Linux ファイル編集コマンド in Windows（grep,join,awk,sed 他）

目次

Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.

Linux ファイル編集コマンド in Windows（grep,join,awk,sed 他）

はじめに

Linux には Unix のデータ加工用（にも使える）ソフトウェアが引き継がれてバージョンアップが続いています。ソースコードが公開され、多くのプログラマーが参加しているので機能、性能、信頼性は申し分ありません。Windows へは機能を模倣したアプリの移植が細々行われてたきた程度でしたが、Windows 10 からは Linux のソフトウェアをそのまま使える環境が増えてきました。マイクロソフト社が Windows に導入したオープンソース（OpenSSH 他）、WSL や Docker のコンテナで Linux を動作させる方法、Linux のライブラリやツールを Windows 用に構築した Mingw/MSYS 等があります。

注意点としては、環境を跨いで渡されたデータに日本語が含まれていると文字コードの変換が必要になる場合があります（Linux 側のデフォルトは UTF-8、Windows は UTF-16 と Shift-JIS:CP932）。

1. Mingw と MSYS2

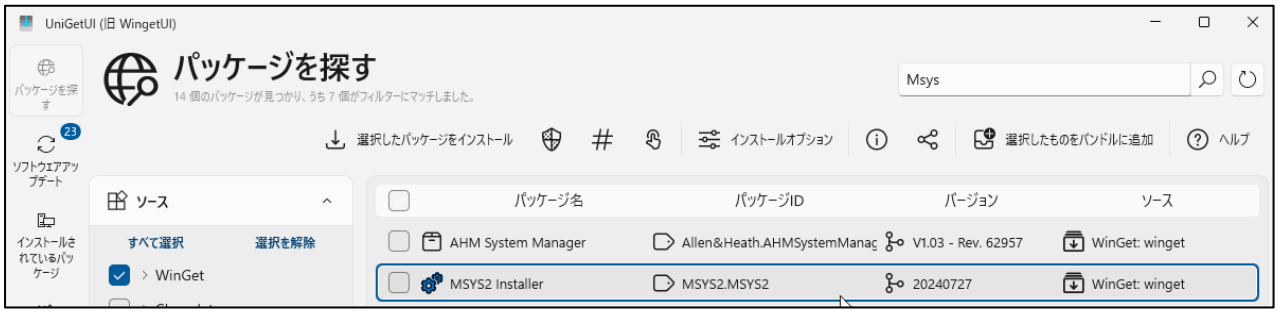
Linux（Unix）のシェルやツールを Windows に移植するプロジェクトやその派生はいくつもあり、プロジェクト毎の移植範囲の違いや組合せもいろいろあるのですが、Mingw-w64 と MSYS2 の組合せがマクロソフト社のサイトでも紹介されていて[1]簡単に導入できます。

コマンドやオプションはバージョン等による違いを除き、Linux でも殆どそのまま使えるはずです。

＜インストール方法＞

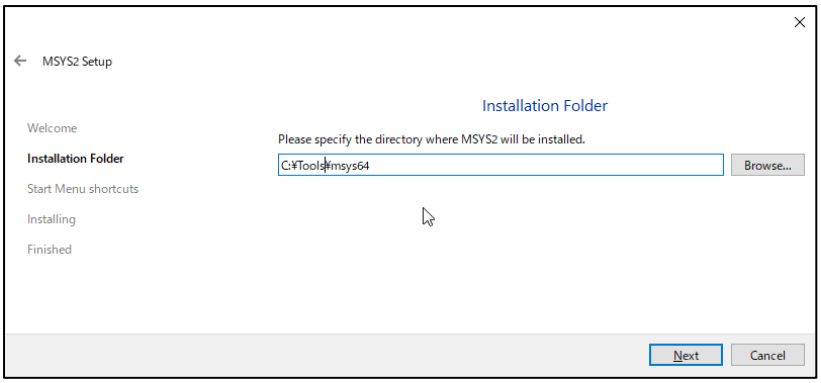
Windows Package Manager（Winget[2]）を使い MSYS2 をインストールします。

（下図は UnigetUI を使っていますが、＞ winget install MSYS2.MSYS2 の実行と同じです）



以降の説明は C:¥Tools¥msys64 というディレクトリにインストールしたものとして進めます。

（初期値で表示される C:¥msys64 のままでも問題はなく、パスに空白や日本語を含まなければよい）



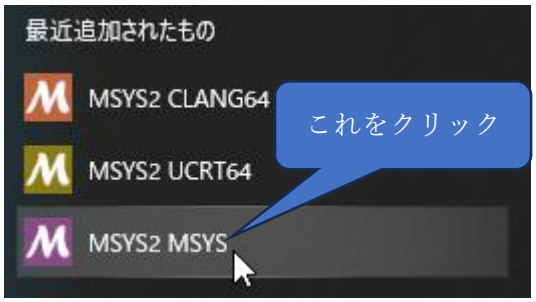
[1] Mingw-w64 https://learn.microsoft.com/ja-jp/vcpkg/users/platforms/mingw

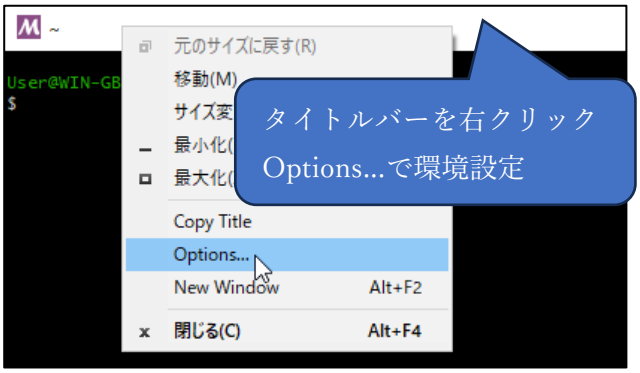
[2] winget ツールを使用したアプリケーションのインストールと管理
https://learn.microsoft.com/ja-jp/windows/package-manager/winget/

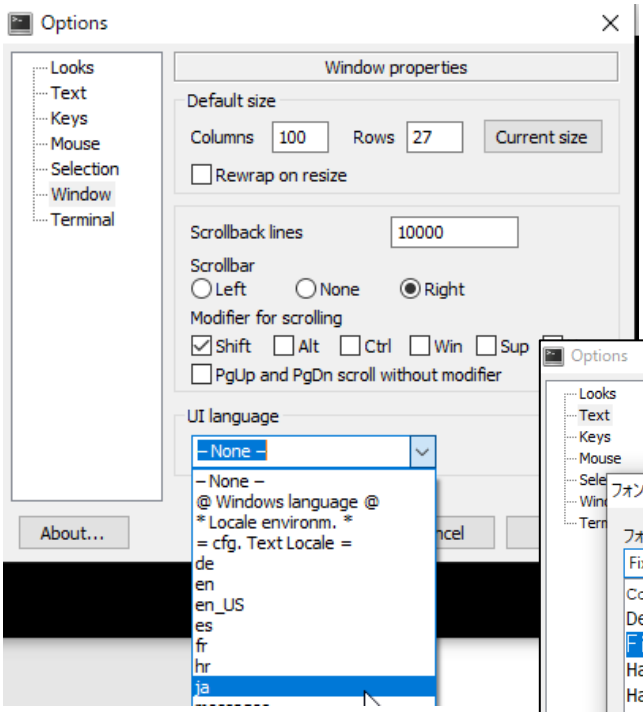
Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.

Linux ファイル編集コマンド in Windows（grep,join,awk,sed 他）

＜インストール後のターミナルの起動・設定＞

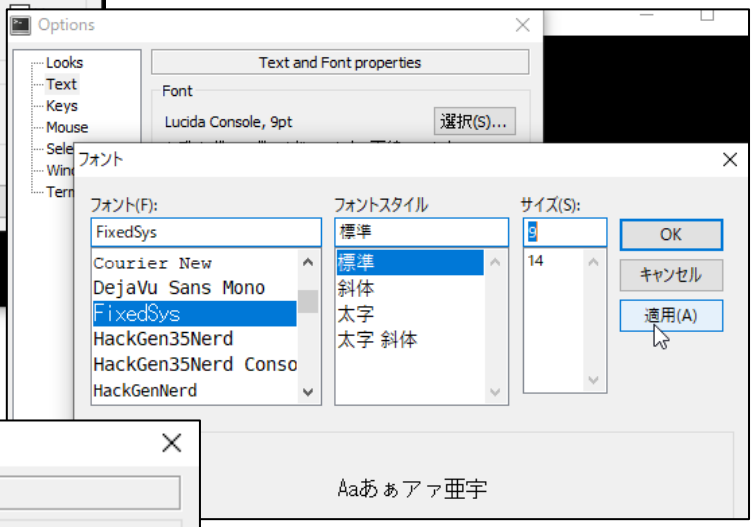


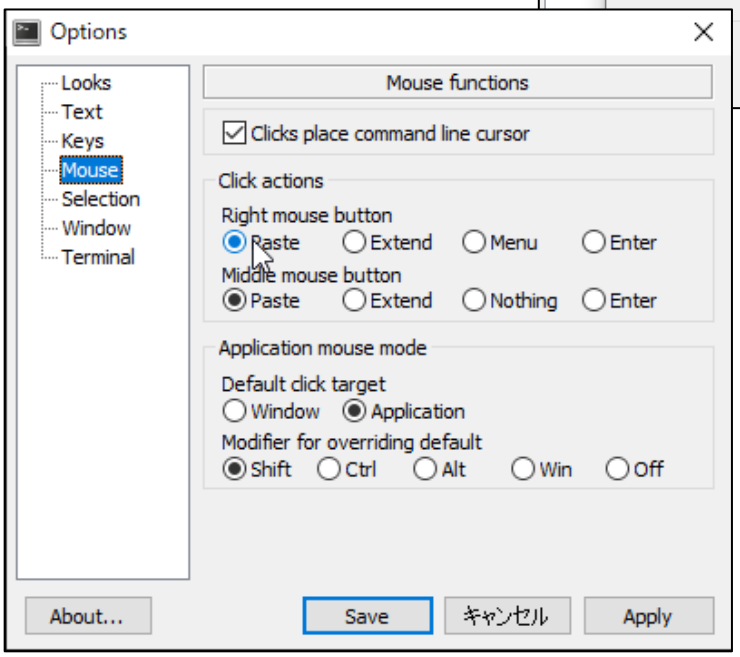




Window – UI language で ja を選択し[Apply]押下

Text Font [選択]を押下





Mouse でマウスの動作を設定
（右クリックをデフォルトの Menu から Paste に設定すると、よくある Linux のターミナル操作に近づきます。
…範囲選択だけで Copy が行われるので、選択⇒右クリックが Copy & Paste になる）

Save して新しい画面を開く（ALT＋F2）と新設定のターミナルが開きます。

Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.

Linux ファイル編集コマンド in Windows（grep,join,awk,sed 他）

2. 前提事項

Linux のシェル（MSYS2 は bash が標準）は制御構文や関数、変数を使い複雑なプログラミングができますが、ここではシェルの制御文は極力使わないでコマンドの機能を中心に紹介します。

とはいえ Linux のコマンドはシェルを通してパラメータや入出力を受け渡すので、少しだけ知っておくべき事柄があります。

（1）入力区切り文字（IFS）

フィールドの区切りとして使う記号・文字で、変数 IFS に設定します。コマンドラインで入力されたパラメータを分割したり、レコードをフィールドに分割する際に参照されます。

IFS のデフォルトは空白' '、タブ'¥t'、改行'¥n' の 3 つが設定されています（表示>printf %q "$IFS"）。

（2）ファイル出力

多くのコマンドは出力ファイルを指定するオプションを持っておらず、コマンドの処理結果は別のコマンドにパイプ「|」で渡すかリダイレクト「>または>>」でファイルに出力します。

（3）日本語

ファイルに含まれる文字コードは、自動認識できるコマンドもありますが基本は環境変数 locale に設定されている文字コードとして扱います。

```
$ locale
LANG=
LC_CTYPE="ja_JP.UTF-8"
LC_NUMERIC="C.UTF-8"
LC_TIME="C.UTF-8"
LC_COLLATE="C.UTF-8"
LC_MONETARY="C.UTF-8"
LC_MESSAGES="C.UTF-8"
LC_ALL=
```

MSYS 2 のバージョン 20240727 では Linux で一般的な UTF-8 が locale の初期設定になっています。

異なる文字コードを使っている場合は、コマンドの実行前に変換する必要があります。具体的には、以下のように入力ファイル名を記述する部分を iconv で変換した結果で置換します。

```
$ cat temp.txt
??????????12345abcde
??????????12345abcde
??????????12345abcde
```

```
$ cat <(iconv -f cp932 temp.txt)
あいうえお12345abcde
あいうえお12345abcde
あいうえお12345abcde
```

iconv -f cp932 temp.txt は temp.txt ファイルの文字コード cp932(=拡張 Shift_jis= Windows-31J)[3]をデフォルト（locale）の文字コードに変換する処理で、<()の中に書いて temp.txt を置き換えます。

iconv のオプションで書ける文字コードは iconv -l で確認してください。

※<(iconv -f cp932 temp.txt)〔<と(の間に空白を開けない〕は「プロセス置換」と呼び、
cat <(iconv -f cp932 temp.txt) と iconv -f cp932 temp.txt | cat は同じ結果が得られます

[3] コード ページ識別子 https://learn.microsoft.com/ja-jp/windows/win32/intl/code-page-identifiers

Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.

Linux ファイル編集コマンド in Windows（grep,join,awk,sed 他）

3. コマンド用例

多くの場合 同一の結果が得られるコマンドの組合せが複数あり、更に、コマンドによっては制御構文を使って複雑なプログラミングもできますが、ここでは簡単なパラメータ変更だけで仕事に使えそうな例を纏めます。

3.1. レコードの抽出（grep）

（１）固定長レコードの n 桁目の値による選択

正規表現で n 桁-1 の任意の文字の次に、選択条件の値を指定します。

例）11 文字目が a か A のレコードを抽出する場合

```
grep -i -E -h  '^.{10}a'  入力ファイル名
```

正規表現：'^.{10}a'　…^:行頭、.:任意の文字、{n}:直前の文字を n 回繰り返し

オプション

-i :大文字・小文字を同一視

-E :拡張正規表現〔標準で正規表現は使えますが、{}を使うには拡張正規表現が必要〕

-h : 各行にファイルパス／名を出力しない（複数ファイルを対象にした場合に有効）

（２）カンマ区切りのレコードから特定のフィールドの値で選択

正規表現で「任意の文字列,」をパターンとして、パターンの繰り返し数を指定します。

例）２つ目のフィールドの先頭が a か A のレコードを抽出する場合

```
grep -i -E -h  '^([^,]*,){1}abcde'  入力ファイル名
```

正規表現：'^([^,]*,){1}a'　…^:行頭、[^,]*:カンマではない文字の繰り返し

():「カンマ以外の任意の文字列＋カンマ」でグループを作成

{}:直前のグループの繰り返し数

＜実行例＞

```
$ cat temp.csv
あいうえお,abcde,"123,456",かきくけこ
かきくけこ,ABCDE,"100,000",さしすせそ
さしすせそ,ghijk,"110,000",たちつてと
たちつてと,jklmn,"111,000",なにぬねの
なにぬねの,mnopq,"111,100",はひふへほ
はひふへほ,pqrst,"111,110",まみむめも
まみむめも,stuvw,"111,111",やゆよ
```

```
$ grep -i -E -h '^([^,]*,){1}abcde' temp.csv
あいうえお,abcde,"123,456",かきくけこ
かきくけこ,ABCDE,"100,000",さしすせそ
```

引用符「”」で囲んだ**フィールドにカンマが含まれる可能性がある場合**は、検索文字列のグループの中に{”引用符以外の任意の文字列”,}を or「|」条件で加え'^("[^"]*",|[^,]*,){*n*}"*検索文字列*'とします。

```
$ grep -iEh '^("[^"]*",|[^,]*,){3}さ' temp.csv
かきくけこ,ABCDE,"100,000",さしすせそ
```

※ 例示した正規表現はここに示したデータバリエーションでのみ試験しています。完全でない可能性もありますので、利用者の要件を満たせているか否かは各自で確実に検証してください

Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.

Linux ファイル編集コマンド in Windows（grep,join,awk,sed 他）

3.2. ファイルのマッチング（join）

join を使うと、２つのファイルをキー項目でマッチングすることができます。

＜実行例＞

仕訳明細 CSV の科目コードで勘定科目マスタ CSV とマッチングさせ、科目名称を付加した出力を作ります。

（１）入力ファイル

PostgreSQL の以下のコマンドで作ったフォーマットです。

```
--仕訳明細レコード：科目コード順、Null は空文字、見出し付き
¥COPY (select * from t_siwake_meisai order by kamokucd)
    TO 'C:¥Users¥User¥Desktop¥meisai.csv' WITH CSV NULL AS '' HEADER;

--勘定科目マスタ：科目コード順、Null は空文字、見出し付き、文字コードを UTF-8 に変換
¥COPY (select * from m_kamoku order by kamokucd)
    TO 'C:¥Users¥User¥Desktop¥kamoku.csv' WITH CSV NULL AS '' HEADER ENCODING 'UTF-8';
```

① ファイル１（maisai.csv）イメージ

```
denpyono,taisyakukb,gyo,kamokucd,kingaku,aitekamokucd
00001,1,0,110101,1000000,000000
（以下略）
```

② ファイル２（kamoku.csv）イメージ

```
kamokucd,kamokunm,dennyukb,taisyakukb,chohyokb
000000,諸口,0,3,0
（以下略）
```

（２）コマンド

```
join -t, -1 4 -2 1 -a 1 --header  meisai.csv  kamoku.csv
```

オプション

-t : 続く文字でフィールドを分割する -t, でカンマ区切りを指示
-1 : １つめのファイルのマッチングに使用するフィールド番号
-2 : ２つめのファイルのマッチングに使用するフィールド番号
-a : 全てのレコードを出力する（マッチしなくても）ファイル番号
--header : 見出し行があるものとして処理を行う

＜結果＞

```
$ join -t, -1 4 -2 1 -a 1 --header ./join/meisai.csv ./join/kamoku.csv
kamokucd,denpyono,taisyakukb,gyo,kingaku,aitekamokucd,kamokunm,dennyukb,taisyakukb,chohyokb
110101,00002,1,0,2000000,000000,現金,1,1,1
110101,00001,1,0,1000000,000000,現金,1,1,1
110102,00002,1,1,20000000,000000,預金,1,1,1
110102,00001,1,1,10000000,000000,預金,1,1,1
120100,00001,1,2,500000000,000000,商品,1,1,1
```

以下略

Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.

Linux ファイル編集コマンド in Windows（grep,join,awk,sed 他）

3.3. テーブル形式に編集（column）

column を使うと、他のコマンドの処理結果等を簡易に見易く処理できます。

＜実行例＞

```
join -t, -1 4 -2 1 -a 1 --header ./join/meisai.csv ./join/kamoku.csv | column -t -s,
```

```
$ join -t, -1 4 -2 1 -a 1 --header ./join/meisai.csv ./join/kamoku.csv | column -t -s,
kamokucd  denpyono  taisyakukb  gyo  kingaku      aitekamokucd  kamokunm                          dennyukb  taisyakukb  chohyokb
110101    00002     1           0    2000000      000000        現金                              1         1           1
110101    00001     1           0    1000000      000000        現金                              1         1           1
110102    00002     1           1    20000000     000000        預金                              1         1           1
110102    00001     1           1    10000000     000000        預金                              1         1           1
120100    00001     1           2    500000000    000000        商品                              1         1           1
120100    00001     2           0    1000000      000000        商品                              1         1           1
120100    00002     2           0    2000000      000000        商品                              1         1           1
120200    00002     2           1    20000000     000000        製品                              0         1           1
120200    00001     2           1    10000000     000000        製品                              0         1           1
810101    00003     2           0    10000000000  000000        売上高（売上控除高を除く）        1         2           2
810109    00003     1           0    10000000     000000        売上控除高                        1         1           2
810200    00003     2           1    1000000000   000000        役務収益                          1         2           2
820100    00003     1           1    1000000000   000000        期首商品・製品たな卸高            1         1           2
820201    00003     1           2    10000000000  000000        当期商品仕入高（仕入控除高を除く）1         1           2
820209    00003     2           2    100000000    000000        仕入控除高                        1         2           2
820300    00003     1           3    100000000    000000        当期製品製造原価                  1         1           2
820400    00003     2           3    1000000000   000000        期末商品・製品たな卸高            1         2           2
```

column のオプション

-c : 幅 表示の全体の幅を指定する

-t :入力行のカラム数を判定して整形する

-s : t オプションを使う際に、入力行をカラムに分ける区切り文字を指定する

-x : 行を埋める前に列を埋める（整列の方向を縦方向優先に変える…省略時の動作）

＜column -x の例＞

```
$ seq 1 50 | column
1       6       11      16      21      26      31      36      41      46
2       7       12      17      22      27      32      37      42      47
3       8       13      18      23      28      33      38      43      48
4       9       14      19      24      29      34      39      44      49
5       10      15      20      25      30      35      40      45      50
```

3.4. グループ集計（awk）

Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.

Linux ファイル編集コマンド in Windows（grep,join,awk,sed 他）

3.4.1. awk について

awk は制御構文と変数を使うことができ、複雑な処理も記述できるプログラミング言語です。

一方で制御構造を ’条件{ 処理 }’のように簡明に表現することができるので、深みにはまらなければ流用可能な使い勝手の良いコードを書くことができます。

（1）コマンド書式

ここではコマンドオプション、フラグ、外部からの変数割り当て等の機能は使わずに、以下の単純な書式だけに限定します。

awk ’ *プログラム* ’ *入力ファイル*

（2）プログラム

プログラムは全体を単一引用符「’」で囲い、処理条件 { ステートメント } の形式で定義します。

① 処理条件 には以下を指定します。

BEGIN {}　　…プログラム開始時に 1 回実行されます

END {}　　…入力ファイルが全て処理された後に実行されます

レコード選択 {}　…レコード番号や/正規表現/で該当レコードを選択し処理対象にします

② 変数

特殊変数　　… awk の組み込み変数で、よく使われるものに以下があります

FS : 入力のフィールド区切り文字(デフォルトは$IFS…空白、タブ、改行)

OFS : 出力のフィールド区切り文字(デフォルトは空白)

NR :入力レコードの番号（NR 番目）

FIELDWIDTHS : 入力を固定幅で扱う場合の各入力フィールドの文字数

$0,$1~$nn : 1～nn はフィールドの通し番号 $0 はレコード全体を表す

利用者定義変数… プログラム内で最初に参照した時点で定義され、文字列、数値の他に連想配列（data["abc”]=123 のように文字列をインデックスに使える配列）が使えます

③ ステートメント

ステートメント（{}）がないと標準出力への書き出し(print)が行われ、grep のように動作します。

```
$ awk '/82/' ./join/meisai.csv
00003,1,1,820100,1000000000,000000
00003,1,2,820201,10000000000,000000
00003,2,2,820209,100000000,000000
00003,1,3,820300,100000000,000000
00003,2,3,820400,1000000000,000000
```

ステートメントは{}で囲み、ステートメントとステートメントの間をセミコロン;で区切ります。

よく使うステートメントには変数をそのまま出力する print と編集して出力する printf があります。

```
$ awk '/82/{ printf "%6s %11d", $4, $5; print " 円" }' FS="," ./join/meisai.csv
820100  1000000000 円
820201 10000000000 円
820209   100000000 円
820300   100000000 円
820400  1000000000 円
```

print は出力後改行します（改行コードが自動付加）
printf は改行が必要なら¥n を出力文字列の最後に付けます

Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.

Linux ファイル編集コマンド in Windows（grep,join,awk,sed 他）

3.4.2. 実行例

仕訳明細 CSV を、科目コードと貸借区分をキーにグルーピングして金額を集計します。

```
awk '
BEGIN{ FS=",";OFS=","; }
 NR==1{ print $4,$2,$5; next; }
 { key=($4 OFS $2); val[key]+=$5; }
 END{ for(key in val) print key, val[key] }
' ./join/meisai.csv
```

2 行目-プログラム開始（BEGIN）で、入出力のフィールド区切り文字をカンマ「,」にします。
3 行目-1 件目（NR==1）は見出し行なので 4 番目(“kamokucd”)、2 番目(“taisyakukb”)、5 番目(“kingaku”)を print して、次のレコードを読込みに進み（; next）ます
4 行目-＜条件指定なし＞2 件目以降 key という変数に 4 番目と 2 番目のフィールドを OFS「,」で連結した値を作り、val という連想配列のインデックスにして 5 番目のフィールドを加算します
5 行目-全レコード処理後（END)、val に格納されている全てのインデックスと値を print します

【入力ファイルの一部】

```
$ cat ./join/meisai.csv
denpyono,taisyakukb,gyo,kamokucd,kingaku,aitekamokucd
00002,1,0,110101,2000000,000000
00001,1,0,110101,1000000,000000
00002,1,1,110102,20000000,000000
00001,1,1,110102,10000000,000000
```

【処理結果】

```
$ awk '
BEGIN{ FS=",";OFS=","; }
 NR==1{ print $4,$2,$5; next; }
 { key=($4 OFS $2); val[key]+=$5; }
 END{ for(key in val) print key, val[key] }
' ./join/meisai.csv
kamokucd,taisyakukb,kingaku
820209,2,100000000
820100,1,1000000000
120100,2,3000000
110101,1,3000000
820300,1,100000000
810109,1,10000000
810101,2,10000000000
810200,2,1000000000
120200,2,30000000
110102,1,30000000
820400,2,1000000000
820201,1,10000000000
120100,1,500000000
```

集計結果を検証するため、
科目コード 110101,110102 のみ抽出

```
$ awk '
BEGIN{ FS=",";OFS=","; }
 NR==1{ print $4,$2,$5; next; }
 { key=($4 OFS $2); val[key]+=$5; }
 END{ for(key in val) print key, val[key] }
' ./join/meisai.csv | awk '/^11010./'
110101,1,3000000
110102,1,30000000
```

※ 連想配列はキー値の順で格納されない（ハッシュ化する）ため、出力順は不定です

Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.

Linux ファイル編集コマンド in Windows（grep,join,awk,sed 他）

3.5. 並べ替え（sort）

ファイルの内容やコマンドの実行結果を並べ替えるには sort コマンドを使います。

＜書式＞

sort -t *区切文字* -k *開始フィールド*,*終了フィールド* *数値属性[n]* *降順[r]* *入力ファイル*

オプション

-t :フィールド区切り文字 - 指定しない場合、空白文字でフィールドを分割する

-k : 並べ替えのキーを指定 - 終了フィールドのデフォルトはレコードの最終フィールド

例）csv の 5 列目を数値として逆順に並べ替える場合は、以下のようになります

```
sort -t, -k5,5nr
```

※ オプションが無視されたり、動作が想定と異なる場合は --debug をオプションに追加して実行することで sort の動作条件・状況を確認することができます

＜実行例 ①＞

csv ファイルを入力に、5 列目を数値として逆順 sort、同値行は 4 列名を文字の昇順で並べ替える

```
sort -t, -k5,5nr -k4,4  ./join/meisai.csv
```

```
$ sort -t, -k5,5nr -k4,4  ./join/meisai.csv
00003,2,0,810101,10000000000,000000
00003,1,2,820201,10000000000,000000
00003,2,1,810200,1000000000,000000
00003,1,1,820100,1000000000,000000
00003,2,3,820400,1000000000,000000
00001,1,2,120100,500000000,000000
00003,2,2,820209,100000000,000000
00003,1,3,820300,100000000,000000
00002,1,1,110102,20000000,000000
00002,2,1,120200,20000000,000000
00001,1,1,110102,10000000,000000
00001,2,1,120200,10000000,000000
00003,1,0,810109,10000000,000000
00002,1,0,110101,2000000,000000
00002,2,0,120100,2000000,000000
00001,1,0,110101,1000000,000000
00001,2,0,120100,1000000,000000
denpyono,taisyakukb,gyo,kamokucd,kingaku,aitekamokucd
```

Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.

Linux ファイル編集コマンド in Windows（grep,join,awk,sed 他）

＜実行例 ②＞

集計済の csv をパイプ（標準入力）で受け取り、フィールド 1 （kamokucd）とフィールド 2（taisyakukb）の昇順で並べ替えます。

```
awk '
 BEGIN{ FS=",";OFS=","; }
 NR==1{ print $4,$2,$5; next; }
 { key=($4 OFS $2); val[key]+=$5; }
 END{ for(key in val) print key, val[key] }
' ./join/meisai.csv ¥
 | sort -t, -k1,1 -k2,2
```

```
$ awk '
 BEGIN{ FS=",";OFS=","; }
 NR==1{ print $4,$2,$5; next; }
 { key=($4 OFS $2); val[key]+=$5; }
 END{ for(key in val) print key, val[key] }
' ./join/meisai.csv | sort -t, -k1,1 -k2,2
110101,1,3000000
110102,1,30000000
120100,1,500000000
120100,2,3000000
120200,2,30000000
810101,2,10000000000
810109,1,10000000
810200,2,1000000000
820100,1,1000000000
820201,1,10000000000
820209,2,100000000
820300,1,100000000
820400,2,1000000000
kamokucd,taisyakukb,kingaku
```

※-k1,1 と-k2,2 の 2 つを指定する代わりに、-k1,2 としても同じ意味になります

＜ヘッダーレコードがあるデータの sort＞

sort はヘッダーレコードを特別扱いしないので、1 レコード目（ヘッダー）と 2 レコード目以降の処理を分ける必要があります。これは Bash の複数コマンドを 1 つの実行単位にグループ化して標準入出力を使う機能を使い解決します。開発元推奨の方法[4]をは以下のようになります。

```
{ sed -u 1q; sort -t, -k1,2; }
```

…中括弧との空白とコマンドの後のセミコロンは必須

・最初の sed は入力データを編集して出力するコマンドです（詳細は sed の項参照）

・sed はヘッダーレコードの 1 件を処理（出力）して終了（q）⇒1q し、2 件目から sort が読む

・sed はバッファを使って一度に複数レコードを処理するので -u でバッファ使用を抑止します

[4] GNU Coreutils https://www.gnu.org/software/coreutils/manual/html_node/Header-lines.html

Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.

Linux ファイル編集コマンド in Windows（grep,join,awk,sed 他）

3.6. 標準入力とファイルのマッチング（join）

Linux のコマンドの多くは先行コマンドの出力をパイプ経由で受け取る（標準入力）ことができ、join コマンドのように複数の入力ファイルを使うコマンド（他に有名なのはファイル比較の diff）でも一方のファイルを標準入力にすることができます。

＜実行例＞

仕訳明細 CSV を科目コードと貸借区分で集計し、科目コードで並べ替えた結果をパイプで受け取り、勘定科目マスタ CSV とマッチングさせる場合は以下のようにします。

```
awk '
 BEGIN{ FS=",";OFS=","; }
 NR==1{ print $4,$2,$5; next; }
 { key=($4 OFS $2); val[key]+=$5; }
 END{ for(key in val) print key, val[key] }
' ./join/meisai.csv ¥
 | { sed -u 1q; sort -t, -k1,2; } ¥
 | join --header -a1 -t, -11 -21 - ./join/kamoku.csv
```

※ join のファイル１またはファイル２のいずれか、標準入力に置換したい方を “-“ に書き換えます

```
$ awk '
 BEGIN{ FS=",";OFS=","; }
 NR==1{ print $4,$2,$5; next; }
 { key=($4 OFS $2); val[key]+=$5; }
 END{ for(key in val) print key, val[key] }
' ./join/meisai.csv ¥
 | { sed -u 1q; sort -t, -k1,2; } ¥
 | join --header -a1 -t, -11 -21 - ./join/kamoku.csv
kamokucd,taisyakukb,kingaku,kamokunm,dennyukb,taisyakukb,chohyokb
110101,1,3000000,現金,1,1,1
110102,1,30000000,預金,1,1,1
120100,1,500000000,商品,1,1,1
120100,2,3000000,商品,1,1,1
120200,2,30000000,製品,0,1,1
810101,2,10000000000,売上高（売上控除高を除く）,1,2,2
810109,1,10000000,売上控除高,1,1,2
810200,2,1000000000,役務収益,1,2,2
820100,1,1000000000,期首商品・製品たな卸高,1,1,2
820201,1,10000000000,当期商品仕入高（仕入控除高を除く）,1,1,2
820209,2,100000000,仕入控除高,1,2,2
820300,1,100000000,当期製品製造原価,1,1,2
820400,2,1000000000,期末商品・製品たな卸高,1,2,2
```

Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.

Linux ファイル編集コマンド in Windows（grep,join,awk,sed 他）

3.7. 固定長レコードから CSV 等の編集出力（awk）

awk を使うとフィールドが区切られていない（分離文字を含んでいない）ファイルを文字数で分割してカンマやタブ区切りにすることができます。

＜実行例 ①＞

ファイルを読込み、連続している文字列を 5 文字で分離して間にカンマを挟みます。また、途中に固定で"****"のフィールドを追加します。

```
awk 'BEGIN{FIELDWIDTHS="5 5 5";OFS=",";}{print $1,$2,"****",$3}' temp-utf8.txt
```

FIELDWIDTHS : 入力ファイルの各フィールドの文字数
OFS : 出力側のフィールド区切り文字

【入力】

```
$ cat temp-utf8.txt
あいうえお12345abcde
あいうえお12345abcde
あいうえお12345abcde
```

【出力】

```
$ awk 'BEGIN{FIELDWIDTHS="5 5 5";OFS=",";}{print $1,$2,"****",$3}' temp-utf8.txt
あいうえお,12345,****,abcde
あいうえお,12345,****,abcde
あいうえお,12345,****,abcde
```

＜実行例 ②＞

フィールドの文字数を調整したい場合は printf を使います。printf は出力区切文字（OFS）を無視するので、必要であれば出力フォーマットに自前で設定します。

```
awk 'BEGIN{FIELDWIDTHS="5 5 5"}
     {printf "%-10.2s:%10.10d:%-5s¥n", $1,$2,$3}' temp-utf8.txt
```

```
$ awk 'BEGIN{FIELDWIDTHS="5 5 5"}
{printf "%-10.2s:%10.10d:%-5s¥n", $1,$2,$3}' temp-utf8.txt
あい        :0000012345:abcde
あい        :0000012345:abcde
あい        :0000012345:abcde
```

printf のフォーマット指定の例

%-10.2s	…左詰(-)で 10 桁のスペースに(.)2 文字(s)出力
%10.10d	…右詰で 10 桁のスペースに 10 桁の数値(d〔先行 0 詰め〕)を出力
%-5s	…左詰で 5 桁のスペースに 5 文字を出力
¥n	…改行コードを出力

Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.

Linux ファイル編集コマンド in Windows（grep,join,awk,sed 他）

3.8. ファイルの形式を確認する（file）

Windows と他の OS（Linux 等）では改行コードや標準で使われる文字コードが異なります。識別して正常に取り扱ってくれるコマンドもありますが、表示が崩れたりエラーが発生するプログラムも残っています。file の形式が想定した形式になっているか file コマンドで確認できます。

＜書式＞

`file` *ファイルパス*

＜実行例＞ Window で作られたファイルの改行コードは CRLF になっています

```
$ file temp-utf8.txt
temp-utf8.txt: Unicode text, UTF-8 text, with CRLF line terminators
```

3.9. １文字（制御コード含む）単位の置換や削除（tr）

tr コマンドを使うと、連続する文字の圧縮や特定の文字の置き換えが簡単にできます。よく使われる場面としては Windows で作成したテキスト（html やソースコード）を Linux サーバに送る際の改行コードの変換です。

（１）特定の文字の削除

Windows の改行コードはキャリッジリターン（CR）＋ラインフィード（LF)、Linux は LF だけなので、ファイル中の全ての CR を消すことで Linux に合わせることができます。

例）全ての CR（メタ文字 ¥r で表記）を削除

```
tr -d '¥r'
```

オプション

-d : 引数で指定された文字を削除して標準出力に出力

＜実行例＞

Windows 形式のファイル temp-UTF8.txt から CR を消して temp-UTF8-delCR.txt に保存します。

```
$ cat temp-utf8.txt | tr -d '¥r' > temp-utf8-delCR.txt
```

```
$ file  temp-utf8.txt temp-utf8-delCR.txt
temp-utf8.txt:       Unicode text, UTF-8 text, with CRLF line terminators
temp-utf8-delCR.txt: Unicode text, UTF-8 text
```

（２）大文字小文字の置換

文字種を指定することにより変換を行うこともできます。

例）アルファベットの小文字を大文字に変換する。

```
tr [:lower:] [:upper:]
```

Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.

＜実行例＞

```
$ cat temp-utf8.txt
あいうえお12345abcde
あいうえお12345abcde
あいうえお12345abcde
```

```
$ cat temp-utf8.txt | tr [:lower:] [:upper:]
あいうえお12345ABCDE
あいうえお12345ABCDE
あいうえお12345ABCDE
```

（３）文字セットによる置換

tr は１文字単位に変換を行いますが、パラメータの置換前の文字列（文字セット 1）と置換後の文字列（文字セット 2）は複数文字で指定することができます。そして、最初は大概 勘違いをして誤った使い方をしてしまいます。

例）複数文字の置換

tr ’文字セット１’ ’文字セット２’

＜実行例 ①＞

文字セット 1 と文字セット 2 が同じ文字数の場合　　1 文字ずつ、以下の変換を行います

```
$ echo 12345111112222233333444 | tr '123' 'ABC'
ABC45AAAAABBBBBCCCCC4444
```

セット１　セット２
1　⇒　A
2　⇒　B
3　⇒　C

＜実行例 ②＞

文字セット 1 の数が多い場合、デフォルトは文字セット 2 の最後の文字が続いたように動作します。

```
$ echo 12345111112222233333444 | tr '1234' 'ABC'
ABCC5AAAAABBBBBCCCCCCCCC
```

3.10. 文字列の挿入／置換（sed）

tr は文字単位ですが、sed（Stream Editor）を使えば文字列の置換や挿入ができます。できることは awk と重複しますが、awk がフィールド／レコードで操作するのに対して sed は行指定や正規表現で該当箇所を特定し、編集コマンドを使って編集します。また、やめておいた方が無難ですが sed は入力ファイルを直接 更新することができます（同時にバックアップファイルを作ることもできます）。

（１）文字（列）の挿入

例）Linux の改行コード（LF）の前に CR を追加して Windows の改行コード（CRLF）にします。

sed -e ’s/$/¥r/’ *入力ファイル*　# s:置換コマンド、/$/:行末に位置付、/¥r/:ＣＲに置換

オプション

-e : 後が実行するスクリプト（スクリプトを書いたファイルを実行する場合は -f を書きます）

-i : 入力ファイルを直接書き換え。-i がなければ標準出力に編集後の内容を出力

※ オプションに-e、-f を省略すると後続のパラメータをスクリプトとして実行します

Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.

Linux ファイル編集コマンド in Windows（grep,join,awk,sed 他）

＜実行例＞

改行コード LF の temp-UTF8-delCR.txt に CR を追加して、temp-utf8-addCR.txt に保存します。

```
sed 's/$/¥r/' temp-utf8-delCR.txt > temp-utf8-addCR.txt
```

＜出力したファイルの属性確認＞

```
$ file temp-utf8-delCR.txt temp-utf8-addCR.txt
temp-utf8-delCR.txt: Unicode text, UTF-8 text
temp-utf8-addCR.txt: Unicode text, UTF-8 text, with CRLF line terminators
```

※ この処理は CR(¥r)の制御コードを挿入しますが、使っているコマンドは置換（s）で、行末（$）を CR に変更（置換元の文字がないので追加される）し、sed が書き出し時に LF(¥n)を追加する（文字列書き出し時の通常動作）ので、出力されるレコードの末が CRLF になります

（２）行の挿入

i コマンドで指定行の前に行を追加。a コマンドで指定行の次に行を追加します。

例）ファイル temp-utf8.txt の先頭行に見出し行を追加します。

```
sed '1i 列1,列2,列3' temp-utf8.txt
```

スクリプトの内容：先頭の 1 が行番号、次の i が挿入コマンド、それ以降が挿入される文字列

＜実行前＞

```
$ cat temp-utf8.txt
1行目あいうえお,12345,abcdeabcde
2行目あいうえお,12345,abcdeabcde
```

＜実行後＞

```
$ sed '1i 列1,列2,列3' temp-utf8.txt
列1,列2,列3
1行目あいうえお,12345,abcdeabcde
2行目あいうえお,12345,abcdeabcde
```

（３）行の削除

d コマンドで指定行を削除します。

例）ファイル temp-utf8.txt から "1 行目"という文字列を含むレコードを削除します。

```
sed '/1行目/d' temp-utf8.txt  # 物理的な1レコード目を削除なら、sed '1d' temp-utf8.txt
```

＜実行前＞

```
$ cat temp-utf8.txt
1行目あいうえお,12345,abcdeabcde
2行目あいうえお,12345,abcdeabcde
```

＜実行後＞

```
$ sed '/1行目/d' temp-utf8.txt
2行目あいうえお,12345,abcdeabcde
```

Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.

Linux ファイル編集コマンド in Windows（grep,join,awk,sed 他）

（4）文字列の変更

s コマンドで文字列を検索し書き換えることができます。文字列の長さは変わっても構いません。

＜書式＞

sed ’s/検索文字列/置換文字列/[flags]’　…flags は置換の範囲や置換後の取り扱いを指定します

例 1）ファイル temp-utf8.txt の abcd を XYZ に書き換えます。

```
sed ’s/abcd/XYZ/’ temp-utf8.txt
```

＜実行前＞

```
$ cat temp-utf8.txt
1行目あいうえお,12345,abcdeabcde
2行目あいうえお,12345,abcdeabcde
```

＜実行後＞

```
$ sed ’s/abcd/XYZ/’ temp-utf8.txt
1行目あいうえお,12345,XYZeabcde
2行目あいうえお,12345,XYZeabcde
```

※ flags になにも指定しないとレコードの最初に見つけた 1 か所だけを処理対象にします

例 2）ファイル temp-utf8.txt の**全て**の abcd を XYZ に書き換えます。

```
sed ’s/abcd/XYZ/g’ temp-utf8.txt
```

```
$ sed ’s/abcd/XYZ/g’ temp-utf8.txt
1行目あいうえお,12345,XYZeXYZe
2行目あいうえお,12345,XYZeXYZe
```

※ flags に g を指定することで 1 か所見つけたあとも検索と置換を続けます

例 3）ファイル temp-utf8.txt の /([0-9]{5}):数値 5 桁の部分を /¥1⇒*([0-9]{5})*+“@”に /g 全て s 書き換えます。

```
sed -E ’s/([0-9]{5})/¥1@/g’ temp-utf8crlf.txt
```

＜実行前＞

```
$ cat temp-utf8crlf.txt
1行目,あいうえお,12345,abcde
2行目,やゆよ,987654,xyz
```

＜実行後＞

```
$ sed -E ’s/([0-9]{5})/¥1@/g’ temp-utf8crlf.txt
1行目,あいうえお,12345@,abcde
2行目,やゆよ,98765@4,xyz
```

オプションの -E は拡張正規表現を使うという意味です。元々 sed は正規表現が使えるのですが検索文字列に合致した部分〔”()”で囲むと保存される〕を後から利用する〔¥*n* で参照〕場合等は、拡張機能が必要になります。

Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.

Linux ファイル編集コマンド in Windows（grep,join,awk,sed 他）

4. 複数ファイルを処理する方法

複数のファイルに対して同一の処理をしたい場合、bash の history を使って手打ちでファイル名を変えながら実行してもよいのですが、数が多くなると手間もかかるし間違いもしがちです。

一回のコマンド打鍵で複数ファイルを処理する方法はアイデア次第でいくつもあるでしょうが、見かけることが多いのは以下の方法です。

4.1. コマンドのディレクトリ再帰オプション

コマンドの中には入れ子になったディレクトリを再帰的に処理するオプションがある grep や再帰的な処理を前提として作られている find コマンド等があります。

例）bashrp という名前のディレクトリとそのサブディレクトリから、6 文字目か 11 文字目が a または A のレコードを含むファイルを探す

```
$ grep -iEm 1 -r  '(^.{5}a|^.{10}a)' ./bashwp
./bashwp/bash-text_tool.txt:324993284=account-8
./bashwp/temp-utf8-addCR.txt:あいうえお12345abcde
./bashwp/temp-utf8-delCR.txt:あいうえお12345abcde
./bashwp/testdata/data1.csv:1,aaaa,bbbb,cccc,ddd,あいう,えおか
```

オプション

-r : 引数に指定されたディレクトリを再帰的に処理（-R はリンク先も対象にする）

-m *n* : ファイル毎に整数 n で指定された件数のレコードが見つかる迄探す

-l :（L の小文字）ファイルのパスだけ出力 … 別のコマンドに渡して処理する際に便利

4.2. シェルの for 文

bash で複数のファイルに同一の処理を行う場合は、for ループが使えます。

＜書式＞

for *変数* in *ファイル名のリスト*

do

コマンド

done

※ ファイル名のリストはワイルドカードを使うと bash が展開してリストにしてくれます

また、do/done の前の改行をセミコロンに置き換えて下のように書いても同一の意味になります

for *変数* in *ファイル名のリスト*; do *コマンド*; done

例）一つのディレクトリに”meisai”+数字 1 桁＋”.csv”（数字部分は可変）という名前のファイルが複数格納されていて、この各ファイルのフィールド 4 と kamoku.csv のフィールド 1 で結合する

＜入力ファイル＞

```
$ ls /c/Users/User/Desktop/bashwp/join | column -t
kamoku.csv
meisai1.csv
meisai2.csv
```

Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.

Linux ファイル編集コマンド in Windows（grep,join,awk,sed 他）

＜コード例＞

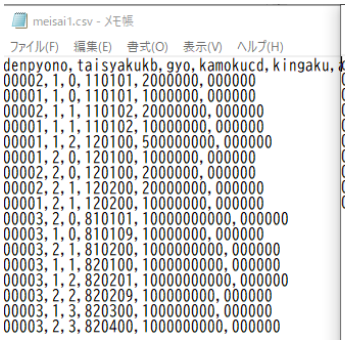
```
pushd /c/Users/User/Desktop/bashwp/join #ファイルのディレクトリにカレントパスを移動
for f in meisai[0-9].csv
do
 join -t, -1 4 -2 1 -a 1 --header "$f" kamoku.csv > "$f".new.csv
done
```

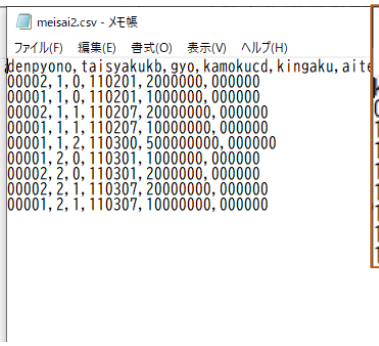
＜実行後＞

元のファイルに対応する 2 つの meisai[0-9].csv.new.csv が作られ、結合も正常に行われている。

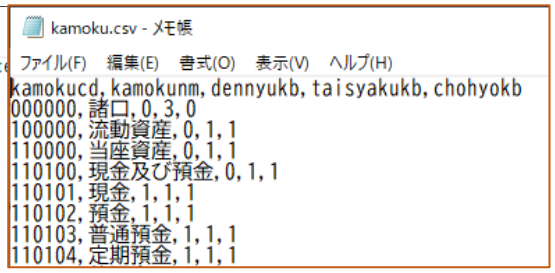
```
$ for f in *.csv; do ls $f ; done
kamoku.csv
meisai1.csv
meisai1.csv.new.csv
meisai2.csv
meisai2.csv.new.csv
```

meisai1.csv と meisai2.csv

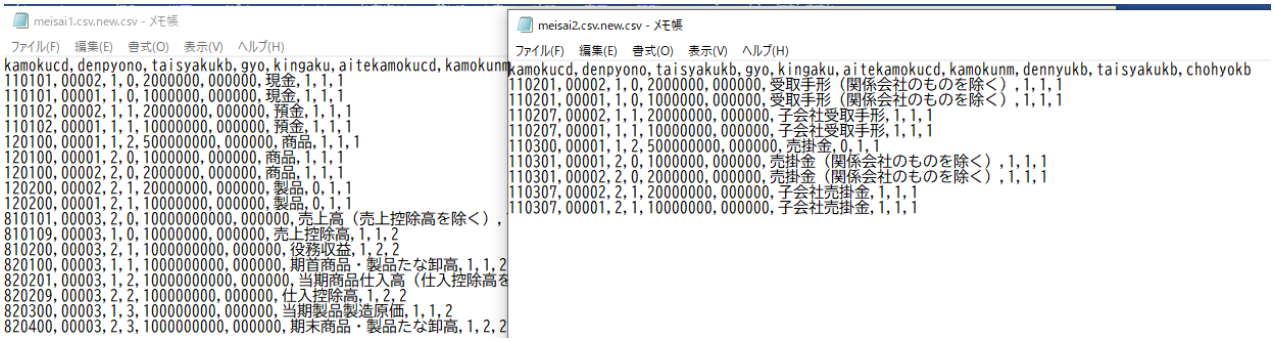




kamoku.csv



meisai1.csv.new.csv と meisai2.csv.new.csv



4.3. find コマンド

find コマンドは各種の条件（種類、名前の一部、作成／更新／参照からの経過時分 他多種）に該当するファイルやディレクトリの名前（ファイルパス）をリスト化します。このコマンドは古く Unix の複数の系統で機能追加が進められ、Linux でもオプションの有無等 若干異なる環境があります。

ここでは MSYS 2（開発団体は GNU）で使えて他環境でも使える可能性が高い方法を紹介します。

Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.

Linux ファイル編集コマンド in Windows（grep,join,awk,sed 他）

4.3.1. コマンドラインの共通的な注意点

コマンドは bash を通して呼び出されます。bash はコマンドを呼び出す前にコマンドラインを解析し、引用符で囲まれていない部分やエスケープされていない記述は展開を試みます。

例えば、bash は *、 ?、 [...] の 3 種をワイルドカードとして扱うため以下の例では echo コマンドに“*.csv”と言う文字列ではなくファイルのリストが渡されています。

```
$ echo *.csv
kamoku.csv meisai1.csv meisai1.csv.new.csv meisai2.csv meisai2.csv.new.csv
```

```
$ echo *[0-9].csv
meisai1.csv meisai2.csv
```

bash の展開を迂回するには、引用符で囲みます

```
$ echo "*[0-9].csv"
*[0-9].csv
```

また、ファイル名を扱う場合は以下の点も注意が必要です。

① 変数もコマンドに渡す前に展開したり置換される

② 記号や空白が含まれているファイル名を変数としてシェルで扱うとコマンドライン内で展開されて想定と異なる実行内容になる場合がある（Windows の場合は空白が問題になります）

以上のことから、必ずシェル変数は引用符で囲むようにします。

```
for f in meisai[0-9].csv
do
 join -t, -1 4 -2 1 -a 1 --header "$f" kamoku.csv > "$f".new.csv
done
```

4.3.2. ファイルのリストを他のコマンドで処理

xargs コマンドを通して別のコマンドで処理を行います。xargs は標準入力で受け取ったリストを別のコマンドのパラメータとして編集し、起動します。

例）ディレクトリパス /c/Users/User/Desktop/bashwp/join に格納されている拡張子が csv の全てのファイルに対して bin/sum.sh（自作のシェルスクリプト）を実行する

```
find /c/Users/User/Desktop/bashwp/join -name "*[0-9].csv" -type f -print0 | xargs -0 -n1 bin/sum.sh
```

find オプション

-name : ワイルドカードを含むファイル名で探す場合は引用符で囲む

-type : 編集エラーを防ぐため、一般ファイルに限定する（f 一般ファイル、d ディレクトリ）

-print0 : ファイル名が空白等を含んでも動作する（区切り文字と混同しない）ように、
リストの区切り文字を null(x00)にして出力

xargs オプション

-0 : null(x00)をファイルリストの区切り文字とする（find -print0 に対応）

-n : 呼出し先のコマンドに渡すパラメータの数を指定する（この例では、1 ファイル名ずつ呼出し）

Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.

Linux ファイル編集コマンド in Windows（grep,join,awk,sed 他）

【bin/sum.sh の内容】

```
echo $*
awk '
BEGIN{ FS=",";OFS=","; }
 NR==1{ print $4,$2,$5; next; }
 { key=($4 OFS $2); val[key]+=$5; }
 END{ for(key in val) print key, val[key] }
'  "$1"
```

＜実行例＞

```
$ find /c/Users/User/Desktop/bashwp/join -name "*[0-9].csv" -type f -print0 | xargs -0 -n1 bin/sum.sh
/c/Users/User/Desktop/bashwp/join/meisai1.csv
kamokucd,taisyakukb,kingaku
820209,2,100000000
820100,1,1000000000
120100,2,3000000
110101,1,3000000
820300,1,100000000
810109,1,10000000
810101,2,10000000000
810200,2,1000000000
120200,2,30000000
110102,1,30000000
820400,2,1000000000
820201,1,10000000000
120100,1,500000000
/c/Users/User/Desktop/bashwp/join/meisai2.csv
kamokucd,taisyakukb,kingaku
110301,2,3000000
110307,2,30000000
110201,1,3000000
110300,1,500000000
110207,1,30000000
```

4.3.3. find の exec オプションで処理

xargs コマンドは先行コマンドの標準出力の受け取り先を柔軟に変えられるのでよく使われますが、find コマンドには検索結果をパラメータにして別のコマンドを呼出す-exec オプションがあります。

例）ディレクトリパス /c/Users/User/Desktop/bashwp/join に格納されている拡張子が csv の全てのファイルに対して bin/sum.sh（自作のシェルスクリプト）を実行する

```
find /c/Users/User/Desktop/bashwp/join -name "*[0-9].csv" -type f -exec bin/sum.sh {} ¥;
```

find オプション

-exec：セミコロン（bash のコマンド区切りと被るので ¥ を付けエスケープ要）で終わっている場合は {} がファイル名 1 つに置換されて exec 以降に書かれたコマンドを実行する

※ 実行結果は xargs を使った場合と同一です

Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.

Linux ファイル編集コマンド in Windows（grep,join,awk,sed 他）

4.3.4.　性能向上のためのコマンド実行形式

xargs も find -exec もファイルリストが巨大になった場合「資源と性能のバランスを考慮したサイズに分割してコマンドを並列実行する」と開発元のドキュメントに書かれています。前項の例では自作のスクリプトが 1 ファイルだけ受け取る前提になっていましたが、大量のファイルを処理する場合はリストで受け取ってループさせた方がサブプロセスの発生数減と並列実行により性能向上が望めます。

この場合、コマンドラインと sum.sh は以下のようになります。

（1）コマンドライン

xargs、find -exec のオプションを以下のように変更します。

―― xargs のオプション -n1 を削除 ――――――――――――――――――――――――――――――

```
find /c/Users/User/Desktop/bashwp/join -name "*[0-9].csv" -type f -print0 | xargs -0 -n1 bin/sum.sh
  ↓
find /c/Users/User/Desktop/bashwp/join -name "*[0-9].csv" -type f -print0 | xargs -0 bin/sum.sh
```

―― -exec オプションの終端をセミコロンから + に変更 ――――――――――――――――――

```
find /c/Users/User/Desktop/bashwp/join -name "*[0-9].csv" -type f -exec bin/sum.sh {} ¥;
  ↓
find /c/Users/User/Desktop/bashwp/join -name "*[0-9].csv" -type f -exec bin/sum.sh {} ¥+
```

（2）スクリプト（sum.sh）の変更

ファイルリスト（=n 個のパラメータ）で受け取る場合は、ループの中でパラメータを shift させながら、パラメータ数が 0 になる迄実行します。

＜パラメータの取り扱い＞ $* : パラメータ群全体、$# : パラメータ数 ※ shift で$2⇒$1, $3⇒$2 …

```
echo "####" file-list $* "####"
while [ $# -gt 0 ]
do
    echo $1
    awk '
    BEGIN{ FS=",";OFS=","; }
     NR==1{ print $4,$2,$5; next; }
     { key=($4 OFS $2); val[key]+=$5; }
     END{ for(key in val) print key, val[key] }
    '  "$1"

    shift
done
```

1 回の呼出しでリストの全ファイルを処理するので、処理結果をファイルに保存する場合はループの中で行う必要があります。

＜実行結果の一部＞

```
$ find /c/Users/User/Desktop/bashwp/join -name "*[0-9].csv" -type f -exec bin/sum.sh {} ¥+
#### file-list /c/Users/User/Desktop/bashwp/join/meisai 1.csv /c/Users/User/Desktop/bashwp/join/meisai2.csv ####
/c/Users/User/Desktop/bashwp/join/meisai 1.csv
kamokucd,taisyakukb,kingaku
820209,2,100000000
820100,1,1000000000
120100,2,3000000
110101,1,3000000
```

Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.

Linux ファイル編集コマンド in Windows（grep,join,awk,sed 他）

4.3.5.　その他の一括処理機能

find コマンドのよく見る使い方は、最後の更新から所定時間を経過したログや受信ファイルの消去などの操作です。一般ファイルもディレクトリも一括して扱えて便利なのですが、動作の順番を意識しておかないと先にディレクトリを削除してから格納されているファイルを削除しようとしたり、格納ファイルをバックアップしてからディレクトリのバックアップを行おうとしてエラーを起こすことがあります。これらに関係する主なオプションを例示します。

（1）ファイルの移動

ディレクトリ階層を保って移動させる場合は、-maxdepth 1 で 2 階層下の探査を抑止します。また、起点に指定したディレクトリパスも検索結果に含まれるので、-mindepth 1 で除外します。

例）*親 dir* 配下で作成・更新日時（mtime）が 3 日以上（72 h ～過去）経過しているディレクトリやファイルを一階層上の backup ディレクトリに移動する

```
pushd 親dir  # 操作対象が格納されているディレクトリをカレントにする
find ./ -mindepth 1 -maxdepth 1 -mtime +2 -print0 | xargs -0 mv -t  ../backup
popd # カレントディレクトリの戻し
```

〔時間範囲〕以下の条件を使って指定します。

atime / amin : アクセス（ファイルの内容を読み取る）
ctime / cmin : ステータスを変更する（ファイルまたはその属性を変更する）
mtime / mmin: 変更する（ファイルの内容を変更する）
*x*time : 0=24 時間未満、1=24～48 時間前、+1=48 時間以上過去、-2=48 時間以内 …
*x*min : 2～6 分前に更新されたものを探すには、 -mmin +2 -mmin -6

〔その他の記法〕

MSYS 及び一般的な Linux ディストリビューションでは使えますが、使えない環境があります

① 他のファイルとの比較（newer）

```
find ./ -mindepth 1 -maxdepth 1 -newer 比較対象のファイル  …以降省略
```

② 日時文字列を使った比較（newerat、newwect、newermt）

```
find ./ -mindepth 1 -maxdepth 1 -not -newermt '3 day ago 00:00'  …以降省略
```

newer**mt** は **mt**ime がより新しいという意味です。-not は後に続く条件の否定で、この例では、「3 日前の 00:00 より新しい」以外（=より古い）ディレクトリとファイルが選択対象になります。

また、-not は ¥! （！が否定で、¥はエスケープ）と同一です。(¥! だけが有効な環境も存在します)

〔参考〕条件が正しいか否かを touch コマンドを使って確認することができます

-c 新規ファイルを作らない、-m 更新日を変更、-a アクセス日を変更、-d 変更する日時文字列

```
touch -cm -d '3 days ago 12:00' temp*
touch -cm -d '72 hour ago' temp*
```

※使用可能な日時文字列の形式は、下記 GNU のサイト（Date input formats）を参照

https://www.gnu.org/software/tar/manual/html_node/Date-input-formats.html

Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.

Linux ファイル編集コマンド in Windows（grep,join,awk,sed 他）

（2）find コマンドによるディレクトリ、ファイルの削除

ファイルが格納されているディレクトリを削除しようとするとデフォルトではエラーになるので、配下の一般ファイルを全て削除してからディレクトリを削除します。実は、これを行う機能は find コマンド自体に備わっていて、-delete アクションを指定するだけです。

例）親 dir 配下で更新日時（mtime）が 3 日（72h～96 h）経過しているディレクトリやファイルを削除する

```
pushd 親dir  # 操作対象が格納されているディレクトリをカレントにする
find ./ -mindepth 1 -mtime 3 -print -delete
```

＜実行例＞

```
$ find ./ -mindepth 1 -mtime 3 -print -delete
./temp-utf8-addCR.txt
./temp-utf8-delCR.txt
./temp-utf8.txt
./tempfolder/temp.csv
./tempfolder/temp.txt
./tempfolder
```

※ 下 3 行で配下のファイル削除の後で tempfolder ディレクトリを削除しているのが確認できます

以下のコマンドラインは、同一の結果になります。

```
find ./ -mindepth 1 -mtime +2 -print -delete
find ./ -depth -mindepth 1 -mtime +2 -exec rm -d {} +
```

※ -depth が指定されているとディレクトリ階層の下位から処理が行われるようになります

以上

Copyright(C)2024 Future Office Coordinate Service Corporation All Rights Reserved.