関西地区 03 京都府 10 舞鶴市 60 89 33 18 6
02 関西地区 03 京都府 @@ @@@@@@@@@@ 112 180 77 27 6
02 関西地区 @@ @@@@@@ @@ @@@@@@@@@@ 629 493 312 302 320
$

```

【例3】

サブキーがない場合には次のような実行結果が得られる。

```
$ cat data
埼玉県 91 59 20 76 54
埼玉県 46 39 8 5 21
埼玉県 82 0 23 84 10
東京都 30 50 71 36 30
東京都 78 13 44 28 51
東京都 58 71 20 10 6
東京都 82 79 16 21 80
東京都 50 2 33 15 62
千葉県 52 91 44 9 0
千葉県 60 89 33 18 6
千葉県 95 60 35 93 76
神奈川県 92 56 83 96 75
神奈川県 30 12 32 44 19
神奈川県 48 66 23 71 24
$

$ sm4 1 1 x x 2 6 data
埼玉県 91 59 20 76 54
埼玉県 46 39 8 5 21
埼玉県 82 0 23 84 10
@@@@ 219 98 51 165 85
東京都 30 50 71 36 30
東京都 78 13 44 28 51
東京都 58 71 20 10 6
東京都 82 79 16 21 80
東京都 50 2 33 15 62
@@@@ 298 215 184 110 229
千葉県 52 91 44 9 0
千葉県 60 89 33 18 6
千葉県 95 60 35 93 76
@@@@ 207 240 112 120 82
神奈川県 92 56 83 96 75
神奈川県 30 12 32 44 19
神奈川県 48 66 23 71 24
@@@@ 170 134 138 211 118
$
```

【関連項目】

kasan(1)、plus(1)、ratio(1)、sm2(1)、sm5(1)、ysum(1)

sm5(1)

【名前】

sm5 : 大計

【書式】

Usage : sm5 [+h] <k1> <k2> <s1> <s2> <file>  
 Version : Thu Dec 15 13:46:38 JST 2011  
 : UTF-8

【説明】

<input>ファイルに大計(全行の合計値)行を追加する。k1で指定したフィールドからk2で指定したフィールドまでをキー、s1からs2までのフィールドを集計フィールドとして、キー以外の集計各フィールドの全合計の行を最終行に挿入する。この行のキーフィールドは@で補完される。

sm4(1)で小計、中計を付与されているファイルを処理する場合は、sm4(1)で付与した@の小計および中計行は無視して大計を計算する。

【例1】

```
$ cat data
01 埼玉県 01 さいたま市 91 59 20 76 54          ←各都市での調査データ
01 埼玉県 02 川越市 46 39 8 5 21
01 埼玉県 03 熊谷市 82 0 23 84 10
02 東京都 04 新宿区 30 50 71 36 30
02 東京都 05 中央区 78 13 44 28 51
02 東京都 06 港区 58 71 20 10 6
02 東京都 07 八王子市 82 79 16 21 80
02 東京都 08 立川市 50 2 33 15 62
03 千葉県 09 千葉市 52 91 44 9 0
03 千葉県 10 市川市 60 89 33 18 6
03 千葉県 11 柏市 95 60 35 93 76
04 神奈川県 12 横浜市 92 56 83 96 75
04 神奈川県 13 川崎市 30 12 32 44 19
04 神奈川県 14 厚木市 48 66 23 71 24
$
```

大計を出す。

```
$ sm5 1 4 5 NF data
01 埼玉県 01 さいたま市 91 59 20 76 54
01 埼玉県 02 川越市 46 39 8 5 21
01 埼玉県 03 熊谷市 82 0 23 84 10
02 東京都 04 新宿区 30 50 71 36 30
02 東京都 05 中央区 78 13 44 28 51
02 東京都 06 港区 58 71 20 10 6
02 東京都 07 八王子市 82 79 16 21 80
02 東京都 08 立川市 50 2 33 15 62
03 千葉県 09 千葉市 52 91 44 9 0
03 千葉県 10 市川市 60 89 33 18 6
03 千葉県 11 柏市 95 60 35 93 76
04 神奈川県 12 横浜市 92 56 83 96 75
04 神奈川県 13 川崎市 30 12 32 44 19
04 神奈川県 14 厚木市 48 66 23 71 24
@@ @@@@ @ @@@@@@@ 894 687 485 606 514
$
```

【例2】

sm4(1)で小計よび中計が追加されたデータを処理する場合は次のようになる。

```
$ cat data
01 埼玉県 01 さいたま市 91 59 20 76 54
01 埼玉県 02 川越市 46 39 8 5 21
01 埼玉県 03 熊谷市 82 0 23 84 10
01 埼玉県 @@ @@@@@@@@@@@@@@@@ 173 59 43 160 64
02 東京都 04 新宿区 30 50 71 36 30
02 東京都 05 中央区 78 13 44 28 51
02 東京都 06 港区 58 71 20 10 6
02 東京都 07 八王子市 82 79 16 21 80
02 東京都 08 立川市 50 2 33 15 62
02 東京都 @@ @@@@@@@@@@@@@@@@ 248 213 151 95 167
03 千葉県 09 千葉市 52 91 44 9 0
03 千葉県 10 市川市 60 89 33 18 6
03 千葉県 11 柏市 95 60 35 93 76
03 千葉県 @@ @@@@@@@@@@@@@@@@ 207 240 112 120 82
04 神奈川県 12 横浜市 92 56 83 96 75
04 神奈川県 13 川崎市 30 12 32 44 19
04 神奈川県 14 厚木市 48 66 23 71 24
04 神奈川県 @@ @@@@@@@@@@@@@@@@ 170 134 138 211 118
$
```

```
$ sm5 1 4 5 NF data
01 埼玉県 01 さいたま市 91 59 20 76 54
01 埼玉県 02 川越市 46 39 8 5 21
01 埼玉県 03 熊谷市 82 0 23 84 10
01 埼玉県 @@ @@@@@@@@@@@@@@@@ 173 59 43 160 64
02 東京都 04 新宿区 30 50 71 36 30
02 東京都 05 中央区 78 13 44 28 51
02 東京都 06 港区 58 71 20 10 6
02 東京都 07 八王子市 82 79 16 21 80
02 東京都 08 立川市 50 2 33 15 62
02 東京都 @@ @@@@@@@@@@@@@@@@ 248 213 151 95 167
03 千葉県 09 千葉市 52 91 44 9 0
03 千葉県 10 市川市 60 89 33 18 6
03 千葉県 11 柏市 95 60 35 93 76
03 千葉県 @@ @@@@@@@@@@@@@@@@ 207 240 112 120 82
04 神奈川県 12 横浜市 92 56 83 96 75
04 神奈川県 13 川崎市 30 12 32 44 19
04 神奈川県 14 厚木市 48 66 23 71 24
04 神奈川県 @@ @@@@@@@@@@@@@@@@ 170 134 138 211 118
@@ @@@@@@ @@ @@@@@@@@@@@@ 894 687 485 606 514
$
```

【コラム】 sm系コマンドの合わせ方

データを集計する帳票をTukubaiで記述する場合、同じキーを集計して小計、中計を出して最後に大計を出す、という処理が頻出する。こういった場合はsm2(1)、sm4(1)、sm5(1)を次のように多段に組み合わせ使用することが多い。

```
cat data |
sm2 1 6 7 NF | ←同一キーで集計
sm4 1 4 5 6 7 NF | ←小計
sm4 1 2 3 6 7 NF | ←中計
sm5 1 6 7 NF > result ←大計を出力
```

【関連項目】

kasan(1)、plus(1)、ratio(1)、sm2(1)、sm4(1)、ysum(1)

## tarr(1)

### 【名前】

tarr : 横型のデータを縦型に展開

### 【書式】

```
Usage : tarr [-<n>][-d<str>] <file>
       : tarr [-<n>][-d<str>] num=<n> <file>
Version : Thu Dec 15 13:46:38 JST 2011
```

### 【説明】

指定したファイルの第1フィールドからnum=<n>で指定したフィールドまでのフィールドをキーとして、横に並んでいるフィールドデータを縦に展開して並べ直す。

#### 【例1】基本的な使い方

```
$ cat data
0000000 浜地_____ 50 F 91 59 20 76
0000001 鈴田_____ 50 F 46 39 8 5
0000003 杉山_____ 26 F 30 50 71 36
0000004 白土_____ 40 M 58 71 20 10
$

$ tarr num=4 data          ←第4フィールドまでをキーに縦に展開
0000000 浜地_____ 50 F 91
0000000 浜地_____ 50 F 59
0000000 浜地_____ 50 F 20
0000000 浜地_____ 50 F 76
0000001 鈴田_____ 50 F 46
0000001 鈴田_____ 50 F 39
0000001 鈴田_____ 50 F 8
0000001 鈴田_____ 50 F 5
0000003 杉山_____ 26 F 30
0000003 杉山_____ 26 F 50
0000003 杉山_____ 26 F 71
0000003 杉山_____ 26 F 36
0000004 白土_____ 40 M 58
0000004 白土_____ 40 M 71
0000004 白土_____ 40 M 20
0000004 白土_____ 40 M 10
$
```

#### 【例2】

-<n>オプションを使用すると<n>フィールドごとに縦展開する。

```
$ tarr -2 num=4 data      ←2フィールド毎に縦展開する。
0000000 浜地_____ 50 F 91 59
0000000 浜地_____ 50 F 20 76
0000001 鈴田_____ 50 F 46 39
0000001 鈴田_____ 50 F 8 5
0000003 杉山_____ 26 F 30 50
0000003 杉山_____ 26 F 71 36
0000004 白土_____ 40 M 58 71
0000004 白土_____ 40 M 20 10
$
```



【例3】

num=<n>を指定しないと単純に縦展開する。

```
$ cat data3  
1 2 3 4  
5 6  
$
```

```
$ tarr data3  
1  
2  
3  
4  
5  
6  
$
```

【例4】

-d<str>オプションを使うことで縦展開するフィールドのフィールドセパレータを指定できる。num=<n>で指定されるキーフィールドは空白区切りである必要がある。

```
$ cat data
0000000 浜地_____ 50 F 91@59@20@76@54
0000001 鈴田_____ 50 F 46@39@8@5@21
0000003 杉山_____ 26 F 30@50@71@36@30
0000004 白土_____ 40 M 58@71@20@10@6
$
```

```
$ tarr -d@ num=4 data
0000000 浜地_____ 50 F 91
0000000 浜地_____ 50 F 59
0000000 浜地_____ 50 F 20
0000000 浜地_____ 50 F 76
0000000 浜地_____ 50 F 54
0000001 鈴田_____ 50 F 46
0000001 鈴田_____ 50 F 39
0000001 鈴田_____ 50 F 8
0000001 鈴田_____ 50 F 5
0000001 鈴田_____ 50 F 21
0000003 杉山_____ 26 F 30
0000003 杉山_____ 26 F 50
0000003 杉山_____ 26 F 71
0000003 杉山_____ 26 F 36
0000003 杉山_____ 26 F 30
0000004 白土_____ 40 M 58
0000004 白土_____ 40 M 71
0000004 白土_____ 40 M 20
0000004 白土_____ 40 M 10
0000004 白土_____ 40 M 6
$
```

```
$ cat data2
0001 1n2n3
0002 4n5n6
$
```

```
$ tarr -d'n' num=1 data2
0001 1
0001 2
0001 3
0002 4
0002 5
0002 6
$
```

【関連項目】

yarr(1)

## tateyoko(1)

### 【名前】

tateyoko : 縦テキストを横テキストへ変換

### 【書式】

Usage : tateyoko <file>  
Version : Thu Dec 15 13:46:38 JST 2011

### 【説明】

<file>の行を列に縦横変換する。出力する<file>ファイルの指定がないとき、あるいはファイル名が-の時は、標準入力ファイルを取り込む。

### 【例1】

```
$ cat data
1 2 3
A B C
4 5 6
$

$ tateyoko data
1 A 4
2 B 5
3 C 6
$
```

### 【例2】

map(1)の出力を縦横変換する。

```
$ cat data
A 10/01 1
A 10/02 2
A 10/03 3
B 10/01 4
B 10/02 5
B 10/03 6
$

$ map num=1 data | keta
* 10/01 10/02 10/03
A   1   2   3
B   4   5   6
$

$ map num=1 data | tateyoko | keta
*   A B
10/01 1 4
10/02 2 5
10/03 3 6
$
```

### 【備考】

tateyoko(1)は指定ファイルを一旦すべてメモリへ展開する。

**【関連項目】**

map(1)、unmap(1)

## tcat(1)

### 【名前】

tcat : ファイルを縦へ並べ替え

### 【書式】

Usage : tcat [-n] <file1> <file2>...  
Version : Thu Dec 15 13:46:38 JST 2011

### 【説明】

引数で指定した複数のファイルを縦に並べて連結して出力する。

### 【例1】

```
$ cat file1
---試験結果---
氏名No 氏名      年齢 性別  A  B  C  D  E
$

$ cat file2
0000008 角屋_____ 50   F 52 91 44  9  0
0000009 米原_____ 68   F 60 89 33 18  6
0000010 柳本_____ 50   F 95 60 35 93 76
0000011 安武_____ 55   F 92 56 83 96 75
$

$ tcat file1 file2
---試験結果---
氏名No 氏名      年齢 性別  A  B  C  D  E
0000008 角屋_____ 50   F 52 91 44  9  0
0000009 米原_____ 68   F 60 89 33 18  6
0000010 柳本_____ 50   F 95 60 35 93 76
0000011 安武_____ 55   F 92 56 83 96 75
$
```

### 【例2】

-<n>オプションを使用すると、ファイルを<n>-で指定した行数分だけ間をあけて連結することができる。

2行あけて連結するには次のようにコマンドを実行する。

```
$ tcat -2 file1 file2
---試験結果---
氏名No 氏名      年齢 性別  A  B  C  D  E

0000008 角屋_____ 50   F 52 91 44  9  0
0000009 米原_____ 68   F 60 89 33 18  6
0000010 柳本_____ 50   F 95 60 35 93 76
0000011 安武_____ 55   F 92 56 83 96 75
$
```

### 【例3】

標準入力から読み込む場合、標準入力側のテキストを引数-で指定する。

```
$ cat file1 | tcat - file2
```

**【関連項目】**

ctail(1)、getfirst(1)、getlast(1)、ycat(1)

## unmap(1)

### 【名前】

unmap : 縦キー/横キー形式の表データを縦型ファイルに変換

### 【書式】

```
Usage : map      [-<l>] num=<n>x<m> <file>
       : map +yarr [-<l>] num=<n>x<m> <file>
       : map +arr  [-<l>] num=<n>x<m> <file>
Version : Sat Dec 10 21:04:17 JST 2011
       : OpenUSP (LINUX+FREEBSD/PYTHON2.4/UTF-8)
```

### 【説明】

map(1)と逆の動作をする。

### 【例1】

表データを縦型ファイルに変換する。num=<n>の値は縦キーのフィールド数を表す。

```
$ cat data
* * 01/01 01/02
001 a店 103 157
002 b店 210 237
$

$ unmap num=2 data | keta
001 a店 01/01 103
001 a店 01/02 157
002 b店 01/01 210
002 b店 01/02 237
$
```

### 【例2】

データ部が複数種ある場合は、複数列にマッピングする。

```
$ cat data
* * 1日目 2日目 3日目 4日目 5日目 6日目 7日目
a店 A 103 157 62 131 189 350 412
a店 B 62 94 30 84 111 20 301
b店 A 210 237 150 198 259 421 589
b店 B 113 121 82 105 189 287 493
c店 A 81 76 38 81 98 109 136
c店 B 52 49 21 48 61 91 110
d店 A 75 72 34 74 91 98 101
d店 B 48 42 19 43 51 69 90
$

$ unmap num=1 data | keta
a店 1日目 103 62
a店 2日目 157 94
a店 3日目 62 30
a店 4日目 131 84
.
.
d店 6日目 98 69
d店 7日目 101 90
$
```

【関連項目】

map(1)、tateyoko(1)



### up3(1)

#### 【名前】

up3 : 2つのファイルを同一キーフィールドでマージ

#### 【書式】

Usage : up3 key=<key> <master> <tran>  
Version : Thu Dec 15 13:46:38 JST 2011

#### 【説明】

masterとtranの各行をkey=で指定されたフィールド値で比較し、同じキーフィールドを持つ行をmasterの該当行の下に挿入して出力する。masterもtranもキーとなるフィールドは整列されている必要がある。

#### 【例1】

```
$ cat master
a店 1日目 103 62
a店 2日目 157 94
a店 3日目 62 30
b店 1日目 210 113
b店 2日目 237 121
b店 3日目 150 82
c店 1日目 81 52
c店 2日目 76 49
c店 3日目 38 21
$

$ cat tran
a店 4日目 131 84
a店 5日目 189 111
b店 4日目 198 105
b店 5日目 259 189
c店 4日目 81 48
c店 5日目 98 61
$

$ up3 key=1 master tran
a店 1日目 103 62
a店 2日目 157 94
a店 3日目 62 30
a店 4日目 131 84
a店 5日目 189 111
b店 1日目 210 113
b店 2日目 237 121
b店 3日目 150 82
b店 4日目 198 105
b店 5日目 259 189
c店 1日目 81 52
c店 2日目 76 49
c店 3日目 38 21
c店 4日目 81 48
c店 5日目 98 61
$
```

複数のキーフィールドを指定する場合はkey=2@1のように@でつないで指定する。

**【関連項目】**

join0(1)、join1(1)、join2(1)、loopj(1)、loopx(1)、マスタファイル(5)、トランザクションファイル(5)

## yarr(1)

### 【名前】

yarr : 縦型のデータを横型に展開

### 【書式】

```
Usage : yarr [-<n>][-d<str>] <file>
       : yarr [-<n>][-d<str>] num=<n> <file>
Version : Thu Dec 15 13:46:38 JST 2011
```

### 【説明】

指定したファイルの第1フィールドからnum=<n>で指定したフィールド数までのフィールドをキーとし、キーが同一の行のフィールドを横に展開して1行として出力する。

### 【例1】 基本的な使い方

```
$ cat data
0000000 浜地_____ 50 F 91
0000000 浜地_____ 50 F 59
0000000 浜地_____ 50 F 20
0000000 浜地_____ 50 F 76
0000001 鈴田_____ 50 F 46
0000001 鈴田_____ 50 F 39
0000001 鈴田_____ 50 F 8
0000001 鈴田_____ 50 F 5
0000003 杉山_____ 26 F 30
0000003 杉山_____ 26 F 50
0000003 杉山_____ 26 F 71
0000003 杉山_____ 26 F 36
0000004 白土_____ 40 M 58
0000004 白土_____ 40 M 71
0000004 白土_____ 40 M 20
0000004 白土_____ 40 M 10
$

$ yarr num=4 data
0000000 浜地_____ 50 F 91 59 20 76
0000001 鈴田_____ 50 F 46 39 8 5
0000003 杉山_____ 26 F 30 50 71 36
0000004 白土_____ 40 M 58 71 20 10
$
```

### 【例2】

-<n>オプションを使うことで<n>個ずつ横展開できる。

```
$ yarr -2 num=4 data > data3
$ cat data3
0000000 浜地_____ 50 F 91 59
0000000 浜地_____ 50 F 20 76
0000001 鈴田_____ 50 F 46 39
0000001 鈴田_____ 50 F 8 5
0000003 杉山_____ 26 F 30 50
0000003 杉山_____ 26 F 71 36
0000004 白土_____ 40 M 58 71
0000004 白土_____ 40 M 20 10
$
```

【例3】

num=<n>を指定しない場合単純に横展開する。

```
$ cat data4
1
2
3
4
$
```

```
$ yarr data4
1 2 3 4
$
```

【例4】

-d<str> オプションは横展開するときのフィールドセパレータを指定する。

```
$ yarr -d@ num=4 data > data2
$ cat data2
0000000 浜地_____ 50 F 91@59@20@76
0000001 鈴田_____ 50 F 46@39@8@5
0000003 杉山_____ 26 F 30@50@71@36
0000004 白土_____ 40 M 58@71@20@10
$
```

```
$ yarr -d'n' num=4 data
0000000 浜地_____ 50 F 91n59n20n76
0000001 鈴田_____ 50 F 46n39n8n5
0000003 杉山_____ 26 F 30n50n71n36
0000004 白土_____ 40 M 58n71n20n10
$
```

【関連項目】

tarr(1)

## ycat(1)

### 【名前】

ycat : ファイルの横連結

### 【書式】

```
Usage   : ycat [-n] file1 file2 ....
Version : Thu Dec 15 13:46:38 JST 2011
        : UTF-8
```

### 【説明】

引数で指定した複数のファイルを横に並べて連結して出力する。各ファイルの形は崩れることはなく、ファイルの見た目そのままに横連結する。

### 【例1】 基本的な使い方

```
$ cat file1
0000000 浜地_____ 50 F
0000001 鈴田_____ 50 F
0000003 杉山_____ 26 F
0000004 白土_____ 40 M
0000005 崎村_____ 50 F
0000007 梶川_____ 42 F
$

$ cat file2
0000000 91 59 20 76 54
0000001 46 39 8 5 21
0000003 30 50 71 36 30
0000004 58 71 20 10 6
0000005 82 79 16 21 80
0000007 50 2 33 15 62
$

$ ycat file1 file2
0000000 浜地_____ 50 F 0000000 91 59 20 76 54
0000001 鈴田_____ 50 F 0000001 46 39 8 5 21
0000003 杉山_____ 26 F 0000003 30 50 71 36 30
0000004 白土_____ 40 M 0000004 58 71 20 10 6
0000005 崎村_____ 50 F 0000005 82 79 16 21 80
0000007 梶川_____ 42 F 0000007 50 2 33 15 62
$
```

### 【例2】

-<n>オプションを使用すると、ファイルを<n>で指定した空白の分だけ間を空けることができる。

```
$ ycat -3 file1 file2
0000000 浜地_____ 50 F   0000000 91 59 20 76 54
0000001 鈴田_____ 50 F   0000001 46 39 8 5 21
0000003 杉山_____ 26 F   0000003 30 50 71 36 30
0000004 白土_____ 40 M   0000004 58 71 20 10 6
0000005 崎村_____ 50 F   0000005 82 79 16 21 80
0000007 梶川_____ 42 F   0000007 50 2 33 15 62
$
```

**【例3】**

-0を指定した場合にはファイルの隙間は追加されない。

```
ycat -0 file1 file2
0000000 浜地_____ 50 F0000000 91 59 20 76 54
0000001 鈴田_____ 50 F0000001 46 39 8 5 21
0000003 杉山_____ 26 F0000003 30 50 71 36 30
0000004 白土_____ 40 M0000004 58 71 20 10 6
0000005 崎村_____ 50 F0000005 82 79 16 21 80
0000007 梶川_____ 42 F0000007 50 2 33 15 62
$
```

**【例4】**

テキストを標準入力から読み込む場合には-を記述する。

```
$ cat file1 | ycat - file2
```

```
$ cat file2 | ycat file1 -
```

**【備考】**

ycat(1)は連結後の各ファイルの形が崩れないように、はじめに各ファイルの最大幅を測っている。このため、単純に空白横連結しただけなら次のようにpaste(1)コマンドを使う方が高速となる。

```
$ paste -d" " file1 file2 file3
```

**【関連項目】**

ctail(1)、getfirst(1)、getlast(1)、tcat(1)

## yobi(1)

### 【名前】

yobi : 曜日算出

### 【書式】

```
Usage : yobi [-ej] <field> <filename>
       : yobi -d[ej] $string
Version : Thu Dec 15 13:46:38 JST 2011
```

### 【説明】

引数や標準入力から読み込んだファイルの指定したフィールドの年月日の曜日のコードを指定したフィールドの次のフィールドに挿入して出力する。元ファイルの日付のフォーマットはYYYYMMDDの8桁である必要があり、曜日のコードは(日=0、月=1、火=2、水=3、木=4、金=5、土=6)で出力する。

### 【例1】

```
$ cat data
0001 0000007 20060201 117 8335 -145
0001 0000007 20060203 221 15470 0
0001 0000007 20060205 85 5950 0
0001 0000007 20060206 293 20527 -17
0001 0000007 20060207 445 31150 0
0002 0000007 20060208 150 11768 -1268
0002 0000007 20060209 588 41160 0
0002 0000007 20060210 444 31080 0
$

$ yobi 3 data
0001 0000007 20060201 3 117 8335 -145
0001 0000007 20060203 5 221 15470 0
0001 0000007 20060205 0 85 5950 0
0001 0000007 20060206 1 293 20527 -17
0001 0000007 20060207 2 445 31150 0
0002 0000007 20060208 3 150 11768 -1268
0002 0000007 20060209 4 588 41160 0
0002 0000007 20060210 5 444 31080 0
$
```

### 【例2】

-eオプションを指定すると英語表記の曜日を挿入して出力する。

```
$ yobi -e 3 data
0001 0000007 20060201 Wed 117 8335 -145
0001 0000007 20060203 Fri 221 15470 0
0001 0000007 20060205 Sun 85 5950 0
0001 0000007 20060206 Mon 293 20527 -17
0001 0000007 20060207 Tue 445 31150 0
0002 0000007 20060208 Wed 150 11768 -1268
0002 0000007 20060209 Thu 588 41160 0
0002 0000007 20060210 Fri 444 31080 0
$
```

【例3】

-jオプションで日本語表記の曜日を挿入して出力する。

```
$ yobi -j 3 data
0001 0000007 20060201 水 117 8335 -145
0001 0000007 20060203 金 221 15470 0
0001 0000007 20060205 日 85 5950 0
0001 0000007 20060206 月 293 20527 -17
0001 0000007 20060207 火 445 31150 0
0002 0000007 20060208 水 150 11768 -1268
0002 0000007 20060209 木 588 41160 0
0002 0000007 20060210 金 444 31080 0
$
```

【例4】

-dオプションで引数に指定した日付に対して曜日を算出する。

```
$ yobi -d 20080112
6
$
```

```
$ yobi -de 20080112
Sat
$
```

```
$ yobi -dj 20080112
土
$
```

【関連項目】

calclock(1)、dayslash(1)、mdate(1)



ysum(1)

【名前】

ysum : 横集計

【書式】

Usage : ysum [+h] num=<n> <file>  
: ysum [+h] <file>  
Version : Thu Dec 15 13:46:38 JST 2011

【説明】

<file>ファイルの、同一行内の各フィールドの合計を行末に追加する。num=<n>で指定したフィールドの次以降の全フィールドを集計して最終フィールドの後に追加する。num=<n>を省略した場合、num=0と同じ動作をする。

【例1】

```
$ cat data
0000000 浜地_____ 50 F 91 59 20 76 54 ←番号 氏名 年齢 性別 教科別点数A B C D E
0000001 鈴田_____ 50 F 46 39 8 5 21
0000003 杉山_____ 26 F 30 50 71 36 30
0000004 白土_____ 40 M 58 71 20 10 6
0000005 崎村_____ 50 F 82 79 16 21 80
0000007 梶川_____ 42 F 50 2 33 15 62
$
```

教科別の点数(5フィールド目以降)を集計して最終フィールドの次に追加する。

```
$ ysum num=4 data
0000000 浜地_____ 50 F 91 59 20 76 54 300
0000001 鈴田_____ 50 F 46 39 8 5 21 119
0000003 杉山_____ 26 F 30 50 71 36 30 217
0000004 白土_____ 40 M 58 71 20 10 6 165
0000005 崎村_____ 50 F 82 79 16 21 80 278
0000007 梶川_____ 42 F 50 2 33 15 62 162
$
```

【例2】

+hオプションを指定すると先頭行以外の行を集計する。先頭行が項目名などのヘッダーのデータなどに使用する。このとき先頭行の集計値フィールドには@が追加される。

先頭行を飛ばして教科別の点数(5フィールド目以降)を集計して最終フィールドの次に追加するには次のようにコマンドを実行する。

```
$ cat data
0000000 浜地_____ 50 F 91 59 20 76 54 ←番号 氏名 年齢 性別 教科別点数A B C D
0000001 鈴田_____ 50 F 46 39 8 5 21
0000003 杉山_____ 26 F 30 50 71 36 30
0000004 白土_____ 40 M 58 71 20 10 6
0000005 崎村_____ 50 F 82 79 16 21 80
0000007 梶川_____ 42 F 50 2 33 15 62
$
```

```
$ ysum +h num=4 data | keta
No      氏名 年齢 性別  A  B  C  D  E  @      ←先頭行には@がつく
0000000 浜地_____ 50      F 91 59 20 76 54 300
0000001 鈴田_____ 50      F 46 39 8 5 21 119
0000003 杉山_____ 26      F 30 50 71 36 30 217
0000004 白土_____ 40      M 58 71 20 10 6 165
0000005 崎村_____ 50      F 82 79 16 21 80 278
0000007 梶川_____ 42      F 50 2 33 15 62 162
$
```

【関連項目】

kasan(1)、plus(1)、ratio(1)、sm2(1)、sm4(1)、sm5(1)

### zen(1)

#### 【名前】

zen : 全角へ変換

#### 【書式】

```
Usage   : zen <f1> <f2> .. <file>
: zen [-k] <file>
: zen -d <string>
Version : Fri Oct 21 11:26:06 JST 2011
OpenUSP (LINUX+FREEBSD/PYTHON2.4/UTF-8)
```

#### 【説明】

引数のファイルまたは標準入力におけるテキストデータの半角英数およびカタカナをすべて全角に変換して出力する。

#### 【例1】

引数のファイルの指定したフィールドの中身を全角に変換する。

```
$ cat data
これはデータです。
This is data
123 456 7890
$
```

```
$ zen 1 2 3 data
これはデータです。
This is data
1 2 3 4 5 6 7 8 9 0
$
```

#### 【例2】

フィールドを指定しない場合は行全体を全角にする。半角空白も全角空白に変換する。

```
$ cat data
1 2 3
$ cat data | zen
1 2 3
$
```

#### 【例3】

zen -k <file>の場合、<file>の半角カタカナだけ全角にする。メールの表題や本文には半角カナは使えないため、このフィルタを使用して前処理を実施する。

```
$ cat data2
1237ｲｲｴ
$ cat data2 | zen -k
123アイウエオ
$
```

【例4】

-dオプションは引数で指定された文字列の半角の部分を変えて出力する。

```
$ zen -d カカABC123      ←半角文字を引数で渡す。  
カタカナ A B C 1 2 3    ←すべて全角にして出力  
$
```

【関連項目】

han(1)

## タグ形式(5)

### 【名前】

タグ形式：1行目にヘッダがついたフィールド形式(5)

### 【書式】

```
ヘッダ1 ヘッダ2 ヘッダ3...  
フィールド1 フィールド2 フィールド3...  
...
```

### 【説明】

タグ形式(5)はフィールド形式(5)のデータの先頭行にヘッダが配置されたもの。主に次の用途で使用される。

複数のネームファイル(5)を単一のファイルにまとめる  
フィールド数がきわめて多い場合

ネームファイル(5)を使用することでフィールドに名前を与えることはできるが、1つの鍵にいくつもの値が存在するケースではファイル数が増えることになり扱いにくい面がある。タグ形式(5)はこうした場合に単一ファイル化する方法として利用できる。

### 【例1】

```
ID DATE AGE GENDER VALUE  
0001 20111008 23 F 0.845814  
0002 20111009 15 M 0.193475  
0003 20111010 81 F 0.484983  
0004 20111011 15 F 0.491736  
0005 20111012 91 F 0.893188  
0006 20111013 22 M 0.571077  
0007 20111014 33 F 0.502931  
0008 20111015 56 M 0.0401409  
0009 20111016 10 M 0.013111  
0010 20111017 72 F 0.482945
```

### 【関連項目】

フィールド形式(5)、ネーム形式(5)、ネームファイル(5)

### ネーム形式(5)

#### 【名前】

ネーム形式：空白区切りの2フィールドテキストデータ形式

#### 【書式】

第1フィールド 第2フィールド  
...

#### 【説明】

空白区切りの2フィールドテキストデータ形式。フィールド形式(5)の特殊な形式の1つ。第2フィールドに含まれる空白は区切り文字としては解釈されない。ネーム形式のファイルはネームファイル(5)と呼ばれる。

ネームファイル(5)は主にWebフォームの入力内容を保存するために使われる。POSTデータは鍵と値が対になったデータになっており、cgi-name(1)を使用することでネーム形式(5)へ変換することができる。ネーム形式を使用するコマンドとしてはほかにcheck\_attr\_name(1)やcheck\_need\_name(1)などがある。

#### 【例1】

```
ID 0001
DATE 20111008
AGE 23
GENDER F
VALUE 0.845814
```

#### 【例2】

ネーム形式(5)では1つめの空白のみがフィールド区切り文字として認識され、2つめ以降の空白は通常の文字として認識される。

```
ID 0000 001
DATE 2011 10/08
AGE 23
GENDER F
VALUE 1 - 0.845814
```

上記例であれば、1行目の第1フィールドはIDであり、第2フィールドは0000 001となる。

#### 【関連項目】

cgi-name(1)、check\_attr\_name(1)、check\_need\_name(1)、タグ形式(5)、フィールド形式(5)、ネームファイル(5)

### フィールド形式(5)

#### 【名前】

フィールド形式：空白区切りのテキストデータ形式

#### 【書式】

第1フィールド 第2フィールド 第3フィールド ...  
...

#### 【説明】

1つの空白文字を区切り文字としたテキストデータ形式。フィールド形式では行をレコード、1行あたりのそれぞれ空白で区切られたものをフィールドと呼んでいる。行頭から順に第1フィールド、第2フィールド、第3フィールド...と呼ばれる。

フィールド形式ではすべての行のフィールド数は同一である必要がある。フィールド形式はOpen usp Tukubaiやusp Tukubaiにおける基本データ形式であり、タグ形式(5)やネーム形式(5)はフィールド形式の一種となる。

マスタの特性を満たしたマスタファイルやトランザクションの特性を満たしたトランザクションファイルもフィールド形式のファイルとなる。

#### 【例1】

フィールド形式のファイル例は次のとおり。

```
0001 20111008 23 F 0.845814
0002 20111009 15 M 0.193475
0003 20111010 81 F 0.484983
0004 20111011 15 F 0.491736
0005 20111012 91 F 0.893188
0006 20111013 22 M 0.571077
0007 20111014 33 F 0.502931
0008 20111015 56 M 0.0401409
0009 20111016 10 M 0.013111
0010 20111017 72 F 0.482945
```

フィールド形式ではすべての行のフィールド数が同一である必要がある点に注意が必要。ユニケージ開発手法のコマンド実装系であるOpen usp Tukubaiやusp Tukubaiは行ごとのフィールド数が異なるフィールド形式のファイルは動作を保証しない。

#### 【関連項目】

タグ形式(5)、ネーム形式(5)、マスタファイル(5)、ネームファイル(5)、トランザクションファイル(5)

※ ユニケージはユニバーサル・シェル・プログラミング研究所 (<http://www.usp-lab.com/>) の登録商標。  
※ usp Tukubaiはユニバーサル・シェル・プログラミング研究所 (<http://www.usp-lab.com/>) の登録商標。  
※ 2012年7月から始まる教育講座に合わせて、従来の「ユニケージ」または「ユニケージ方法論」は「ユニケージ開発手法」へ名称を統一。

### マスタファイル(5)

#### 【名前】

マスタファイル：マスタの性質を満たすフィールド形式(5)のファイル

#### 【書式】

第1フィールド 第2フィールド 第3フィールド ...  
...

#### 【説明】

それぞれの行がIDなどで管理され重複する行がなく、データの更新もあらかじめ定められたタイミングでのみ実施され、随時データの変更が入ったり、逐次データが追加されるといったことがなく、商品や店、都道府県、市区町村などデータ処理中に主要なデータとして出現するようなデータを一般的にマスタと呼んでいる。

ユニケージ開発手法では、マスタに該当するフィールド形式(5)のファイルをマスタファイル(5)と呼んでいる。マスタの特性以外に、ユニケージ開発手法におけるマスタファイル(5)は次の特性を満たしている必要がある。

第1フィールドまたは第1フィールドから連続するいくつかのフィールドがキーフィールドになっている。  
それ以外のフィールドはキーフィールドに対応する値になっている。  
キーフィールド同士は重複しない。  
キーフィールドがLANG=C sortで整列されている。

たとえば第1フィールドが店番号、第2フィールドが店名で構成される2フィールドのフィールド形式(5)のファイルで、第1フィールドの値がLANG=C sortで整列されたファイルはマスタファイル(5)と呼ばれる。

マスタファイル(5)は短縮してマスタと呼ばれることがあり、また、Tukubaiコマンドマニュアルではmasterと表記される。マスタファイル(5)とは対照的に随時データが追加されるようなフィールド形式(5)のファイルはトランザクションファイル(5)と呼ばれている。

#### 【例1】

マスタファイルの例は次のとおり。

```
001 新橋店
002 目黒店
003 五反田店
004 新宿店
005 池袋店
006 原宿店
007 渋谷店
008 吉祥寺店
009 下北沢店
010 品川店
```

#### 【関連項目】

ネームファイル(5)、トランザクションファイル(5)、フィールド形式(5)、タグ形式(5)、ネーム形式(5)

※ ユニケージはユニバーサル・シェル・プログラミング研究所 (<http://www.usp-lab.com/>) の登録商標。



### ネームファイル(5)

#### 【名前】

ネームファイル：ネーム形式のファイル

#### 【書式】

第1フィールド 第2フィールド

...

#### 【説明】

ネーム形式(5)のファイル。第2フィールドに含まれる空白は区切り文字としては解釈されない。

ネームファイルは主にWebフォームの入力内容を保存するために使われる。POSTデータは鍵と値が対になったデータになっており、cgi-name(1)コマンドを使用することでネーム形式へ変換することができる。ネーム形式を使用するコマンドとしてはほかにcheck\_attr\_name(1)やcheck\_need\_name(1)などがある。

#### 【例1】

```
ID 0001
DATE 20111008
AGE 23
GENDER F
VALUE 0.845814
```

#### 【例2】

ネームファイルでは1つめの空白のみがフィールド区切り文字として認識され、2つめ以降の空白は通常の文字として認識される。

```
ID 0000 001
DATE 2011 10/08
AGE 23
GENDER F
VALUE 1 - 0.845814
```

上記例であれば、1行目の第1フィールドはIDであり、第2フィールドは0000 001となる。

#### 【関連項目】

cgi-name(1)、check\_attr\_name(1)、check\_need\_name(1)、タグ形式(5)、フィールド形式(5)、ネーム形式(5)

### トランザクションファイル(5)

#### 【名前】

トランザクションファイル：トランザクションの性質を満たすフィールド形式(5)のファイル

#### 【書式】

第1フィールド 第2フィールド 第3フィールド ...  
...

#### 【説明】

売上データなど随時データの追加などが実施されるタイプのデータはトランザクションと呼ばれている。トランザクションは短縮してトランと呼ばれることも多い。

ユニケーj開発手法では、トランザクションに該当するフィールド形式(5)のファイルをトランザクションファイル(5)と呼んでいる。

トランザクションファイル(5)は短縮してトランと呼ばれることがあり、また、Tukubaiコマンドマニュアルではtranと表記される。トランザクションファイル(5)とは対照的にデータ変更のタイミングがあらかじめ定められており、キーフィールドが重複せずかつ整列されているようなフィールド形式(5)のファイルはマスタファイル(5)と呼ばれている。

#### 【例1】

トランザクションファイル(5)の例は次のとおり。

```
20110405 001 13513543
20110405 002 3403346
20110406 001 3543024
20110406 002 12412311
20110406 003 88797982
20110407 001 1354131
20110407 002 12343423
20110407 003 7657765
```

#### 【関連項目】

マスタファイル(5)、ネームファイル(5)、フィールド形式(5)、タグ形式(5)、ネーム形式(5)

※ ユニケーjはユニバーサル・シェル・プログラミング研究所 (<http://www.usp-lab.com/>) の登録商標。

Contact us: [uecinfo@usp-lab.com](mailto:uecinfo@usp-lab.com)  
Copyright (c) 2012, 2013 Universal Shell Programming Laboratory All Rights Reserved.