J. Plasma Fusion Res. Vol.84, No.1 (2008) 33-45

講座 オープンソースソフトウェアを使った実践データ解析

2．便利なツールの使い方

鈴木康浩
（核融合科学研究所）

（原稿受付：2007年11月14日）

2.1 はじめに

この章では，3章以降で紹介される様々なスクリプト言語やツールを使ったデータ解析を便利にするツールとその使い方を紹介します．ここで紹介するツールは，ほとんどがUNIXベースのオペレーティングシステムで標準で用意されています．Windowsでは Cygwin[1]か SFU（Windows Services for UNIX）[2]を利用すればよいでしょう．

2.2 awkでテキスト処理

計測データや計算データの多くは，データがいくつかの列にわたって記述されているか，行列になっている場合が多いと思います．そのようなデータである列だけを処理したい，ある部分だけを処理したいということが出てくると思います．商用のグラフソフトであれば，グラフを書くだけでなくデータ処理の機能も充実しています．ただし，そのような機能を使う場合，データフォーマットが，多くの場合，そのソフト独自のバイナリ形式になってしまい，データのやりとりをするためには，同じソフトを使用するかテキストファイルに変換して出力する必要があります．逆にデータがテキストで提供されている場合，テキストを処理するには，エディタで開くか別のツールが必要になります．一方，データ解析をするプログラムやスクリプトを編集する場合，ある単語を一括して置換したい場合が出てきます．機能を拡張する上で変数名を変更する必要がどうしても出てきてしまうので，やむを得ないでしょう．そのような時に活躍するのがawkです．もともと，awkはUNIX上で開発されたフィールド（表）指向のテキスト処理ツールです．つまり，データが数列にわたって記述されているファイルを処理するのが得意です．一方，awkに似た働きをするツールとして，grepや sedがあります．しかし，grepや sedはテキスト処理ツールでも，入力されたテキストを一括で処理します．ところが，awkは演算機能を持っており，スクリプト言語と遜色のないインタプリタとなっています．

では，細かい説明は文献を参照していただいて具体的な使い方を説明していきます．次のようなデータを考えます．

リスト1

```
1        s              reff[m]          Te[keV]          ne[/m^3]
2   0.0000000e+00    0.0000000e+00    0.5000000e+00    0.2000000e+20
3   0.1000000e+00    0.1897367e+00    0.4500000e+00    0.1999800e+20
4   0.2000000e+00    0.2683282e+00    0.4000000e+00    0.1996800e+20
5   0.3000000e+00    0.3286335e+00    0.3500000e+00    0.1983800e+20
6   0.4000000e+00    0.3794733e+00    0.3000000e+00    0.1948800e+20
7   0.5000000e+00    0.4242641e+00    0.2500000e+00    0.1875000e+20
8   0.6000000e+00    0.4647580e+00    0.2000000e+00    0.1740800e+20
9   0.7000000e+00    0.5019960e+00    0.1500000e+00    0.1519800e+20
10  0.8000000e+00    0.5366563e+00    0.1000000e+00    0.1180800e+20
11  0.9000000e+00    0.5692100e+00    0.5000000e-01    0.6878000e+19
12  0.1000000e+01    0.6000000e+00    0.0000000e+00    0.0000000e+00
```

リスト1のデータは電子温度と電子密度のプロファイルを示したものです．**先頭の列は行番号で，実際のデータではありません．** 1列目から規格化トロイダルフラックス，実効小半径，電子温度，電子密度がそれぞれ記述されています．このファイルをawkで処理してみましょう．awkの特徴は次の二つです．

・awkは行単位で処理をする．

1 http://www.cygwin.com/
2 http://www.microsoft.com/japan/technet/interopmigration/unix/sfu/default.mspx

Practical Data Analysis Using Open Source Software 2. How to Use Convenient Tools
SUZUKI Yasuhiro *author's e-mail: ysuzuki@LHD.nifs.ac.jp*

・awk は空白文字を区切り文字として取り扱う．

リスト 1 のデータを"data1.txt"という名前で保存したとします．では，次のようにコマンドを入力して下さい．**先頭の％はプロンプトです．**

```
% awk '{print $1}' data1.txt
```

すると，

リスト 2

```
 1  s
 2  0.0000000e+00
 3  0.1000000e+00
 4  0.2000000e+00
 5  0.3000000e+00
 6  0.4000000e+00
 7  0.5000000e+00
 8  0.6000000e+00
 9  0.7000000e+00
10  0.8000000e+00
11  0.9000000e+00
12  0.1000000e+01
```

と出力されます．awk はファイルを行単位で読み込みます．そして，行は空白をフィールド間の区切り文字としてレコードを構成します．例題のコマンドで'{print $1}'は 1 番目のフィールド[3]を表示せよと言う意味なのです．awk は，

```
awk 'コマンド'  ファイル名
```

という形で利用します．例題ではコマンド print を使用しました．コマンド print は 1 つだけでなく，いくつものフィールドを'print $1, $2'のように取り出すことができます．フィールドの区切り文字は，デフォルトで空白が指定されていますが，次のように変更することができます．

```
% awk 'BEGIN { FS="," } { print $1, $3}' data1.csv
```

FS は"Field Separator"の頭文字で，区切り文字を制御するための組み込み変数です．ファイル data1.csv は，区切り文字がカンマである CSV 形式のファイルです．変数 FS に","を代入することにより，カンマが区切り文字でもフィールドごとに取り出すことができます．変数 FS と共に覚えておくと便利な組み込み変数が NR です．変数 NR は行数を表し，

```
% awk 'NR > 1 {print $0}' data1.txt
```

のように用いれば，先頭の行をとばして処理することができます．ファイル data1.txt の先頭の行は，それぞれの列の物理量と次元が書かれています．通常，計測データのほとんどは，装置固有のフォーマットに従って，ファイルの先頭にヘッダ情報が書かれています．ヘッダ情報を取り除いて，数値情報だけの情報を取り出したい場合に awk は重宝するはずです．さらに，条件判定はフィールドに対しても行えます．ファイル data1.txt の 1 列目は，規格化トロイダルフラックス s です．s が 0.2 以上のデータだけを取り出したい場合は，

```
% awk '$1 >= 0.2 {print $0}' data1.txt
```

として下さい．s が 0.2 以上で，2 列目の実効小半径が 0.5 以下のデータも

```
% awk '$1 >= 0.2 && $ <= 0.5 {print $0}' data1.txt
```

とすれば取り出せます．&&は論理和です．論理演算は C 言語やシェルのそれと同等の文法で行えます．特定の列だけ演算したい場合は，どうすればよいのでしょうか？ 次の例題を考えます．

```
% awk '{print $1**0.5,$3*1000}' data1.txt
```

これは，1 列目の規格化トロイダルフラックスの平方根を取って規格化小半径にし，3 列目の電子温度の単位を keV から eV に変更します．例題を実行すると，

リスト 3

```
 1  0 0
 2  0 500
 3  0.316228 450
 4  0.447214 400
 5  0.547723 350
 6  0.632456 300
 7  0.707107 250
 8  0.774597 200
 9  0.83666 150
10  0.894427 100
11  0.948683 50
12  1 0
```

の様に出力されます．空白でフィールドを区切ってありますが，入力ファイルのフォーマットではありません．これは，awk の内部処理の問題で，数値を出力しようとしても実際は文字列として出力されてしまうからです[4]．出力精度を指定したい場合は，C 言語と同様に printf 文を使います．上の例題を printf 文を使って書き直すと次のようになります．

```
% awk '{printf "%5.4e %5.4e¥n",$1**0.5,$3*1000}' data1.txt
```

上のコマンドを実行した結果は，下のようなリストになります．

3 $0 はレコード全体を表します．
4 sprintf 関数を使っています．

リスト4

```
 1  0.0000e+00  0.0000e+00
 2  0.0000e+00  5.0000e+02
 3  3.1623e-01  4.5000e+02
 4  4.4721e-01  4.0000e+02
 5  5.4772e-01  3.5000e+02
 6  6.3246e-01  3.0000e+02
 7  7.0711e-01  2.5000e+02
 8  7.7460e-01  2.0000e+02
 9  8.3666e-01  1.5000e+02
10  8.9443e-01  1.0000e+02
11  9.4868e-01  5.0000e+01
12  1.0000e+00  0.0000e+00
```

ここまでの説明を応用するだけで，かなりのデータファイルの加工が行えるようになると思います．繰り返しになりますが，ここで行ったファイルの加工は既存のグラフソフトの数値エディタを使ってもできるものです．しかし，計測装置や数値コードから吐き出されるデータ形式はやたら変わるものではなく，一度，コマンドを決めてしまえば，同じ加工を繰り返すのは awk の方がはるかに効率的です．

awk は，変数が使えるので data1.txt の 3 列目と 4 列目から電子圧力を求めたりする計算が簡単にできます[5]．また，sed や grep といったコマンドを呼び出すこともでき，ループやフロー制御もできます．文献やインターネット等で調べていろいろ試してみてください．

2.3 make を使って楽々コンパイル

make というプログラムは，"Makefile"というファイルに記述されたルールに従ってプログラムのコンパイル，ファイルの更新を行うユーティリティです．

例題として，main.c と言うプログラムをコンパイルして main.exe と言う実行ファイルにすることを考えてみましょう．通常は，

```
% gcc main.c -o main.exe
```

と入力すれば望みの main.exe が作られるはずです．では，いくつかのコンパイルオプションやライブラリをリンクすることを考えてみましょう．次の場合を考えます．

```
% gcc -Wall -O2 -g main.c -lm -o main.exe
```

この例題では，デバッグ用のオプション'-Wall -O2 -g'を付けて，数値演算関数を使うためにライブラリ m をリンクしています．この程度の量なら，いちいちコマンド入力ですますこともできますが，さらに最適化オプションや多数のライブラリをリンクさせる場合には手間がかかります．そのような複雑なコンパイル作業を簡便にするのが make です．例題を make でコンパイルしてみましょう．この場合，次の内容を記述した Makefile という名前のファイルを作成します．

リスト5

```
1  main.exe: main.o
2          gcc main.o -lm -o main.exe
3
4  main.o: main.c
5          gcc -c -Wall -O2 -g main.c
```

1 行で済んだコマンド入力が 4 行になってしまいました．Makefile を作成するときに重要なことは，

ターゲット名： ソース名
　　コマンド

というルールです．このルールは「ターゲットはソースに依存し，そのソースによりターゲットが作成される．」を意味します．リストの Makefile 中で 3 〜 4 行目はオブジェクトファイル main.o は main.c により作成され，main.o をコンパイルするコマンドは，

```
gcc -c -Wall -O2 -g main.c
```

ということです．作成されたオブジェクトファイル main.o を元に 1 〜 2 行目のルールに従って，コマンド

```
gcc main.o -lm -o main.exe
```

により，数値演算関数 m をリンクして実行ファイル main.exe が作成されます．作った Makefile を main.c と同じディレクトリに置き，make とタイプしてみましょう．

```
gcc -c -Wall -O2 -g main.c

〜コンパイルメッセージ〜

gcc main.o -lm -o main.exe
```

環境にもよりますが，上記のようなメッセージが表示されて，プログラム main.exe が作成されたと思います．もう一度，make を実行してみましょう．そうすると，

```
make: 'main.exe' is up to date.
```

か

```
make: 'main.exe'は更新済みです
```

と表示されると思います[6]．では，main.c を変更してみます．適当な printf 文でも加えて，もう一度 make を実行してみてください．すると，

```
gcc -c -Wall -O2 -g main.c

〜コンパイルメッセージ〜

gcc main.o -lm -o main.exe
```

と出力され，実行ファイル main.exe が更新されます．つまり，make はどのファイルが更新済みかを自動的に判別し，再コンパイルを行うプログラムなのです．

5　あとの章で紹介するスクリプト言語を使った処理のほうが汎用性がありますが，awk だけでも処理できます．
6　環境変数 LANG に依存します．

ここまでの例題は，一つのソースファイルにライブラリを一つリンクさせるだけでした．このような簡単なプログラムならば，いちいちコマンドを入力してコンパイルしても，シェルスクリプトを作成してもさほど手間にはなりません．ところが，解析プログラムが大規模になるほど，機能が増えるほどプログラムのソースファイルは複数になり，ヘッダファイルやモジュール構造が複雑になるものです．このような複雑なプログラムの依存関係をルールに従って Makefile に記述しておけば，make は Makefile に従って自動的に更新を判別し，必要なものだけ再コンパイルしてプログラムを更新してくれます．では，複数のファイルから構成されるプログラムを make でビルドする例を考えてみましょう．次の例を考えます．

・プログラム main2 は main2.c，subA.c，subB.c，header.h から構成される．
・オブジェクトファイル main2.o は main2.c から作られる．
・オブジェクトファイル subA.o は subA.c から作られる．
・オブジェクトファイル subB.o は subB.c から作られる．
・header.h は subA.c,subB.c でインクルードされる．

上記のようなプログラムをコンパイルするためには，次のような Makefile を作ります．

リスト 6

```
1  main2.exe: main2.o subA.o subB.o
2     gcc main2.o subA.o subB.o -o main2.exe
3
4  main2.o: main2.c
5     gcc -c main2.c
6
7  subA.o: subA.c
8     gcc -c subA.c
9
10 subB.o: subB.c
11    gcc -c subB.c
12
13 subA.c: header.h
14 subB.c: header.h
```

この例は 4 〜11行目のルールに従ってオブジェクトファイル main2.o，subA.o，subB.o を作り，1 〜 2 行目のルールに従って実行ファイル main2.exe を作成します．注意してほしいのは13〜14行目のルールです．これはヘッダファイル header.h が subA.c，subB.c でインクルードされているために依存関係があるということです．もし，header.h が変更されれば，変更がプログラムに反映させる必要があるので，このようにルールを書いておけば make は subA.c と subB.c を自動的に再コンパイルしてくれます．このように，ファイルの更新は make に判別させることでプログラム作成者の負担を減らすとともに，必要なものだけ再コンパイルをすることで時間を節約することができます[7]．

これまで，簡単な Makefile の書き方と make の動作を見てみました．しかし，リストの書き方でソースファイルの数が増えていった場合，いちいち依存関係のルールを書き加えなければならないことになってしまいます．それでは本講座の目的であるお手軽とはいきません．この問題を解決する方法として，make にはサフィックスルールという機能が備わっています．サフィックスとは「拡張子」を意味します．リスト 6 の Makefile を以下のように書き換えます．

リスト 7

```
1  main2.exe: main2.o subA.o subB.o
2     gcc main2.o subA.o subB.o -o main2.exe
3
4  .c.o:
5     gcc -c $<
6
7  subA.o: header.h
8  subB.o: header.h
```

重要な点は，4 〜 5 行目でサフイックスルール".c.o"が追加されたことです．このルールにより拡張子".c"を持つファイルは拡張子".o"を持つオブジェクトファイルにコンパイルされます．サフィックスルールを使うことでプログラムを構成するファイルの数が増えても柔軟に対応できます．ただし，注意してほしいのはヘッダファイルに対する依存関係が，

```
subA.c: header.h
subB.c: header.h
```

ではなく

```
subA.o: header.h
subB.o: header.h
```

のようにオブジェクトファイルに対する依存関係に置き換わっていることです．このようにしないと make はうまく動いてくれません．さらにマクロを使って，リスト 7 を以下のように書き換えてみます．

リスト 8

```
OBJ=main2.o subA.o subB.o

main2.exe: $(OBJ)
        gcc -o $@ $(OBJ)

.c.o:
        gcc -c $<

subA.o: header.h
subB.o: header.h
```

マクロとは変数のように扱えるもので，1 行目で

```
OBJ=main2.o subA.o subB.o
```

7 一般に，コンパイラオプションに最適化命令を追加するとコンパイル時間は長くなります．

とマクロ OBJ にオブジェクトファイル名を代入しています．これを$(OBJ)のように引用することで，マクロ OBJ が自動的に展開されます．サフィックスルール$@にはターゲット名が自動的に代入されます．オブジェクトファイルのリストは，前節で学んだ awk と sed を使えば，

```
% ll *.c | awk '{print $9}' | sed s/.c/.o/
```

のように作れます．依存関係も makedep[8]を使えば簡単に調べることができます．

さらにマクロを使って改良してみましょう．

リスト9

```
1  CC=gcc
2  CFLAGS=-Wall -O2 -g
3  LIBS=-lm
4  PROGRAM=main2.exe
5  OBJ=main2.o subA.o subB.o
6
7  $(PROGRAM): $(OBJ)
8          $(CC) -o $@ $(OBJ) $(LIBS)
9
10 .c.o:
11         gcc -c $(CFLAGS) $<
12
13 subA.o: header.h
14 subB.o: header.h
```

リスト9のMakefileではコンパイラの名前やコンパイラオプション，リンクするライブラリ名などがすべてマクロで指定されています．さらにプログラム名もマクロ PROGRAM で指定しています．これにより元の実行ファイルを残したまま，ちょっとした修正を加えたバージョンや，異なるコンパイラで作成した実行ファイルを併存させる場合に楽になります．マクロには引数で値を与えることができるので，リストではコンパイラ名をマクロ CC で CC=gcc と GNU C コンパイラを指定していますが，

```
% env CC=icc make
```

と make を実行すればCコンパイラとしてIntel Cコンパイラを指定できるので便利です．

ここまではC言語のサンプルプログラムで make の使い方を説明してきましたが，make はC言語だけのユーティリティではありません．当然，他の言語でも make を使ってプログラムをコンパイルできます．Fortran などのプログラムの場合でも依存関係に注意して Makefile を書けば，問題なく使うことができます[9]．make の特徴を活かしてこんなこともできます．

リスト10

```
1  DOC    = main
2
3  LATEX   = c:/tex/bin/platex
4  DVIPDFM  = c:/tex/bin/dvipdfmx
5  DVIPS   = c:/tex/bin/dvipsk
6
7  DVIPDFMOPT = -v
8  DVIPSOPT  = -D600 -t a4 -P dl
9
10 .SUFFIXES: .tex .dvi .ps .pdf .sty
11
12 .tex.dvi:
13         $(LATEX) $<
14         $(LATEX) $<
15
16 .dvi.pdf:
17         $(DVIPDFM) $(DVIPDFMOPT) -o $@ $<
18
19 .dvi.ps:
20         $(DVIPS) $(DVIPSOPT) -o $@ $<
21
22 dvi: $(DOC).dvi
23
24 pdf: $(DOC).pdf
25
26 ps: $(DOC).ps
27
28 clean:
29         rm -f *.dvi *.aux *.log *.tag *.pdf
           *.ps *~
30
```

リスト10は LaTeX 処理用の Makefile です[10]．make を実行すると LaTeX ソースファイル main.tex がマクロ TEX で指定されたプログラムでコンパイルされ，dvi ファイルが作成されます．'make pdf'と実行すれば PDF ファイルが，'make ps'と実行すれば Postscript ファイルが作成されます．論文など複数の章から構成される文章でも，マクロであらかじめ

```
DOC=main chapter1 chapter2
```

と指定しておけば大丈夫です．

2.4 プログラムを管理する ～RCS と CVS～

前節では，プログラムをコンパイルするための make を紹介しました．make はあらかじめソースコードの依存関係を Makefile に記述しておけば，更新されたファイルを探

8 http://sourceforge.net/projects/makedep/

9 makedep と同じような結果が得られる Fortran90言語用の makedepf90 があります．http://personal.inet.fi/private/erikedelmann/makedepf90/

10 リスト10は Windows 環境上で角藤版 LaTeX の例です．

して自動的にコンパイルし直してくれるツールでした．ところで，データ解析のプログラムを作成していると，ついつい機能を追加したくなってくると思います．機能を追加して作業が充実してくるのは良いのですが，ミスをしてしまって前のソースに戻したくなることがあると思います．慎重な人は，プログラムに変更を加える前に必ずバックアップを取っていると思いますが，なかなか面倒な作業です．そのような作業を効率よく行うツールがRCSやCVSと呼ばれるツールです．

では，RCSからいきましょう．RCSは"Revision Control System"の頭文字を取ったもので，文字通りファイルのバージョン（更新履歴）を管理するプログラムです．RCSの使い方は簡単です．例題として，Linuxマシン上で"main.c"というC言語で書かれたプログラムのリビジョン管理をRCSですることを考えます[11]．プログラムmain.cは以下のような，簡単なプログラムです．

リスト11
```
1  #include <stdio.h>
2
3  void main()
4  {
5          printf("Hello World¥n");
6  }
7
```

まず最初にリビジョンを作ります．リビジョンを作るにはソースをチェックイン（Check-in）します．

```
% ci main.c
```

とコマンド入力してください．すると

```
main.c,v <-- main.c
enter description, terminated with single '.'
or end of file:
NOTE: This is NOT the log message!
>>
```

とメッセージが現れると思います．最後の行はコメント行なのでコメントを入力します．ここでは"original version"と入れることにします．Enterを押して.を入力しもう一度Enterを押すと

```
>> original version
>> .
initial revision: 1.1
done
```

とメッセージがでて，リビジョンの作成が完了します．最後に

```
initial revision: 1.1
```

と出力されますが，リビジョン番号の初期値は1.1です．ファイルを確認するとmain.cというソースコードが消えて"main.c,v"というファイルが新しく作成されていると思います．main.c,vの中は，以下のようになっています．

リスト12
```
1  head   1.1;
2  access;
3  symbols;
4  locks; strict;
5  comment  @*@;
6
7
8  1.1
9  date   2007.11.13.04.04.00;       author
          suzuki;    state Exp;
10 branches;
11 next   ;
12
13
14 desc
15 @orignal version
16 @
17
18
19 1.1
20 log
21 @Initial revision
22 @
23 text
24 @void main()
25 {
26         printf("Hello World");
27 }
28 @
```

もともとのソースコードに加え様々な情報が加わっています．それぞれの意味は飛ばして，とにかくどのように動くか検証してみましょう．main.cというソースコードがなくなってしまってはどのようにプログラムを編集すればよいのでしょう．プログラムを編集するには，main.cをチェックアウト（Check-out）する必要があります．チェックアウトは以下のように行います．

```
% co -l main.c
```

すると以下のようなメッセージが出てチェックアウト作業が完了します．

```
main.c,v --> main.c
revision 1.1 (locked)
done
```

カレントディレクトリにソースコードmain.cが作成されていると思います．main.cを以下のように書き換えてみます．

11 Windows環境上ではCygwin等を使えば良いでしょう．

リスト13
```
1  #include <stdio.h>
2
3  void main()
4
5          printf("Hello World\n");
6          printf("Hello World\n");
7  }
8
```

printf 文をもうひとつ追加しただけの簡単な変更です．変更を RCS に反映させるには，もう一度チェックインします．コマンドは最初のチェックインと同じです．コメントも好きなように入れてください．するとリビジョンが1.2に番号が上がってチェックイン作業が完了します．main.c，v の中を見てみて下さい．情報が追加されていると思います．ここでも細かい説明は省いて，RCS を使うとどのように簡単に作業の履歴が確認できるか見てみます．RCS には履歴を確認するためのツールがいくつかあります．まずはコマンド rlog です．このコマンドの動作は以下のようになります．

リスト14
```
% rlog main.c

RCS file: main.c,v
Working file: main.c
head: 1.2
branch:
locks: strict
access list:
symbolic names:
keyword substitution: kv
total revisions: 2;   selected revisions: 2
description:
original version
----------------------------
revision 1.2
date: 2007/11/01 01:49:43; author: suzuki; state: Exp; lines: +1 -0
second version
----------------------------
revision 1.1
date: 2007/11/01 01:36:08; author: suzuki; state: Exp;
Initial revision
=============================================================================
```

何時，誰がファイルを変更したのかすぐにわかります．具体的にどこを変更したかを見たい時には，コマンド rcsdiff を使います．このコマンドの動作は，次のようになります．

リスト15
```
% rcsdiff -c -r1.1 -r1.2 main.c
==========================================
RCS file: main.c,v
retrieving revision 1.1
retrieving revision 1.2
diff -c -r1.1 -r1.2
*** main.c    2007/11/01 01:36:08   1.1
--- main.c    2007/11/01 01:49:43   1.2
***************
*** 1,4 ****
--- 1,5 ----
 void main()
 {
    printf("Hello World");
+    printf("Hello World");
 }
```

'-c'を付けるのはおまじないと思ってください．リビジョン間の差をみるために'-r1.1 -r1.2'を付けています．これはリビジョン 1.1 と 1.2 の差を見なさいという命令です．先頭に + がついている行が変更箇所です．どこをどう変えたのかがよくわかるシステムになっていると思います．

では，話を CVS に移したいと思います．RCS は非常に優れたプログラムなのですが，欠点もあります．最大の欠点は，基本的に RCS は一つのファイルのリビジョンを管理するためのプログラムで，複数のファイルから構成されるプログラムのリビジョン管理をするのが苦手です[12]．そこで

12 できないわけではありません．

登場したのがCVS（Concurrent Versions System）です．CVSとRCSの最大の違いは，CVSはソースコードをリポジトリ（repository）という単位で管理する点です．リポジトリはファイルの数が限定されておらず複数のファイルを含むことが可能です．CVSはRCSを元に構成されたプログラムで，ファイル形式も共通です．最初，ソースコードをRCSで管理していたけれど，ファイルの数が増えてきたらCVSに移行することも可能です．

早速，CVSを使ってみましょう．ここでもRCSと同様に，まずLinux上での利用を考えます．CVSはRCSに比べるといくつか環境変数等の設定やディレクトリの作成が必要です．CVSは，クライアント・サーバーシステムとして動作するので，実行にはrshを利用します．最近は，セキュリティ上の理由からrコマンドの利用は制限されている場合が多いと思いますので，本講座ではsshの利用を推奨します．まず，リポジトリを管理するディレクトリを用意します．今回はホームディレクトリ上にCVSというディレクトリを用意します．次に環境変数を設定します．

```
CVS_RSH=ssh
CVSROOT = : ext : suzuki @ 192.168.0.1 : / home / suzuki/CVS
```

1行目はrshの代わりにsshを利用しなさいという意味です．2行目はCVSリポジトリの場所とリポジトリが存在するサーバーを指定します．IPアドレスは実際のIPアドレスかホスト名を入力して下さい．

では，CVSを使ってみましょう．まず，CVSを初期化します．環境変数CVSROOTで指定したリポジトリのディレクトリに移り，次のように初期化コマンドを実行します．

```
% cvs init
```

しばらくしてプロンプトが帰ってきたら初期化は終了です．初期化が終了すると，CVSROOTというディレクトリが作成されていると思います．ディレクトリCVSROOT以下には，拡張子",v"のたくさんのファイルが作成されているはずです．これらのファイルがリポジトリを管理するための設定ファイルになります．CVSの特徴はこれらの設定ファイル自体もCVSで管理されていることです．また，拡張子,vから推測されるよう，CVSはRCSと同じ形式で履歴を管理します．次に，リポジトリを作成します．リポジトリはプロジェクト毎に作成するので，まずプロジェクト名を決めます．ここでは，例題としてプロジェクト名をtestとします．プロジェクトtestでは次の5つのファイル，Makefile，main.c，subA.c，subB.c，header.hが含まれるとします．新しいプロジェクトを作成するには，上記の5つのソースコードを置いてあるディレクトリ上で，

```
% cvs import test vox start
```

と入力します．すると，エディタが開いて以下のようなメッセージが表示されると思います[13]．

リスト16

```
CVS: ----------------------------------------------------------------------
CVS: Enter Log. Lines beginning with 'CVS:' are removed automatically
CVS:
CVS: ----------------------------------------------------------------------
```

ここではコメントを入力できます．行頭のCVS:はコメント行を表します．今後，プログラムを編集して変更をリポジトリに反映させる度にこの画面がでますので，変更点がわかるようにコメントを入力すると作業の流れがわかりやすくなります．コメントを入力してエディタを終了すると，以下のようにメッセージが表れ，リポジトリの作成が終了します．

リスト17

```
% cvs import test vox start

message unchanged or not specified
a)bort, c)ontinue, e)dit, !)reuse this message unchanged for remaining dirs
Action: (continue)
N test/main.c
N test/Makefile
N test/subB.c
N test/header.h
N test/subA.c

No conflicts created by this import
```

環境変数CVSROOTで指定したディレクトリを見ると新しくディレクトリtestが作成されて，test以下に拡張子,vがついたファイル群が作成されていると思います．RCSと同様に，拡張子,vがついたファイルにこれからの変更履

13 ここで開かれるエディタは環境に依存します．利用するエディタは環境変数CVSEDITORで指定できます

歴が記録されていきます．

では，プログラムを更新して，履歴を反映させてみましょう．CVS の作業では以下の流れで行います．

1．登録されているプロジェクトの読み出し（チェックアウト）
2．編集
3．変更箇所の反映（コミット）

まず，プロジェクトを読み出します．チェックアウトは以下のように行います．念のため適当な作業ディレクトリに移って，以下のコマンドを実行します．

```
% cvs checkout test
```

すると，以下のようなメッセージが表示され，カレントディレクトリにプロジェクトがリポジトリから読み出されます．

リスト18

```
% cvs co test
cvs checkout: Updating test
U test/Makefile
U test/header.h
U test/main.c
U test/subA.c
U test/subB.c
```

CVS のコマンドの基本は，

cvs コマンド オプション

です．checkout は co と省略することもできます．任意のディレクトリにチェックアウトしたい場合は，

-d ディレクトリ名

をオプションとして指定します．RCS の場合，チェックアウトするとカレントディレクトリにファイル自体が生成されましたが，CVS はプロジェクト名がついたディレクトリごと読み出されることに注意して下さい．作成されたディレクトリ test 以下には元々の 5 つのファイルとディレクトリ CVS が作成されています．ディレクトリ CVS 以下には設定情報が記されたファイルが格納されます．次に，プログラムの変更をリポジトリに反映させてみましょう．ここでは，subA.c を変更します．関数 printf を追加するか，コメントを入れるかして修正してください．では，修正点をコミットしてみましょう．コミットするには以下のコマンドを実行します．

```
% cvs commit
```

すると以下のようなメッセージが表示されると思います．

リスト19

```
CVS: ----------------------------------------------------------------------
CVS: Enter Log. Lines beginning with 'CVS:' are removed automatically
CVS:
CVS: Committing in .
CVS:
CVS: Modified Files:
CVS:  subA.c
CVS: ----------------------------------------------------------------------
```

プロジェクトをインポートしたときと同じようにコメントの入力を促されるのですが，変更されたファイルとして subA.c が表示されます．CVS はリポジトリに格納されている，最新のファイルと比較して，変更されたファイルがあるかどうかを自動的に判別します．コメントを入力してインポートの時と同様に作業を進めると，以下のようなメッセージが出力されてコミットが終了します．

リスト20

```
% cvs commit
cvs commit: Examining .

message unchanged or not specified
a)bort, c)ontinue, e)dit, !)reuse this message unchanged for remaining dirs
Action: (continue)
Checking in subA.c;
/raid1/CVS/test/subA.c,v <-- subA.c
new revision: 1.2; previous revision: 1.1
done
```

subA.c のリビジョンが 1.2 に上がりました．試しに subA.c と subB.c の両方に変更を加えて見ましょう．適当な変更を加えてもう一度コミットします．すると，

リスト21
```
CVS: ----------------------------------------------------------------------
CVS: Enter Log. Lines beginning with 'CVS:' are removed automatically
CVS:
CVS: Committing in .
CVS:
CVS: Modified Files:
CVS:  subA.c subB.c
CVS: ----------------------------------------------------------------------
```

とsubA.cとsubB.cの変更ログを入力するように促された後,

リスト22
```
% cvs commit
cvs commit: Examining .

message unchanged or not specified
a)bort, c)ontinue, e)dit, !)reuse this message unchanged for remaining dirs
Action: (continue)
Checking in subA.c;
/raid1/CVS/test/subA.c,v <-- subA.c
new revision: 1.3; previous revision: 1.2
done
Checking in subB.c;
/raid1/CVS/test/subB.c,v <-- subB.c
new revision: 1.2; previous revision: 1.1
done
```

としてコミットが終了します．cvsはクライアント・サーバー型のシステムになっているので，複数のメンバーで一つのリポジトリを編集することが可能です．そのときに誰が何をどのように変更したかを確認できること，他の人が加えた変更を自分のソースに反映させる仕組みが必要です．cvsはそのようなことが簡単にできるようになっています．すべての変更履歴を見るにはコマンド`log`を使います．

```
% cvs log
```

と入力すると，すべてのログが表示されます．また，リポジトリに格納されている最新のソースコードと自分のソースコードの差を見るには

```
% cvs status
```

を実行します．もし，リポジトリのソースより自分のソースが古ければ,

```
% cvs update
```

により最新のソースに更新できます．もし，自分が加えた変更がリポジトリに反映されていない場合，コミットしてリポジトリに変更点を反映させてください．

ここまではLinux環境を念頭にcvsの利用を考えてきました．したがって，すべて端末上でコマンドを入力することにより利用します．ところが，cvsは何もLinuxをはじめとするUNIX系の環境のためのものではありません．たとえばWindows環境上で完結するcvsとしてWinCVSがあります[14]．WinCVSは，通常のWindows系のソフトと同様にGUI環境を持っていますので，より気軽に利用することが可能です．**図1**は，WinCVSのスクリーンショットです．Windows95時代のエクスプローラ感覚でcvsを使うことが可能です．また，なれている人は環境変数を設定すれば，コマンドプロンプト上でコマンドラインベースで利用することも可能です．WinCVSのページでは，MacOS X版やgtk版のGUIベースのcvsも配布されているので，興味がある人は見てください．

cvsは，本来，大規模なプロジェクト開発を効率よく行うために開発されました．本講座で紹介するソフトウェアは，そのほとんどが複数の開発メンバーによって開発・維持されています．そのような開発形態では，あるメンバーが他のメンバーの開発状況を確認でき，かつ他の人の開発結果を共有する仕組みが必要です[15]．そのような場面にお

14 http://www.wincvs.org/

15 cvswebというプログラムを使えば，Webでのcvsを使うことができます．次のURLを見て下さい．http://www.freebsd.org/cgi/cvsweb.cgi/

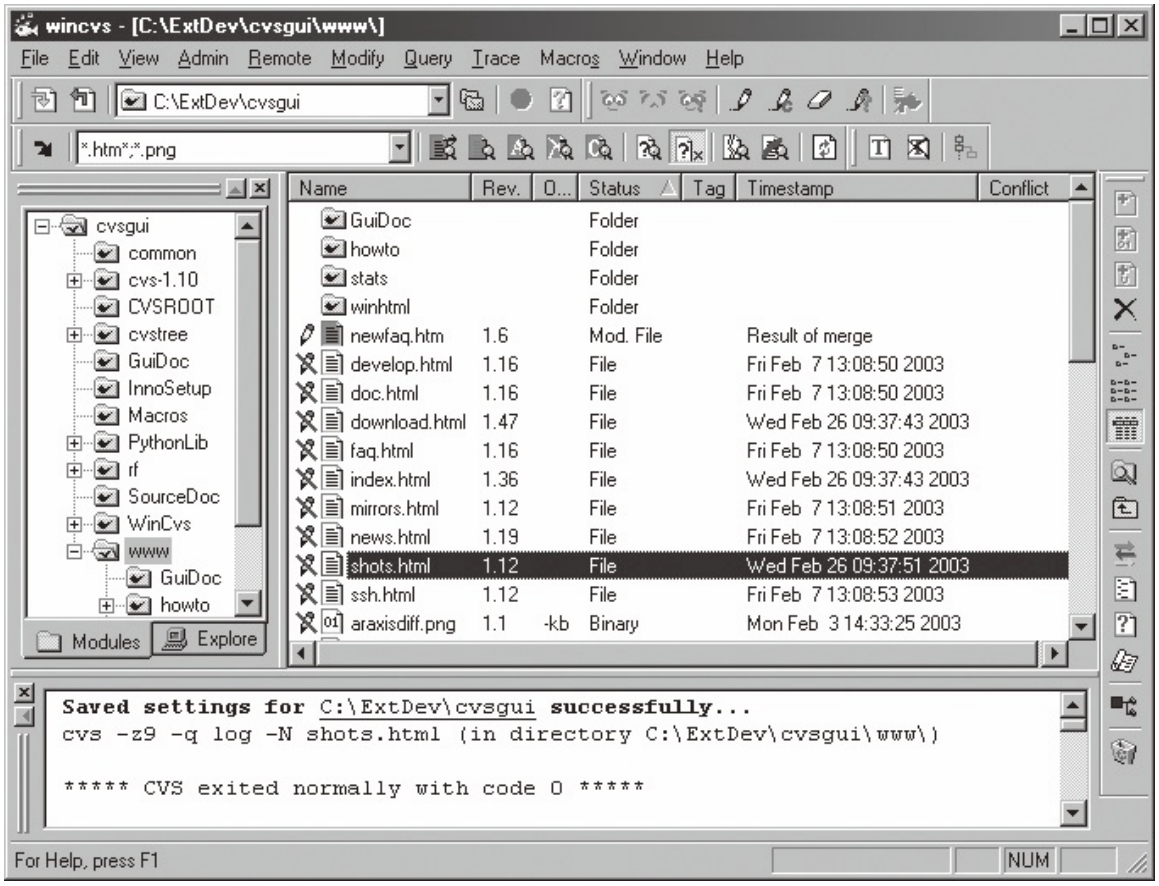

図1　WinCVS のスクリーンショット

いて，CVS が非常に役に立つことは想像に難くありません[16]．本講座では，そのような大規模なプロジェクト開発を対象にはしませんが，CVS のクライアント・サーバー型のシステムは個人ベースの作業でも役に立ちます．まず，CVS のリポジトリを作ります．そうすると CVS と ssh の使える環境であれば，どこからでもリポジトリをチェックアウトすることにより最新のソースコード，あるいはドキュメントに同期することができます．

最後に，subversion[17]について紹介します．CVS は便利なシステムですが，RCS をベースにしているために，少々，実装が古くなってきています．特に，リポジトリにディレクトリ構造を含む場合に，その取り扱い（移動や削除）が不便という欠点がありました．もう一度リスト22を見ていただきたいのですが，複数のファイルが存在するリポジトリの場合，ファイルのリビジョン番号が統一されません．一方，subversion では複数のファイルが存在するリポジトリにコミットがなされた場合に，全体のリビジョン番号が一斉に上がります．これは，見方によっては非常にわかりやすいシステムといえると思います．ディレクトリに対する操作が楽なことや，すべてのファイルのリビジョン番号の管理が行えるのは，subversion が Berkeley DB[18]を元にしたシステムだからです．CVS は RCS をベースにしたシステムですので，ファイル単位のリビジョン管理は RCS と同様にファイル単位で行っていました．RCS の制約上，すべてのファイルのリビジョン管理を同期させることは難しかったのです．しかし，subversion では，RCS から解き放たれたことにより，ファイル間でのリビジョン番号の同期やディレクトリ操作などが簡単になっています．

subversion は CVS からの移行が簡単になっています[19]．リポジトリの作成だけ svnadmin というツールを使いますが，それ以外の操作はコマンド svn を使います．例えば，コミットは以下のように行います．

```
% svn commit
```

つまり，cvs の代わりに svn を使えば，ほとんどの操作が共通に行えます．CVS と subversion のどちらがよいかは好みの問題です．筆者は CVS をメインに使っていますが，一部の計算コードのリポジトリを subversion に移行し始めています．subversion は，Web との親和性が高い

16　いくつかの BSD 系 OS（NetBSD，OpenBSD）は CVS を使って開発されています．CVS で常に最新のソースツリーに同期させることができます．

17　http://subversion.tigris.org/

18　http://www.oracle.com/database/berkeley-db/index.html

19　CVS から移行するためのツール cvs2svn が公開されています．http://cvs2svn.tigris.org/

ので，海外の共同研究者との共同開発が便利だからです[20].

2.5 統合開発環境を使ってみよう

これまで，便利なツールの使い方ということで，いくつかのツールを紹介してきました．どちらかというとコマンドラインベースで駆使するものだったのですが，ここでは変わって統合開発環境を紹介します．統合開発環境とは，コンパイラ，エディタ，デバッガを GUI 環境から統一して利用できるものと考えてください．確かに make は便利なシステムですが，Makefile にコンパイルルールと依存関係をきちんと書かなくてはなりません．プログラムを編集しつつ Makefile を自動的に書いてくれ，クリック一発でコンパイルできれば，こんなに便利なことはありません．これを可能にしてくれるシステムが統合開発環境です．

統合開発環境で有名なのは，Microsoft 社の VisualStudio[21]でしょうか．しかし，VisualStudio は製品でオープンソースではありません．オープンソースで開発されている統合開発環境はいくつかありますが，ここでは最も有名な Eclipse[22]を紹介します．Eclipse は，もともと IBM で開発された統合開発環境で，紆余曲折を得た後に非営利団体である"Eclipse Foundation"で開発が続けられています[23]．Eclipseの特徴は，機能がプラグインという形で提供されるという点です．もともと，Eclipse は JAVA 開発を行うために開発されました．従って，Eclipse 自体が当初から持っている機能は JAVA 開発のみです．ところが，サードパーティーや他プロジェクトが提供しているプラグインを使えば，C 言語や C＋＋言語，Python をはじめとするスクリプト言語や Fortran まで扱えます．

図 2 は Eclipse のスクリーンショットです．Eclipse にプラグイン photran[24]を組み込んだものです．編集中のプログラムのコンパイル作業が表されています．**図 2** は，英語環境ですが，当然，日本語化することは可能です．**図 3** に，筆者の日本語化された環境のスクリーンショットを示します．Texlipse という LaTeX用のプラグインを組み込んで作業しているところです．設定用のダイアログも含めて日本語化されているのがわかります．

インストール方法や細かい使い方は紙面の都合で割愛します．ただし，基本的に GUI 環境ですのでほとんどの作業は簡単に行えると思います．Web 上に Eclipse 関係の情報はたくさんあるので，興味がある人は調べていろいろ試してください．

参 考 文 献

［1］D. Dougherty and A. Robbins : sed & awk 改訂版（オライリー・ジャパン，1997）.

［2］A. Oram and S. Talbott: make 改訂版（オライリー・ジャパン，1997）.

［3］R. Mecklenburg: GNU Make 第 3 版（オライリー・ジャ

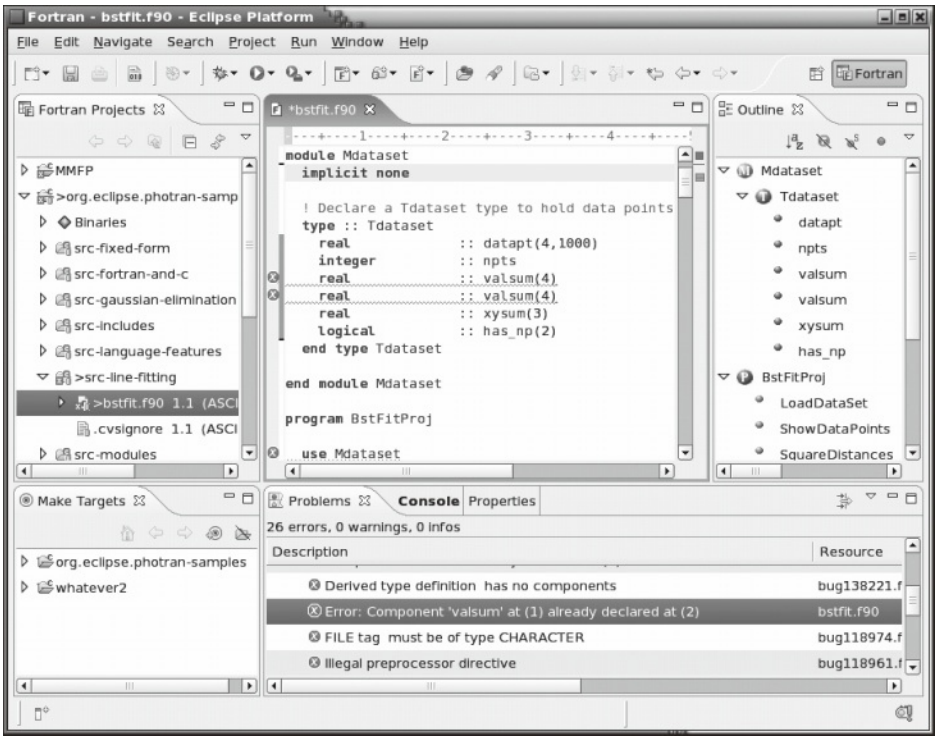

図 2　Eclipse のスクリーンショット

20　どちらがよいか多数決で負けたというのもありますが.orz…

21　http://www.microsoft.com/japan/msdn/vstudio/

22　http://www.eclipse.org/

23　企業が絡んだオープンソース開発の難しさを物語っています．

24　http://www.eclipse.org/photran/

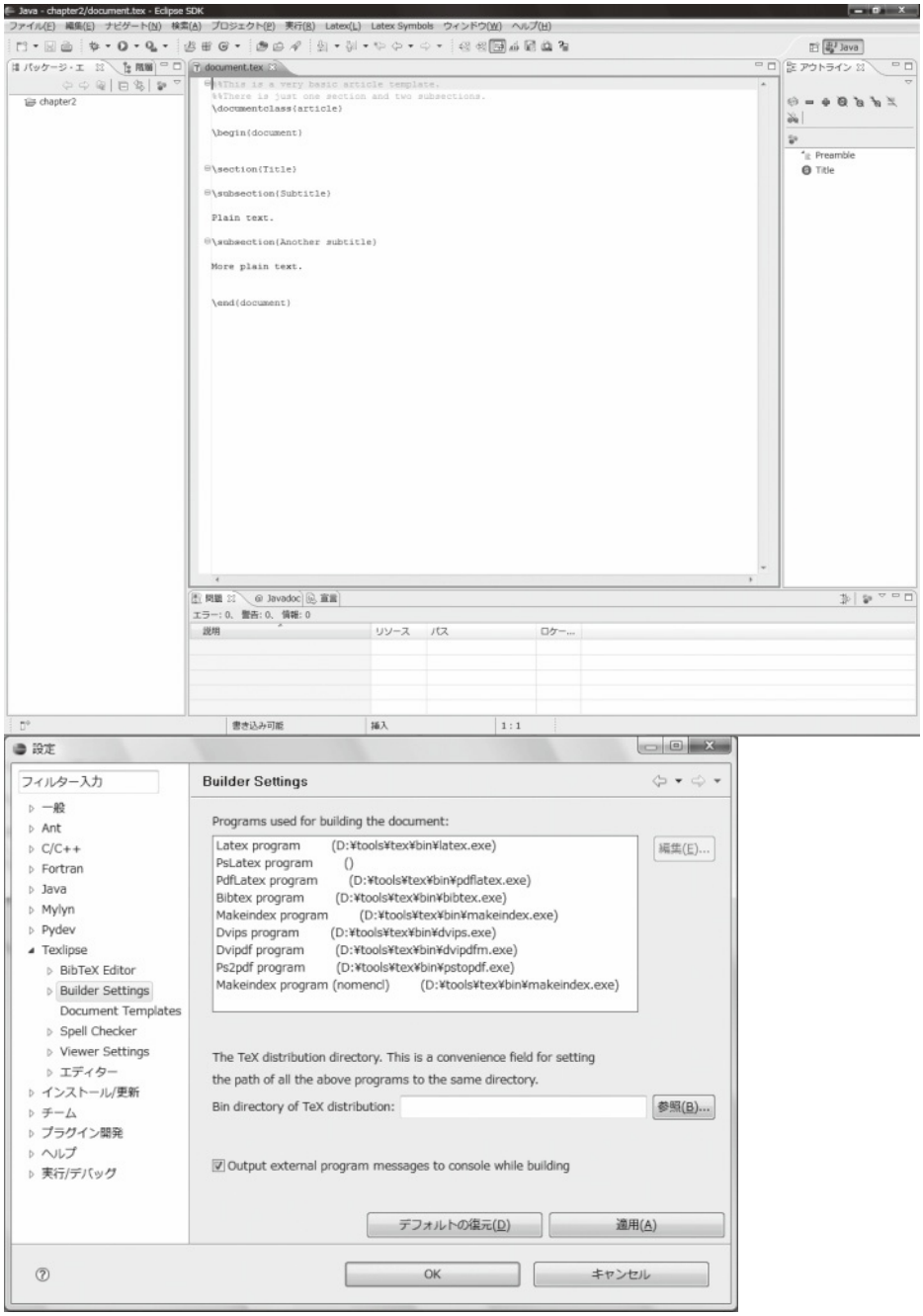

図 3　Eclipse のスクリーンショット 2

パン，2005）．
［4］Official RCS Homepage (http://www.cs.purdue.edu/homes/trinkle/RCS/).
［5］鯉江英隆，西本卓也，馬場 肇：バージョン管理システム（CVS）の導入と活用（ソフトバンクパブリッシング，2000）．
［6］大月美佳：入門 CVS 第 2 版（秀和システム，2002）．