月刊 世界で唯一の

本誌正式名称は「シェルスクリプトマガジン vol.25」です

Shell Script Is The Supreme Glue Technology
FOR THE SOPHISTICATED SHELL SCRIPTERS

シェルスクリプトマガジン

よりみち号

2015
5 May
Vol. 25
¥500

電子工作ができなくても大丈夫！
CD-ROMドライブではじめるかんたんIoT

スズラボ通信

未来に活きる！現場で使える！
データモデリング

めざせシェル女子！
貝殻高校、パソコン部の日常

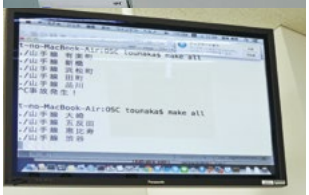
よしおかひろたかのIT散歩日記

特集

「シェルのプロが語る、make」

電車で剛！makefileを実行してみよう！

CONTENTS



世界で唯一の
月刊シェルスクリプトマガジン
2015 May vol.25

04 **特集 シェルのプロが語る、make**
電車で剛! Makefileを実行してみよう!

14 **よしおかひろたかのIT散歩日記　第2回**
よしおかひろたか

16 **電子工作ができなくても大丈夫!**
CD-ROMドライブではじめるかんたんIoT
あっきぃ

20 **ユニケージ開発手法コードレビュー　第14回**
大内智明

25 **姐のBENTO**

26 **法林浩之のFIGHTING TALKS　第12回**
法林浩之

28 **ITエンジニアのためのマーケティング入門　第2回**
水間丈博

31 未来に活きる!現場で使える!
データモデリング　第15回
熊野憲辰

34 **人間とコンピュータの可能性　第25回**
大岩元

36 **漢のUNIX　第21回**
後藤大地

42 **縁の木、育てよう 第8回**
白羽玲子

44 **スズラボ通信　第17回**
すずきひろのぶ

50 中小企業診断士が解説する、超実践的な会話術!
円滑コミュニケーションが世界を救う! 第3回
濱口誠一

52 **めざせシェル女子!　第2貝**
～貝殻高校、パソコン部の日常～
ちょまど

58 **Tech数独**

59 **「いんたーねっと」**
桑原滝弥

61 **サイバーフィジカル社会で必要なこと**
シェル魔人／編集後記

ちんじゅうちゃん

本誌名称について
vol.21より「USPMAGAZINE」から「シェルスクリプトマガジン」に、変更となりました。バックナンバーは「USP MAGAZINE」としてお求めいただけます。
本誌正式名称は「シェルスクリプトマガジンvol.25」となります。


稲苗月
今月号の特集は、シェルのプロ達に「make」について語っていただきました！
早速makefileを実行してみよう！

「シェルのプロが語る、make」

電車で剛! makefileを実行してみよう!

makeコマンド。make、というと物を作るというイメージだけど、Unixの世界でのmakeコマンドとは一体なんでしょう。まずはUnixのマニュアル、manの解説から見てみよう!

【日本語 man による説明から引用】

makeユーティリティの目的は、大きなプログラムの中に再コンパイルする必要がある部分を自動的に決定し、再コンパイルのためのコマンドを実行することである。(中略)これは、Richard StallmanとRoland McGrathが書いたものである。(中略)makeは、シェルコマンドからコンパイラを起動できるどんなプログラミング言語とでも組み合わせて使用できる。実際、makeの利用対象は、プログラムだけに限られない。makeは、あるファイルを書き換えたら、その書き換えたファイルを元にして別のファイルも自動的に更新しなければならないような任意の作業で利用できる。makeを使う準備をするためには、まずmakefileと呼ばれるファイルを書かなければならない。このファイルは、プログラムを構成するファイル間の関係と各ファイルを更新するためのプログラムを記述したものである。プログラムの場合は普通、実行ファイルはオブジェクトファイルによって更新され、このオブジェクトファイルもまたソースファイルのコンパイルによって生成される。

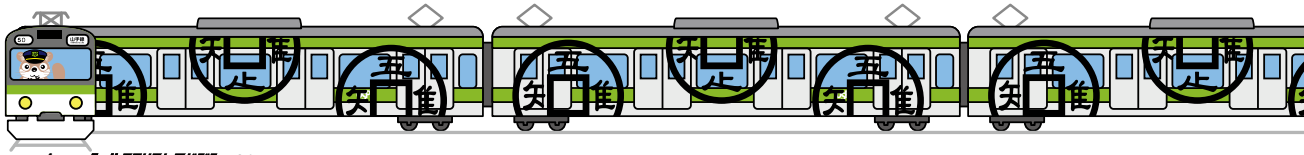
適切なmakefileさえあれば、ソースファイルを一部変更するたびに、

make　という、

簡単なシェルコマンドを実行するだけで、必要な再コンパイルが全て行われる。

makeプログラムは、makefileデータベースとファイルの最終更新時刻を用いて、更新する必要があるファイルを見つける。このようなファイルに対して、makeはデータベースに記録されているコマンドを実行する。

う〜ん、難しい!コンパイルって、人間が書いたコードをコンピュータが理解できるか、実行できる形にするってことだよね。実際どういう風に使われるんだろう。あ、そうだ。こないだOSCってイベントでmakeのお話があったみたいなんだ。廊下に人が溢れるぐらいのすごい大盛況だったんだって。面白そう!そのときの再録があるみたいだから、ちょっと見てみよう!



シェルのプロ エージェントその1 とうなか

みなさんこんにちは！ 今日は実を言うと昨日まで本当に体調が悪くて、とても無理だから辞退しようと思っていたんですよ。ドタキャンッ。だから資料も作っていなくて。だけど今朝起きたら何故か頭がすっきり！それで「よし、行こう！」と決め、今朝電車の中で資料を作ったのです。

電車の中で資料を作るってすごいね。

今日の話題はですね。「make」。我々UNIX屋、シェル屋でございまして、シェルとmakeで出来る事。いわゆるOSのビルドであったりなど、そういったものに使われるmakeです。これについてお話します。まずは自己紹介させてください。私なんですけれども、當仲（とうなか）と申しまして、USP研究所の代表をやっています。趣味は食べ物、旅行、運動、まあ普通なんですけれどもこんな人間です。いたって普通の人です。USP研究所というところは変わっているところでして、どんなシステムでもシェルスクリプトだけでシステムを作るシェル狂信集団。JavaもRubyも何も使わないという。世の中がどの方向に進もうと、いつでもどこでもシェルスクリプトでやる。なんとデータベースさえも使わない。ありふれたDB、分散型とか、オンメモリだとか、いろいろありますが、いっさい使わないで、ファイルシステムだけを見続けてやる。そういうちょっと変わった人たちです。

加えて、AWK大好き。本日さいとうさんをお呼びしたのはそんな理由です。

今日のお題、「make」ですね。OSのビルドとして使うのではなく、システムのリカバリに使うという話です。コンピュータを長年やっていると、障害などで途中でシステムが止まることがよくありますね。夜中にたたき起こされて「やれ、直せ!!」と、こう来るわけです。仕事ですからね。自分が作ったものだったらまだしも、他人が作ったものも同様に来る。そしてその場で直せと理不尽な状況に遭遇するわけです。そうなると当然、どこが壊れているのか、一生懸命見て、必要ならばリカバープログラムなどを作って、その場でリカバーしてみせる。これが現場のエンジニアの腕の見せ所であると同時に、夜中やることが多いので、これでやってもどうせ・・・結局評価されない。とか愚痴も吐きつつ、ポチポチと直す。そんな世界ですよね。

大変なんだね。

それらを上手に解決できないか。makeとシェルを組み合わせると簡単に出来ちゃう。そういうお話です。
makefileの簡単なおさらいですが、makeはおまじないみたいな処方があって、おまじないをどう覚えるか。
基本はこんな感じで・・・

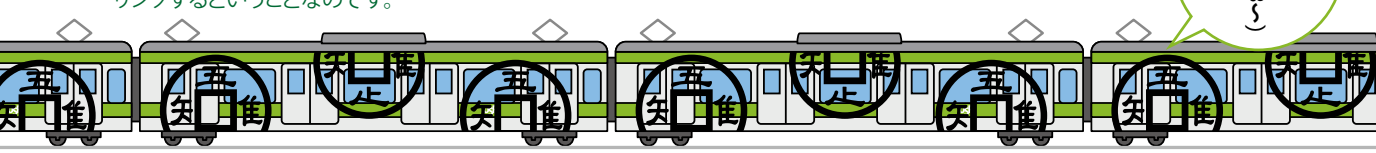
ターゲット：ソース

プログラム

ターゲットよりソースが新しければ、プログラムを動かす

ターゲットとソースをまず並べて、その下に実行のプログラムの名前を書く。単純な仕組で、ターゲットよりもソースのほうが新しいということであれば、このプログラムを動かす。こういった設定ファイルを一個書いておけば、あとはmakeとコマンドを打つだけでいいのです。ターゲットとソースが古いか新しいかを比較しているのです。動きを見て動かす、という仕組みです。これがビルドの世界では、モジュールがアップデートされたときに、リ・コンパイルしてリンクするということなのです。

（会場：ほぉ～）

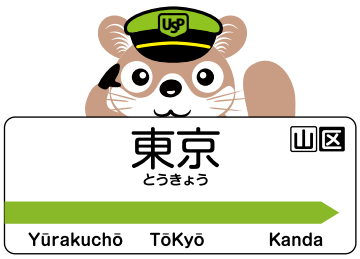


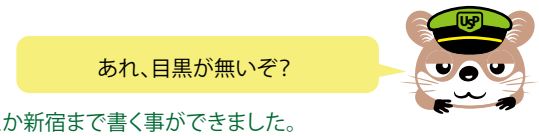
これを我々シェル狂信集団がですね、makeを見たらどう見えるか、ということなんですけど…
シェルというのはコマンドで出来ています。コマンドというのは、ファイルを加工して、何かファイルを吐き出す。必ず元があって先があるんです。データベースみたいに直接書き込むということではないんですね。"元があって先がある"というこの良さ・利点はmakeを開発の運用で使えるということなんです。
処理元ファイルと、処理後ファイルを並べてあげて、ファイル名を書いてあげれば、処理が行われていなくても、動かせるのではないか。という風に見えるのですね。世の中のスケジュール管理はですね、いろんなものがありますが、そういったものも順番があり、それで動いているので、これはもしかしてmakeで全て書けるのではないかということです。処理が動いて無ければ、シェルを動かすと。

それでは作ったmakefileを見てみましょう。電車の中で作ったので、電車のネタで行きます。
電車が山手線を走っていくというもの。東京を出ると次は有楽町、有楽町の次は新橋だ…と、電車は順番に着く訳ですから、こういう風にターゲットとソースを書いていくのです。有楽町に行きたければ山手線に乗る。そういうプログラムを動かしてあげるという単純な仕組みです。有楽町、新橋、というファイルが作られる訳です。新橋に行きたければ、山手線に乗って新橋に行けばいいのですよと。

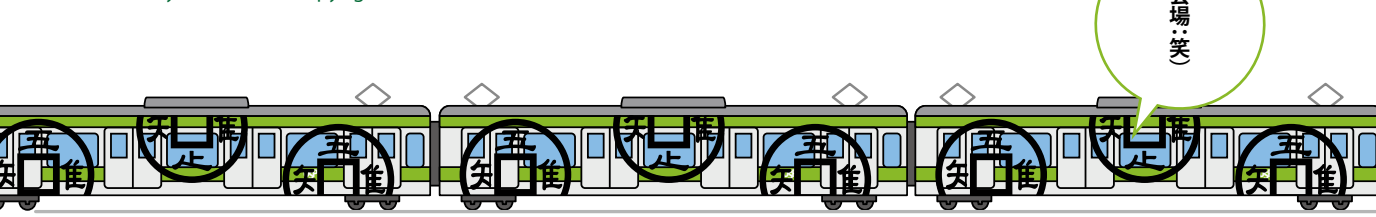
<makefile>

```
#
# Makefile のサンプル
#
# 処理を途中で止めても、リカバリポイントを
# 見つけて再開しますー
#
#
有楽町 : 東京
        ./山手線 有楽町
新橋   : 有楽町
        ./山手線 新橋
浜松町 : 新橋
        ./山手線 浜松町
田町   : 浜松町
        ./山手線 田町
品川   : 田町
        ./山手線 品川
大崎   : 品川
        ./山手線 大崎
五反田 : 大崎
        ./山手線 五反田
恵比寿 : 五反田
        ./山手線 恵比寿
渋谷   : 恵比寿
        ./山手線 渋谷
原宿   : 渋谷
        ./山手線 原宿
代々木 : 原宿
        ./山手線 代々木
新宿   : 代々木
        ./山手線 新宿

all    : 有楽町 新橋 浜松町 田町 品川 大崎 五反田 恵比寿
渋谷 原宿 代々木 新宿
clean  :
        rm -f 有楽町 新橋 浜松町 田町 品川 大崎 五反田
恵比寿 渋谷 原宿 代々木 新宿
```





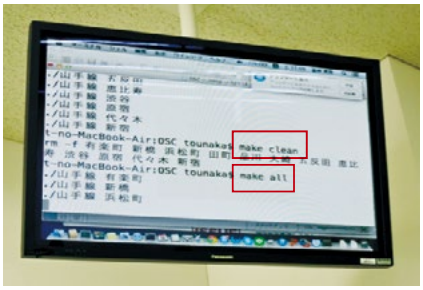
電車の中でどこまで書けるかな、と思ったらなんとか新宿まで書く事ができました。
実際に山手線のシェルスクリプトがあるのですが、これも見てみましょう。シェルスクリプトなので、一応、シェルの名前を書いた訳です。「でんしゃで、ゴー」とキーを打ったら、「電車で、剛」と変換されたのでこれでいいだろうと。つよしで行きます。
Written by シェル魔人。copyright 自由です。



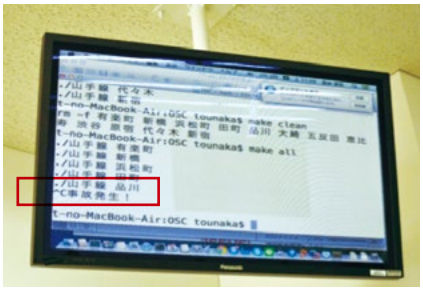
<シェルスクリプト（山手線）>

```
#!/bin/bash
#
# 電車で、剛
#
# Written by シェル魔人 / Date : 今 / Copyright 自由

trap 'echo 事故発生！' 2 # 事故処理

station=$1
sleep 1                  # 乗車時間
echo $station > $station # 着いたー
```

「Ctrl」+「C」で事故発生のtrapをいれます。
１秒電車に乗ったら、次の駅に行く。
これをmakeを使って動かしてみましょう。
東京駅を出発して、有楽町、新橋、浜松町…新宿、で
おしまいとなるわけですが、



一度ファイルを全部消します。「make clean」。
「make all」で動かす。
こんどは悪さをして、電車を止めてみます。
有楽町、新橋、浜松町、田町、品川、
と、このあたりで「Ctrl」+「C」とするわけです。
ここで、「事故発生!」と出る訳ですね。



ダイヤを回復させるためには、再度「make all」で動かします。そうすると、次は品川より先の駅から動き始めるのです。

<実行例>

```
bash-3.2$ make clean
rm -f 有楽町 新橋 浜松町 田町 品川 大崎 五反田 恵比寿
渋谷 原宿 代々木 新宿
bash-3.2$ touch 東京
bash-3.2$ make all
./山手線 有楽町
./山手線 新橋
./山手線 浜松町
./山手線 田町
./山手線 品川
^C事故発生！

bash-3.2$ make all
./山手線 大崎
./山手線 五反田
./山手線 恵比寿
./山手線 渋谷
./山手線 原宿
./山手線 代々木
./山手線 新宿
bash-3.2$
```

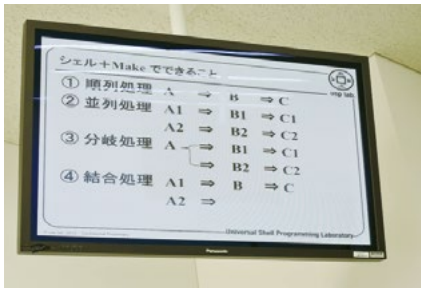
<本makefile 、make実行の仕方>

```
$ make clean      <-- ゴミファイルを消去
$ touch 東京      <-- 東京ファイルをはじめに作成する
$ make all        <-- Makefile に従って、順にすべての駅に
                      電車が走る
./山手線 有楽町
./山手線 新橋
(Ctrl+C)          <-- Ctrl + C を押下して電車を止める
事故発生！        <-- 山手線スクリプトには、割り込み定義の
                      記述が予めしてある
$ make all        <-- 運転再開（シェルスクリプトやMakefile
                      を変更する必要は無い）
./山手線 浜松町   <-- 自動的に継続ポイントを発見して処理
                      を継続する
./山手線 田町
./山手線 品川
.....
./山手線 新宿     <-- 最終目的地新宿に着いたー
$
```

※「chmod　+x」で「山手線」ファイルに実行権限を与えてね。

単純な仕組みなのですが、これが意味するのは、夜中に何かシステムトラブルがあった場合、makefileを使って、インとアウトのファイルの定義をしてあげるということです。原因を除去してあげればいいのですね。

ほんまかいな、って思われる方もいらっしゃると思います。しかし最近、金融の仕組みもこれで動き始めているんで、結構現実的なお話です。
makefileを気合い入れて書き始めると、順番にABCと動いているものも書くことができます。にんじん、じゃがいもの料理のように、並列のものが出来ますね。

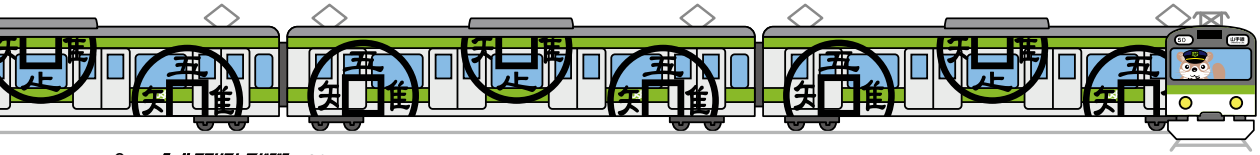


３番目の分岐、４番目結合であったりなどもmakeで書けるわけです。良く知った方は「それってグラフ理論ではないですか?」と鋭い指摘もされるかと思います。
そういったもの、単点と結びつきというものをmakeで書いてあげれば、システム全体を記述できると自然に思うわけです。

単純なものは簡単にできます。さらなる課題ということなのですが、どこまでmakeが運用で耐えられるのかですが、下記の3つの場合は可能だと私は踏んでいます。
例えば一つ目はタイムリミット。Aという処理は３時まで。というルールを作った場合、３時を過ぎたらダメということにしてください。という設定をどうしたらいいか。
二つ目は並走禁止。A1のあとにA2が走る、B1のあとにB2が走る。という処理ですが、A1とB2、この２つは同時に走ってはいけない。わかりやすい例で言うと、料理で人参とじゃがいもを切りたいのだけれども包丁が１本しかない場合。制約条件がある。こういうことは良くあるので必ず取り入れたい。
あと三つ目、これは空論に近いのですが、定時刻開始。

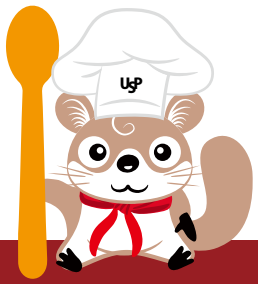
Aという処理は2時に開始しなさいという設定があれば、定時刻に処理を行うことができます。これが出来ればいろんな仕組みに使うことができます。
条件を入れてやるとmakefileを書き直す。ちょっとしたラッパーを作る。こういったことが出来るのではないかと。
これが出来るようになると、例えば大災害が起きて都市機能がマヒしてしまった。いろんなところで火事が起きて、いろんなところでパイプが寸断されている。さぁあなたなら、どうやってリカバーしますか?限られた人、限られたエネルギーの中でどういう順番で対処すべきか、考慮すべきことはたくさんあります。こういった大規模なものでも、make一発で解決できるのではないか、と夢見ているんです。

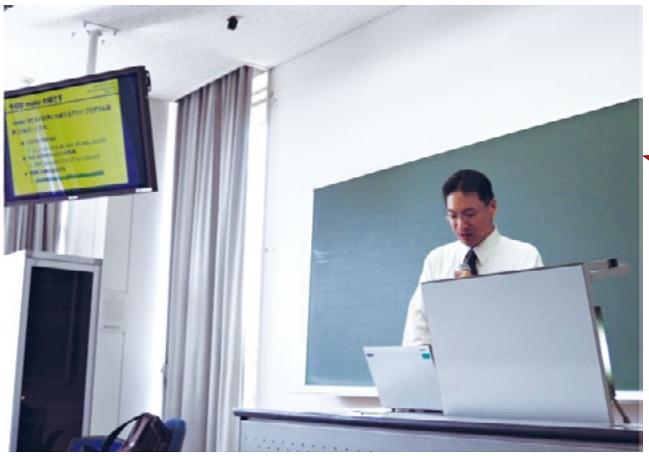
でも僕はもういい年なんで力尽きていまして、これをやってくれる人が居ないか探しています。さいとうさんとも「これは宗教なんではないか?」という話になったんです。これは入信していただくのが一番だと。そういうことで、４月からうちの会社でフェロー制度を作って、レポートを書いていただいて、優秀な方にはフェローになっていただく。ということを考えています。こちらではお題を用意します。是非興味のある方はご連絡ください。今日は以上でございます。



日本GNU AWKユーザー会の斉藤の方から、補足的なところや、追加的なところを含めてお話させていただきます。

よ!待ってました!





簡単に自己紹介させていただくと、最近はAWKよりもbashをよく使っていて、bashでプロキシサーバーをたてています。結構良くて、個人的にはお気入りですが家の中では評判悪いです。
日本GNU AWKユーザー会というものがありまして、こちらの会長をやっています。Webページは当然AWKでできています。それからUSP友の会で幹事をやらせていただいております。その友の会主催のシェル芸勉強会というものがありまして、私はTA的なところをやらせていただいています。

OSC2015 Tokyo/Spring 発表資料：OSC・シェルのプロが語る『make を使ったデータ処理。』【make 教】
http://www.slideshare.net/HirofumiSaito/osc2015-tokyo-spring20150228

makeに絡む話としては、Sphinxという、Pythonで書かれたドキュメントツールがありまして、最近はこちらを結構使っています。Sphinxって、ドキュメントツールなのに、なぜかmakeを使うという…ちょっと変わったツールで、CUIを使う人にはオススメです。
さて、今日はmakeの話ですよということで、先ほども出ましたが、makeはビルド以外にも使えるグラフ、プログラム言語…プログラム言語というよりも、グラフを作るソルバーのような意味合いかなあと私は思ってます。
皆さんご存知のようにソフトウェアのビルドは、

```
$ ./configure && make && make install
```

…たぶんこれもう何度も毎日のようにやっている方もいらっしゃると思いますが、makeはこういう時によく使われます。
また、ドキュメントツールみたいなもので、例えばTeXなどを、使った事がある方はわかると思うんですが、作品を作るときに、「二回ビルドしなきゃいけない」とか、「このTeXって何回実行しなきゃいけないんだっけ?」という風に、わからなくなりがちなのですが、makeファイルを準備しておくと、「ああこれを実行するだけでものができちゃうんだ」というのがわかります。

今回は、どちらかというと普通に手順を記述する、というところに着目してお話しします。いわゆるバッチ処理だとか、グルー言語としての利用を考えています。…この場合グルーは"糊(のり)"を指しますが、「グルー言語」と言えば昨日、われらが友の会、会長の上田さんが開発中の「Glue Lang」という言語について、面白いセミナーがありましたが、こちらのことではありません。

グルーってそういう意味だったんだ。

では、私のほうでまとめたmakeの特徴をご紹介します。

①順をまとめられる。
いわゆるバッチとかグルーとかそういう意味合いのところですね。
②エラーで止まる。
③再開できる。
これは非常に便利なもので、Cのビルドをする時も、エラーが出て途中で止まっても、Cのソースを直してまた途中から実行することができます。
④復元できる。
…と書いてますが、自分でやらないといけないんですけども、makeには、必ずmake cleanというものが入っていて、これを実行すると、ビルド前の状態に復帰できる、というのが特徴です。

ではもう少し詳しくお話していきます。makeは手順を記述するだけで動作します。私は料理本に近いと考えています。まず作りたい物があって、それを作る材料を用意します。そして、用意した材料で、作りたい物はどうやればできるか、というレシピを書いていきます。

それからお約束というのがいくつかあって、作りたい物の後には、コロン(:)を置くだとか、作る材料はタブか、スペースで区切ります。一点、Windowsで編集するとよくミスする例として、レシピの前に必ずタブを入れないといけません。大半のmakeでは、ここがスペースだと動きません。

<Makefileの記述法>

```
作りたいもの: 作る材料1 作る材料2 …
        レシピ1
        レシピ2
```

では具体的な例をご紹介していきます。まず、「おまかせ」の例です。例えば「allという物を作る為には、fizzとbuzzというものを作ると、すべて完了ですよ」とする場合。言い換えると、「allというものはfizzとbuzzで出来てますよ」ということです。

<おまかせのMakefile>

```
fizz:
        sleep 1

buzz:
        sleep 1

all: fizz buzz
```

<おまかせの実行>

```
$ make all
```

この例では、「fizzは1秒間待ちなさい、buzzは1秒間待ちなさい。」と書いておきました。makeのこういったサンプルでは、手間なので「sleep 1」をよく使います(笑)。
この「おまかせ」では、順序は決定しておらず、「とりあえず、fizzとbuzzが出来れば良いよ」という場合に、make allという記述で実行できます。
逆に、「順序を守ってください」という記述もできます。

<逐次処理のMakefile>

```
fizz:
        sleep 1

buzz: fizz
        sleep 1

all: buzz
```

<逐次処理の実行>

```
$ make all
```

例えばこの場合だと、「allを作るには、buzzを作ります。ただし、buzzをつくるにはfizzが要ります。fizzはこうですよ」と、こういう記述です。これを実行すると、fizzが始めでbuzzが次、それで終了、というように、必ず順序が決定されるというわけです。
それからBSDにもあるかもしれませんが、GNUのmakeには「並列でやる」というものがあります。

<逐次処理のMakefile>

```
fizz:
        sleep 1

buzz:
        sleep 1

all: fizz buzz
```

<並列処理の実行>

```
$ make -j
```

これは先ほどの、「おまかせ」の例ですが、まず、「allを作るにはfizzとbuzzをやってください」というのがあり、それに対して並列実行「job」の「j」ですね、「-j2」を加えると、2並列で動きます。これを利用することで分岐処理のようなことが出来ます。
make以外にも並列処理というのは意外とUNIXで準備されています。Perl、PHP、Ruby、Python、C…等で、いわゆるforkを使ってプロセスforkをかけるという方法も、もちろんありますが実は始めから並列処理の機能は大体用意されています。マルチタスクOSなので、あって当然なのですが。
例えばmakeだと、「-j」オプションですね。
またmakeファイルには、「-l」というオプションがあり、ロードアベレージを考慮し「あるロードアベレージよりも、超えている時に並列処理は、させちゃダメだよ。」という動作をします。

それから、シェル芸勉強会でよく出てくるxargsですね。xargsは「-P」というオプションで、Parallelの数を指定できます。xargsを使われる方もいらっしゃいますし、GNU parallelを使われる方も結構いらっしゃいます。それから「xinetdスーパーサーバー」というものがありまして、コレを使うと、Socket側の並列処理がとっても簡単に出来ます。先ほどbashでプロキシサーバーを立てているとお話しましたが、並列化の部分はすべてxinetdを使っています。

makeにも不得意なところというのがありまして今までの話だけだと、「そこまで出来るんだったらシェルスクリプトの代わりになるんじゃないの」と思えるのですが、そうは問屋がおろさないんですね。

①標準入力ができない。

簡単に言うと、1行が1シェルに渡ってしまうので、別プロセスで動作してしまいます。なので、標準入力というものを扱えません。要はパイプでmakeにつないでいく事が(用途によっては出来るんですけども)基本的にできません。

②途中で変数代入というのはできない。

これは何かというと、makeは決定論で動くので、始めに全てが決定されます。冒頭でちょこっとお話ししましたが、configureというものがあります。始めに全ての材料を用意してあげるというのがconfigureで、それを順序通りに組み立てていくのがmakeです。

③ヒアドキュメントが書きづらい。

最後にバックスラッシュが必要で、書くのが汚くなる、というのがありがちなパターンです。

④PIPESTATUSの考慮。

これはbashやzshで、使われている方もいると思うんですが、シェル芸のようにパイプでつないでいったときに、makeを使うと、一番最後の戻り値しか見ないため、例えば始めでエラーを出した時、そこで止まってほしいのに、止まってくれません。

上記のようなことが不得意なわけですが、バッドノウハウみたいなものがありまして、上記③のヒアドキュメントに関しては、「GNU make のMakefileにシェルスクリプトを自然に書くたった一つの方法」(http://d.hatena.ne.jp/holidays-l/20110823/)というホームページがあります。実際の例を書きますと、dangerという…いわゆる危険シェル芸ですね。危険シェル芸を書くときに、後ろにバックスラッシュとか、改行の何らかは要らず、そのまま書きます。そして少し手間ですが、exportして、$$を付けてあげると、実はヒアドキュメントっぽいことが可能になります。

<ヒアドキュメントっぽい記法>

```
define danger
: ( ) { : | : } ; :
: ( ) { : | : } ; :
endef

export danger

foo:
	echo "$${danger}"
```

それから④のPIPESTATUSですね。bashなどにはpipefailというのがあって、パイプの始めであれ、最後であれ、パイプの中でエラーを起こしたら、エラーで止める、というものがあります。

<pipefailを使う>

```
SHELL = /bin/bash

foo:
   set -o pipefail; exit 1 | echo " : ( ) { : | : } ; : "
```

そのままでは使えませんが例えば、「makeのルールでパイプを使った時のふるまいや、記述について教えて」という希望があったとします。これは私がOSC第一回で出展したときに、一緒にやっていた上鍵さんという方のページに書いてあったのですが、1行が1シェルなので、その中にset -o pipefailと記述して、exit…と、上記の様に書いてあげると、うまくいくという話があります。

make の仕組みについてもっと知りたい方へ。実は USP 研究所さんが出している、プログラミング言語 AWK という本の中に、AWK の作者、エイホ・ワインバーガー・カーニハンという三賢者による簡単な make の実装法が載ってます。

◆まとめ

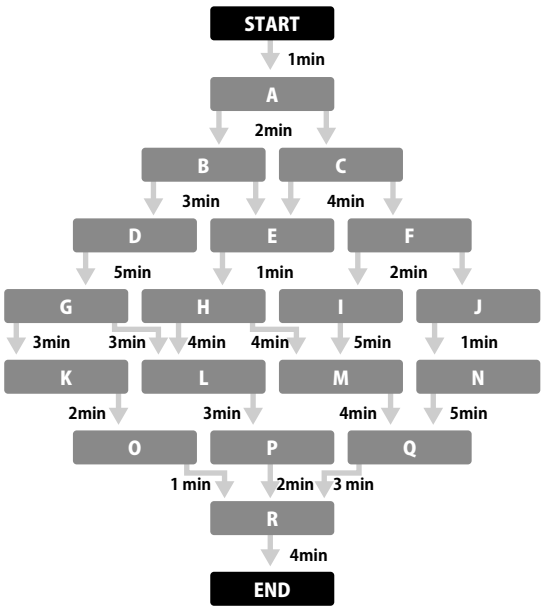
ココまでのお話をまとめますと、以下のような要件だったら make を使った方が良いですね。

①手順をまとめいたい！	→	それは全力でmake化するべきです。
②皆に使ってもらいたい！	→	それは体を張ってmake化してください。
③逐次処理や並列処理をしたい！	→	それは命をかけてmake化するべきです。

ということです。

で、ちょっとオマケで分岐・結合の話で、こんなのを。

以下を行うのに何分かかる？



適当な例ですけども、スタートからエンドがあって、ABCD と色んな手順があって、→（矢印）のところに、何分間その仕事にかかるかというのが書いてあります。この図をぱっと見ただけでは、「全体で何分かかりますね」とか「これを効率よくやるためには、何人必要ですよね」というのはわかりません。こういうものを make は解く事が出来ます。

makefile は以下のとおりになります。

< その 1>

```
START:
                @echo $@
                @sleep 1
A: START
                @echo $@
                @sleep 2
B: A
                @echo $@
                @sleep 3
C: A
                @echo $@
                @sleep 4
D: B
                @echo $@
                @sleep 5
E: B C
                @echo $@
                @sleep 1
```

< その 2>

```
F: C
                @echo $@
                @sleep 2
G: D
                @echo $@
                @sleep 3
H: E
                @echo $@
                @sleep 4
I: F
                @echo $@
                @sleep 5
J: F
                @echo $@
                @sleep 1
K: G
                @echo $@
                @sleep 2
```

< その 3>

```
L: G H
                @echo $@
                @sleep 3
M: H I
                @echo $@
                @sleep 4
N: J
                @echo $@
                @sleep 5
O: K
                @echo $@
                @sleep 1
P: L
                @echo $@
                @sleep 2
Q: M N
                @echo $@
                @sleep 3
```

< その 4>

```
R: O P Q
                @echo $@
                @sleep 4
END: R
                @echo $@
```

< 逐次実行 >

```
$ time make END
START
A
B
D
<snip>
Q
R
END

real      0m55.066s
user      0m0.001s
sys       0m0.020s
```

<2 並列 >

```
$ time make -j2 END
START
A
B
D
<snip>
Q
R
END

real      0m32.032s
user      0m0.004s
sys       0m0.020s
```

<3 並列 >

```
$ time make -j3 END
START
A
B
D
<snip>
Q
R
END

real      0m26.033s
user      0m0.006s
sys       0m0.023s
```

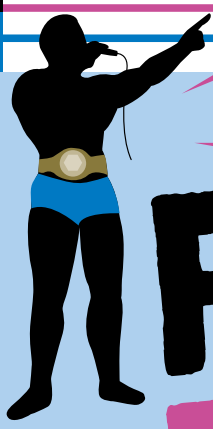
<4 並列 >

```
$ time make -j4 END
START
A
B
D
<snip>
Q
R
END

real      0m25.031s
user      0m0.003s
sys       0m0.025s
```

逐次実行すると 55 秒かかります。それに対して 2 並列でやると 32 秒、3 並列だったら 26 秒、4 並列だったら 25 秒…という風に、並列でどんどん短縮するということができます。
これをうまく利用できないかということを考え、例えば
業務を何人でやると効率がいいのか
誰がボトルネックになっているのか
などがすぐにわかるように役立てることができないでしょうか。最近流行のニューラルネットワーク、これはある意味グラフ理論に近い物なので、そういうところで make をうまい具合につかえないかなと考えています。

それから実際には違うのですが『探索問題』ですね。
make ってソルバーの一種だと考えているので、そういう問題解く為のものとして使えないでしょうか。
あとは友人関係をわかりやすく出来ないか。というのも最近、Facebook が人との関係を絵のグラフにして、誰と誰が友達ということや、誰まで行くのにどういう関係を作るのが一番近いか等、全世界のデータを勝手に集めています。こういったことも例えば私が「この人知りたいな」と思ったときに、知り合いの知り合いの知り合いを、どう辿っていけばいいんだろうか、ということに make を利用できないだろうかと考えています。

SNSかぁ、身近なことだね。いろいろ利用法はありそう!まずはぼくも簡単なmakefileとスクリプト、書いてみよう!今日はどうもありがとうございました!



法林浩之の FIGHTING TALKS

第12戦 アイアンチキン

written by 法林浩之（日本UNIXユーザ会）

日本のIT業界では日々数多くの勉強会やイベントが行われているが、その舞台裏が語られることは少ない。本稿は、自らの半生をITイベント運営に捧げた、IT業界の「闘う男」が、イベントやトークについて語る連載である。

前回は、エンジニアサポートCROSS 2015[*1]にて開催し好評を博した「プログラム言語対抗綱引き」[*2]の舞台裏を紹介した。今回はその続編として、この企画が成功した理由や、このような熱狂的な試合を生み出すコツを、他のイベントでの経験も踏まえて考察する。

企画の目新しさ

このプロジェクトを振り返ると、まずなんと言っても企画の斬新さが目を引く。プログラム言語間の争いという長年の課題に決着をつける方法を、綱引きという誰が見てもわかる手段に求めるのである。このような目新しい企画を創造するには、2つの力が必要であると筆者は考えている。1つは幅広い見識、もう1つは柔軟な発想である。

幅広い見識というのは、もちろんITの知識は必要だが、むしろ他分野の知識をどれだけ持っているかが、特にイベントでは役に立つと思う。筆者の場合はプロレスや音楽などがそれにあたる。読者の皆さんも各自の趣味を生かしてほしい。

そして、柔軟な発想は、蓄えた知識をいかに自由に組み合わせるか、ということに他ならない。今回であればプログラム言語と綱引きの組み合わせである。発想の柔軟性を磨くコツはよくわからないが、筆者が心がけているのは、日頃から何を見ても、それがイベントでどう使えるかを考える習慣を持つことである。その積み重ねがイザというときに良い発想をもたらすのではないかと思う。

企画を実装する力

どんなに斬新な企画であっても、それを現実のものにするには、実装する力が必要である。

実装するにあたり必要なことの1つは、要素となる技術や知識を持っていることである。前の項で述べたいろいろ見ておけという話だが、いつ何が役立つかはわからないし、必要になったときに急に勉強しても間に合わないので、日頃からいろいろ吸収しておくことが肝心である。

例えば綱引き企画においては、トーナメントの実装方法を知っていたことが役に立った。プロレスでは1回の興行の中でトーナメントを行い優勝者を決める事例がままある。何チーム出場するとこれぐらい時間がかかるとか、優勝するには何試合もする必要があり体力を消耗するので試合間に休憩をはさむ、などのノウハウを拝借した。また、当然ながらプログラム言語に関する知識も必要である。これは主にLLイベント[*3]で各言語とプログラマーを観察してきた経験が生かされた。対戦カードも好評だったが、あれは各言語の関係性を把握していたからこそ作れたものである。

しかし、実装においては知識も重要だが、もっと大切なのは企画を実現させようという情熱だと思う。その情熱の下に考える力が生まれ、なんとかしようという動きになる。インターネットやオープンソースの世界でも、理屈をこねるよりもとにかく動かそうと頑張って実装した人が高く評価される。それと同じである。

見せ方と演出で光らせる

同じ企画でも見せ方ひとつで印象は大きく変わるので、筆者は演出には気を使うようにしているのだが、今回はそちらの面でも満足のいくものを作ることができた。綱引きの用具や審判を専門の業者に依頼したこと (これはケガの防止など安全確保の意味合いもあるが)、見た目の面白さやチームの一体感を出すために各言語の T シャツを作ったこと、因縁丸出しの対戦カードやいかにもエンジニアっぽい作画のトーナメント表、手作り感あふれるボードを抱えての実況と解説などである。こういうのはとにかく、やれることは徹底してやるのがよい。

手製のボードを抱えて解説する筆者と実況の古賀さん

また、このような徹底した演出ができたことの背景には、綱引きスタッフのチームワークの良さもあったと思う。特に主担当者の古賀及子さんが、他のスタッフの特技をうまく引き出して仕事を依頼したおかげで、各自が持ち味を生かして関わることができたのが成功の要因と考えている。

実装力が試される試合再び

とりとめもなく書いてきたが、まとめると、日頃からさまざまな物事に触れ、それがイベントにどう役立つかを考え、そこから得た発想を情熱を持って実装する、そしておもしろく見せる工夫をする、ということになる。

そして、今年の夏、また実装力を試される試合が決まった。それは今年の LL イベントで、9月5日 (土) に新木場 1st RING[*4] で開催する。RING という名の通り、場内にプロレス用のリングが常設されており、日頃は主にプロレスの興行が行われる会場だが、そこを借りてカンファレンスをやろうというのである。それ自体、突拍子もないことだと自分でも思うが、実はこの会場を借りるのは初めてではなく、2006年の LL イベント・LL Ring[*5] で使用して以来、9年ぶりになる。


新木場 1stRING のリング上で記念撮影する LL イベント実行委員会の面々

新木場 1stRING を借りることにした経緯はまた別の機会に書こうと思うが、この会場にはプロレスの興行で使う設備がそろっているので、それをそのまま演出に使える。実際、LL Ring のときも、発表者はテーマ曲に乗って花道から入場、机を口の字型に配置してのパネルディスカッション、ライトニングトークの開始と終了はゴングを鳴らして通知、などの演出を行った。もちろんプロジェクターやスクリーンも用意されているので、IT 系の発表にも問題なく対応できる。

前回から9年が経ち、場内の備品は更新されているが基本設計は変わっていない。よって前回やったようなことは今回もできそうだが、せっかくもう一度やるからには前回を超える試合を見せたいと、今から演出のアイデアを練る今日この頃である。皆さんの予定表の9月5日の所に書いておいていただければ幸いである。

FIGHTING TALKS, to be continued...

*1：http://2015.cross-party.com/
*2：http://portal.nifty.com/kiji/150203192687_1.htm
*3：http://ll.jus.or.jp/
*4：http://www.1st-ring.com/
*5：https://ll.jus.or.jp/2006/

ISBN 978-4-904807-19-4
C3455 ¥500

USP 研究所
定価（本体 500円+税）