# ＩＴ特別講座
# 実践シェル・スクリプト

## ～システム管理で手を抜く秘訣～

---

Ver 1.3

LA-Linux 専任講師 矢越昭仁

2011/5/28

シェルスはコマンド入力だけでなく、プログラミング言語としの機能も持ちます。このコースではシステム管理として想定される作用を省力化するスクリプトの例と、それを通して LIPC で登場するコマンド群の概要を学ぶことができます。

目次

## 1.はじめに

システム管理のコマンド操作は、慎重に慎重を期して行う必要があります。たった１文字のスペルミスで必要なファイルの削除や、システム停止といった事故に繋がる可能性があるからです。また新人受入れ時の大量ユーザ登録などの繰り返し作業や、日々のバックアップもうっかり事故が発生しやすいので注意が必要です。
この様な誤操作回避、繰り返し実行、自動化に有効な方法としてシェル・スクリプトによるシステム管理の省力化・自動化があります。
このコースでは、具体的な運用シナリオを想定し実際にシェル・スクリプトを作成することで、実践的なノウハウを身につけることを目指しています。

### 1.1 シェル・スクリプトとは

普段、キーボードからの入力された文字列を解析しプログラムを実行しているプログラムがシェルです。Linux では bash(Bone Again Shell)とよばれるシェルが標準で採用されています。
このシェルは、直接キーボードから入力されたコマンドを実行するだけでなく、条件判断、繰り返しといった機能を用いることで作業を自動化することができます。
また一連のコマンドをファイルに書き溜めて、一気に実行することができ、これをシェル・スクリプト（シェルに与える台本）と呼んでいます。

### 1.2 シェルのメカニズム

シェルを対話形式で使っていると、コマンドを途中までタイプし、タブキーを使ってコマンド名やファイル名の不足部分を補ってくれる「コマンド行補完機能」はよく使っていると思います。シェルはどうやって、この補完を行っているのでしょう？

#### コマンドの補完

まずコマンド名の補完では、コマンドが収められているディレクトリから指定された文字で始まる実行可能なファイルを検索しています。コマンドの格納場所は環境変数 PATH に定義されています。

```
$ echo $PATH
/bin:/usr/bin
```

同じ名称のファイルがあれば、PATH の登場順で早いもの（左側から）が採用されます。この設定は ~/.bashrc に記述することで変更可能です。また、今いるディレクトリ(.)を加える場合は誤動作防止の観点から、最後に付けることを強くお勧めします。
自分用のオリジナル・スクリプトを格納する場所としては、~/bin などがお勧めです。

#### ファイル名の補完

ファイル名はコマンドの引数（空白で区切った２つ目以降の文字列）として入力された文字列について行われます。

```
$  ls  /bin/l <TAB><TAB>
$  ls  /bin/?s <TAB><TAB>
$  ls  /etc/*/*.conf<TAB><TAB>
$  ls  /etc/rc[0-9].d<TAB><TAB>
```

過去に実行したコマンドの実行

さて。長いコマンドの場合は、再度入力するのは面倒ですし、矢印キーをつかって辿っていくのも大変です。そんな時は！が有効です。

- ✓ !!　　直前に実行したコマンドの再実行
- ✓ !str　　str で始まるコマンドの再実行
- ✓ !str:p　　str で始まるコマンドの表示
- ✓ history　　過去に実行したコマンドの一覧表示
- ✓ !nn　　コマンドの一覧の履歴番号 nn を実行

繰り返し実行（無限）

特定のファイルの大きさの変化を監視し続けるには、どうしたらよいでしょうか。!! を連打してもよいですが、while コマンドを使って自動化することが可能です。

```
$ while true
do
      ls –l /var/spool/mail/student
      sleep 5
done
```

終了するには ^C を入力します。sleep は指定した秒数、一時停止します。sleep を指定しないと、極端な負荷をシステム与えるので ^ C さえ受付けてない事態になる場合があります。

繰り返し実行（演算型）

先の例は、無限に繰り返しましたが 10 回だけ実行したいという場合は、for 文を用います。

```
$ for ((n=0;n<10;n++))
do
     ls –l /var/spool/mail/student
     sleep 5
done
```

これらのコマンドは、非常に長く一回で入力できない事もあるでしょう。そんな時は fc コマンドを実行します。fc と入力すると直前に実行したコマンドを vi エディタで開くことが出来ます。fc str として、直前に実行したコマンド str で始まるコマンドを編集します。保存すると再度実行してくれます。(:q! で中止終了すると、実行されません)
なお fc で起動するエディタは環境変数 EDITOR により変更することが可能です。

**繰り返し実行（列挙型）**

for 文のもう一つの構文として要素を並べ先頭から順に変数に代入する方法があります。たとえば「新人３０名の名簿を元に、ユーザを作成する」といった事も手軽に行うことが出来ます。

想定シナリオ：

この 4 月に入社する新人の名簿が用意されており、その内容は以下の通り。

```
漢字氏名,1st name, last name, ユーザ名, 所属, 電話番号, 社員番号
```

ファイルはカンマで区切られた CSV とよばれる形式です。このファイル member.txt を元に、まずユーザを作ることを考えてみます。

カンマで区切った特定の要素を取り出すには cut コマンドを用います、この例では 4 つ目の要素を取り出すので、次のように行います。

```
$  cut  -d,  -f4    member.txt
```

この結果を使って、useradd コマンドを実行するには、for を応用して次のようにします。

```
#  for user    in  `cat  -d,  -f4   member.txt`
do
    useradd   $user
done
```

いきなり実行するよりは、useradd [1]コマンドが実際に有効な値になっているかどうかを確認するために、最初は echo を使って確認し、fc を使って上記の作業を行う事がよいでしょう。

```
     echo  “useradd   $user”
```

引用符の違い：

- ✓ シングル・クォーテーション(‘)　　Shift + [7] :
  額面通り、そのまま表示
- ✓ ダブル・クォーテーション(“)　　Shift + [2]  :
  $ があれば変数の値に置き換える
- ✓ バック・クォーテーション(`)　　Shift + [@] :
  コマンドとして実行した結果と置き換える

[1] CentOS ではまとめてユーザ登録をする newusers(8)コマンドが用意されています。

## 2.スクリプトの作成事例

それでは本題として、以下のシナリオ（課題）を解決するスクリプトを少しずつ改良しながら開発していきます。

シナリオ：

株式会社ＡＢＣはネット販売を行う優良企業で、貴方はそのシステム管理者です。
最近何人かのユーザ（社員）から、ファイルサーバが妙に遅いとの苦情をうけていますが、確認するとなぜか普通に動作していて、なかなかその事象にお目にかかることができません。そこであなたは、まず ping を使って 3 台あるファイルサーバを数日監視することを思いつきました。

### 2.1 ping の自動化

ping をずっと実行していれば何かネットワークの変化に気づくかもしれません、しかし端末から ping XXXX としても、永遠とログが表示されるばかりで能がありません。
そこで crontab を使って、毎分 3 回づつ ping を実行しておく事を思いつきました。

```
$  crontab  -e
  * * * * *  ping  -c 3  www.yahoo.co.jp  >>  /tmp/ping.log
```

1 時間ほどして、crontab が掃出したファイルを見てみると、確かに実行しているのですが、いつ実行した結果なのかわからない事に気づきます。しかも監視に必要なのは最後の行にある統計情報(RRT: Round Trip Time 最小/平均/最大/平均差)だけで、あとは不要です。

```
PING www.ya.gl.yahoo.co.jp (203.216.235.154) 56(84) bytes of data.
64 bytes from f7.top.vip.tnz.yahoo.co.jp (203.216.235.154): icmp_seq=1 ttl=51 time=6.19 ms
64 bytes from f7.top.vip.tnz.yahoo.co.jp (203.216.235.154): icmp_seq=2 ttl=51 time=5.80 ms
64 bytes from f7.top.vip.tnz.yahoo.co.jp (203.216.235.154): icmp_seq=3 ttl=51 time=5.23 ms

--- www.ya.gl.yahoo.co.jp ping statistics ---
5 packets transmitted, 3 received, 0% packet loss, time 3999ms
rtt min/avg/max/mdev = 5.131/5.660/6.199/0.423 ms
PING www.ya.gl.yahoo.co.jp (124.83.147.204) 56(84) bytes of data.
64 bytes from f4.top.vip.ogk.yahoo.co.jp (124.83.147.204): icmp_seq=1 ttl=53 time=18.5 ms
64 bytes from f4.top.vip.ogk.yahoo.co.jp (124.83.147.204): icmp_seq=2 ttl=53 time=18.4 ms
64 bytes from f4.top.vip.ogk.yahoo.co.jp (124.83.147.204): icmp_seq=3 ttl=53 time=18.1 ms

--- www.ya.gl.yahoo.co.jp ping statistics ---
5 packets transmitted, 3 received, 0% packet loss, time 4002ms
rtt min/avg/max/mdev = 18.054/18.384/18.781/0.297 ms
```

補足）crontab は、結果が画面に出るような場合（この例ではファイルにリダイレクションしなかった場合）実行したユーザに結果をメールで報告します。

### 2.2 pinglog の作成

そこで、必要な情報だけを表示する pinglog スクリプトを作る事にします。必要な情報は実行した日時と ping の最後の行だけです。早速 vi で pinglog ファイルを作成してみましょう。

pinglog ver 1.0

```
#! /bin/bash
RESULT=`ping -c 3 www.yahoo.co.jp | grep ^rtt`
echo "`date`      $RESULT"
```

実行してみると、表示はシンプルになりました。

```
$   ./pinglog
2009 年  10 月  27 日  火曜日  15:38:45 JST      rtt min/avg/max/mdev = 4.600/5.046/5.405/0.262 ms
```

しかし、監視するサーバは複数台あるので、このような決め打ちでは応用がききません。また日付の表示や rrt の内容が冗長な気がします。もうひと工夫して、引数を指定できるように改造しましょう。

pinglog ver 1.1

```
#! /bin/bash
LANG=C
HOST=$1
RESULT=`ping -c 3 $HOST | grep ^rtt | cut -d/ -f5`
DATE=`date +"%Y/%m/%d(%a) %T"`
echo "$DATE $HOST $RESULT"
```

$1 は引数の 1 番目を示し、同様に$2、$3 と引数の分だけ用意された特別な変数です。

cut は指定した区切り文字で要素をわけ、必要な要素を取り出します。今回はスラッシュ(/)を区切りとして、５つ目の要素（平均値）を取り出しています。

また date は書式を工夫して 2009/10/27(Tue)といった形式で表示しています。表記を英語にするために LANG 変数を使っています。日本語表示はシステムによって文字化けを起こすことがあること、Xが立ち上がっていない状況ではそもそも日本語が表示できないなど制限があるため、今回は「C」（C言語＝コンピュータ言語）を使用しています。

```
$ ./pinglog   www.yahoo.co.jp
2009/10/27(Tue) 16:25:52 www.yahoo.co.jp 4.691
```

### 2.2 pinglog の拡張 1(オプション処理)

pinglog はＩＴチーム内で汎用的に使えそうですが、ping の結果 3 回だけでなく、任意に指定したいと思います（どうも実験したサイトへの RTT にバラつきが大きそうだし…)。また使い方を間違えると、ping のエラーが出てしまい他のメンバーにお薦めするするは完成度が低すぎます。そこで Linux のコマンド同様、オプションを使えるように改造します。-c で繰り返す回数を指定し、-h で簡単な使い方を表示。また引数やオプションに誤りがあった場合にも簡単な使い方(usage)を表示します。

pinglog ver 2.0

```
#! /bin/bash
# pinglog ver 2.0 / (C) ABC co. IT-Team Yakoshi
LANG=C
opt_c=5;

function usage()
{
        cat <<EOF
$0 [-h][-c count] hostname
ping logger
   repeat pinging and display TOD and RTT
   -h    print this message
   -c    pinging count
EOF
        exit 0
}

while getopts hc: opt
do
        case $opt in
        c) opt_c=$OPTARG ;;
        h) usage ;;
        *) exit 0 ;;
        esac
done
let OPTIND--
shift $OPTIND
[ $# = 0 ] && usage

HOST=$1
DATE=`date +"%Y/%m/%d(%a) %T"`
RESULT=`ping -c $opt_c $HOST | grep ^rtt | cut -d/ -f5`
echo "$DATE $HOST $RESULT"
```

オプション処理を考えると、非常に厄介です。たとえば pinglog –c 10 www.yahoo.co.jp と入力した場合、そのままでは $1 が –c 、$2 が 10、$3 がホスト名となります。オプションは指定しない場合もあるので、その時は従来のように $1 が ホスト名になります。このように「あったり、なかったり」する引数を扱うには getopts 組込コマンドが便利です。getops には引数が 2 つあり、オプション文字の羅列と、処理を行う際に用いる変数です。

| getopts　オプション文字列　変数 |
|---|

オプション文字列はマイナスを伴わないオプションの一文字(c や h)の羅列で順序は問いません。またオプションに値（パラメータ）が付く場合には、そのオプションの直後にコロンを加えます。getopts はシェル引数全てを調べ、オプション文字(-c や -h)であれば、変

数にその値を代入し、オプションのパラメータを $OPTARG に代入します。

```
while  getopts  hc:  opt
```

は、オプションが h,c であり、-c では更にパラメータを伴う事を表しています。オプションは変数 opt に代入されます。while ループ内で、case～esac を用いて各オプションの処理を行っています。

case は変数の値によって、複数の処理を行います。記述した順に上から判定し、値が合致すればその行を実行します。処理の終わりは２つのセミコロン(;;)で、複数行に渡る処理を記述することも出来ます。

1. オプションが c だった場合は、変数 opt_c にパラメータを渡します。たとえば 10 といった値が代入されます。
   もし c オプションが指定されていない場合は、初期値として 5 を用いるよう、スクリプトの先頭部分で予め代入しています。
2. オプション h の場合は、予め定義した関数 usage を呼び出します。
3. 上記どれでもない場合は、エラーとして終了します。
   エラーメッセージは getopts が自動的に生成してくれます。

最後に OPTIND 変数に、オプション処理を行わなかった引数の位置が入ります。たとえば先の例では以下のようになります。

```
コマンド行：     pinglog    -c     10    www.yahoo.co.jp
シェル引数：        $0      $1     $2        $3
OPTIND：                              ▲ = 3
```

オプション関連の処理が終われば、オプション文字列およびそのパラメータは続く処理では不要となります。そこで OPTIND と shift を使って、シェル変数の位置を移動します。shift は指定された分だけシェル引数の開始位置をずらします。

今回は OPTIND を 1 つ減じ(let　OPTIND--)、その値の分だけシェル引数をずらします。

```
コマンド行：     pinglog    -c     10    www.yahoo.co.jp
シェル引数：        $0      $1     $2        $3
                                  ↓
                  shift   2   (= OPTIND – 1 )
                                  ↓
コマンド行：     pinglog    www.yahoo.co.jp
シェル引数：        $0          $1
```

ここまでオプションの処理を行い、最後にシェル引数の数( $# )を判定し、引数がなかった場合には、簡単な使い方(usage)を呼び出しています。
以下に実行例を示します。

```
$ pinglog
./pinglog [-h][-c count] hostname
ping logger
   repeat pinging and display TOD and RTT
   -h    print this message
   -c    pinging count

$ pinglog -x 10 www.yahoo.co.jp
./pinglog: illegal option -- x
$ pinglog -c 10 www.yahoo.co.jp
2009/10/28(Wed) 12:03:54 www.yahoo.co.jp 4.704
```

補足）

getopts の練習用スクリプト

getopts.sample

```
#! /bin/bash
# @(#) getopts sample

echo "pre args $# / $*"
while getopts habn: n
do
        case $n in
        a) echo " a ";;
        b) echo " b ";;
        n) echo " n with $OPTARG ";;
        h) cat <<"EOF" >&2
            getopt [-abh]    [-n PARAM] ...
EOF
                ;;
        ?) echo " unkown $n" ;;
        esac
done
let OPTIND--
shift $OPTIND
echo "post args $# / $*"
echo "OPTIND = $OPTIND"
```

実行例

```
$ getopts.sample -a -n 10 aaa bbb
pre args 5 / -a -n 10 aaa bbb
  a
  n with 10
post args 2 / aaa bbb
OPTIND = 3
```

### 2.3 pinglog の拡張 2(多重処理)

pinglog は版を重ねることで、なかなか「らしく」なって来ましたが、監視するサーバは複数台あるので、毎回その台数分を数回に分けて実行するのは面倒だと感じました。そこで引数にホスト名ではなく、ホストの一覧を記録したテキストファイルを用いる事にしました。

pinglog ver 3.0

```
#! /bin/bash
# pinglog ver 3.0 / (C) ABC co. IT-Team Yakoshi
LANG=C
opt_c=5;

function usage()
{
        cat <<EOF
$0 [-h][-c count] host-filename
ping logger
   repeat pinging and display TOD and RTT
   -h    print this message
   -c    pinging count

EOF
        exit 0
}

while getopts hc: opt
do
        case $opt in
        c) opt_c=$OPTARG ;;
        h) usage ;;
        *) exit 0;;
        esac
done
let OPTIND--
shift $OPTIND
[ $# = 0 ] && usage

for HOST in `cat $1`
do
        DATE=`date +"%Y/%m/%d(%a) %T"`
        RESULT=`ping -c $opt_c $HOST | grep ^rtt | cut -d/ -f5`
        echo "$DATE $HOST $RESULT"
done
```

忘れずに usage のメッセージを変更し、末尾に ping を実行している部分を改良して for による繰り返しにしました。

for は in に続く文字列を順番に変数に代入し繰り返します。簡単な例示をすると以下のようになります。

```
$ for n in a b c
> do
> echo $n
> done
a
b
c
```

pinglog ver 3.0 は、一見これでよさそうなのですが、すぐにダメだとわかります。

```
$ pinglog3.0 -c 10 myhost.txt
2009/10/28(Wed) 15:42:15 www.yahoo.co.jp 4.254
2009/10/28(Wed) 15:42:24 www.google.com 4.475
2009/10/28(Wed) 15:42:33 www.mixi.jp 3.710
2009/10/28(Wed) 15:42:42 www.rakuten.co.jp 3.965
```

処理が直列に繰り返されるので、pinglog は、とのほか時間がかかっています。

time コマンドを使うと、実に 30 秒以上もかかっている事がわかりました。

```
$ time pinglog3.0 -c 10 myhost.txt
2009/10/28(Wed) 15:43:34 www.yahoo.co.jp 20.903
2009/10/28(Wed) 15:43:43 www.google.com 4.407
2009/10/28(Wed) 15:43:52 www.mixi.jp 3.801
2009/10/28(Wed) 15:44:01 www.rakuten.co.jp 3.964

real    0m36.103s
user    0m0.008s
sys     0m0.036s
```

そこで、pinglog ver 3.0 は破棄し、pinglog ver 2.0 を繰り返し呼び出す pingset を新たに作ることにします。

ついで各ホストごとにログ・ファイルへ吐き出す機能も装備することにしました。

pingset ver 1.0

```
#! /bin/bash
# pingset ver 1.0 / (C) ABC co. IT-Team Yakoshi
LANG=C
opt_c=5;

function usage()
{
        cat <<EOF
$0 [-h][-c count] host-filename
ping logger
   repeat pinging and display TOD and RTT
   -h    print this message
   -c    pinging count

EOF
        exit 0
}

while getopts hc: opt
do
        case $opt in
        c) opt_c=$OPTARG ;;
        h) usage ;;
        *) exit 0 ;;
        esac
done
let OPTIND--
shift $OPTIND
[ $# = 0 ] && usage
```

```
for HOST in `cat $1`
do
        pinglog -c $opt_c $HOST >> $HOST.log &
done
wait
```

pingset はpinglog とかなりの部分が同じで、最後に ping をする部分が異なるだけです。pinglog 2.0 をコピーして作成するとよいでしょう。

for 文以降が異なります。まず ping を実行していた部分を pinglog に置き換えています。また pinglog の結果を「ホスト名.log」に記録しています。またアンパサンド(&)を使って、各 pinglog はバックグラウンド実行を行っています。

最後に wait コマンドで、バックグラウンド実行中の pinglog が全て終わるまで停止させています。このコマンドがないと、すぐに pingset が終了し、それに合わせて各 pinglog も強制終了されていまいます。

実行してみると、実行時間が 1/3 ほどに低減されています。

```
$ time pingset -c 10 myhost.txt

real    0m9.072s
user    0m0.000s
sys     0m0.080s
```

動きはこれで良いのですが、メンバーに使ってもらうと、いくつかアイデアが出てきました。

- ✓ 実行するとカレント・ディレクトリにログ・ファイルが散乱して見苦しいので、オプションを指定してログの格納場所を指定したい。またディレクトリが無ければ作る。
- ✓ ログ・ファイル名が「ホスト名.log」ではなく、「ホスト名.txt」という風に拡張子（suffix）を指定したい。
- ✓ デフォルトの値を usage で表示してほしい。

これら要望を取り入れて、完成版 pingset (ver1.1)が出来上がりました。

実行例

```
$ pingset -c 10 -d logs -s txt myhost.txt
$ ls -l logs
合計 16
-rw-rw-r-- 1 ycos ycos 46 10 月  28 16:45 www.google.com.txt
-rw-rw-r-- 1 ycos ycos 43 10 月  28 16:45 www.mixi.jp.txt
-rw-rw-r-- 1 ycos ycos 49 10 月  28 16:45 www.rakuten.co.jp.txt
-rw-rw-r-- 1 ycos ycos 48 10 月  28 16:45 www.yahoo.co.jp.txt

$ pingset
./pingset [-h][-c count][-d dir][-s suffix] host-filename
ping logger front end ver 1.1
   repeat pinging and display TOD and RTT
   -h    print this message
   -c    pinging count (default 5)
   -d    log file directory (default .)
   -s    log file suffix (default log)
```

完成版 pingset

pingset ver 1.1

```
#! /bin/bash
# pingset ver 1.1 / (C) ABC co. IT-Team Yakoshi
LANG=C
opt_c=5;
opt_d=".";
opt_s="log";


function usage()
{
        cat <<EOF
$0 [-h][-c count][-d dir][-s suffix] host-filename
ping logger front end
   repeat pinging and display TOD and RTT
   -h    print this message
   -c    pinging count (default 5)
   -d    log file directory (default .)
   -s    log file suffix (default log)

EOF
        exit 0
}

while getopts hc:d:s: opt
do
        case $opt in
        c) opt_c=$OPTARG ;;
        d) opt_d=$OPTARG
           [ -d $opt_d ] || mkdir $opt_d ;;
        s) opt_s=$OPTARG ;;
        h) usage ;;
        *) exit 0 ;;
        esac
done
let OPTIND--
shift $OPTIND
[ $# = 0 ] && usage

for HOST in `cat $1`
do
        pinglog -c $opt_c $HOST >> $opt_d/$HOST.$opt_s &
done
wait
```

### 2.4 crontab 再び

pinglog, pingset が完成したので、スケジュール実行できるよう crontab に組み込みます。ここで注意しなければならないのは、crontab では$PATH を/bin, /usr/bin に限定している点です。~/bin に pinglog, pingset を配置した場合には、フルパスで指定するか、PATH の設定を行う必要があります。

```
PATH=/bin:/usr/bin:/home/student/bin
HOME=/home/student/
* * * * * pingset -d logs myhosts.txt
```

## 3.スクリプトの作成事例（応用編）

pinglog のおかげで、ネットワークの状況を監視できるようになりましたが、どの時間帯に問題があるのか解析するにはとても手間がかかります。そこで作ったログ・ファイルを元にグラフを描くことにしました。

グラフ作成には gnuplot を使っています。予め yum を使ってインストールしておいてください。

mkgraph ver 1.0

```
#! /bin/bash
# pinglog make graph ver 1.0 / (C) ABC co. IT-Team Yakoshi
WORKDIR=mkgraph$$
trap    "rm -rf $WORKDIR" EXIT TERM

PLOT=ping.plot
opt_d="logs"
opt_s=1

function usage()
{
        cat <<EOF
$0 [-h][-d dir] graph-filename
ping logger
   repeat pinging and display TOD and RTT
   -h    print this message
   -d    log file directory (default logs)
   -s    silent mode

EOF
        exit 0;
}

# option process
while getopts shd: opt
do
        case $opt in
        s) opt_s=0;;
        d) opt_d=$OPTARG
            [ -d $opt_d ] || {
                echo "Can't access $opt_d."
                exit 1
            } ;;
        h) usage ;;
        *) exit 0;;
        esac
done
let OPTIND--
shift $OPTIND
[ $# = 0 ] && usage
FNAME=$1.gif

# log file copy
mkdir $WORKDIR
for logfile in $opt_d/*
do
```

```
        base=`basename $logfile`
        [ $opt_s = 1 ] && echo "copy logfile $logfile to $WORKDIR"
        cat -n $logfile > $WORKDIR/$base || exit 2
done

# Create plot file
cat <<EOD > $PLOT
# Generated by $0 at `date`
set term gif
set out '$FNAME'
EOD

files=( $WORKDIR/* )
let max=${#files[*]}-1
echo -n "plot " >> $PLOT
for ((i=0;i<${#files[*]};i++))
do
        title=`echo ${files[$i]} |cut -d. -f2`
        echo -n ¥"${files[$i]}¥" u 1:5 w l t ¥"$title¥" >> $PLOT
        [ $i -lt $max ] && echo -n ", " >> $PLOT
done
cat <<EOD >> $PLOT

set out
EOD
gnuplot $PLOT
[ $opt_s = 1 ] && echo "$FNAME created"
```

### 3.1 ワークファイル

ログ・ファイルをいったん作業用のディレクトリにコピーする為に、ディレクトリを作成し、そこに指定されたログ・ファイルをコピーします。このとき cp ではなく cat –n を使って行番号をふっています。

また同時に実行した際にファイルを潰しあわないよう、作業用のディレクトリは毎回違う名称を使っています。$$はプロセスＩＤなので、同時に実行したとしても名前が重複することはありません。

また、複雑な処理をしているので途中でエラーになる場合があります。エラー終了でも作業用のディレクトリを確実に削除するため、先頭部分で trap を使っています。

また他のディレクトリからファイルをコピーする際に、ディレクトリ部分を削除する basename コマンドを使っています。

```
$ basename   /a/b/c
c
```

basename はスラッシュを含む文字列のうち、先頭から最後（最も右側）のスラッシュまでを削除します。

### 3.2 ログ・ディレクトリ

ログ・ディレクトリは –d オプションで指定されたものがなければ、エラーとします。判定は [ コマンドの –d （ディレクトリ）で行い、|| コマンドを使って偽であれば、{}内を実行します。このスクリプトでは簡単なメッセージを表示後、値１で終了します。

### 3.3 配列の扱い

変数に代入する際に（）を用い、複数の値を指定すると配列になります。要素を参照する場合は添え字を[]で囲み、変数全体を｛｝で括ります。

要素数は、変数名の直前に#を付け、添え字[*]で表します。

```
$   prime=(2 3 5 7 11 13)
$   echo   ${prime[3]}
7
$   echo   ${#prime[*]}
6
```

スクリプトでは files=( $WORKDIR/*) によって、ログ・ファイルの一覧を file[0]~file[n] に格納しています。

### 3.4 gnuplot コマンド

このスクリプトは、ping.plot という gnuplot 用のコマンドファイルを作成し、それを gnuplot へ投入します。例えば以下の様なファイルが作成されます。

```
#   コメント
set    term    gif
set    out   'today.gif'
plot   "mkgraph23153/www.google.com.log"   u   1:5   w   l   t   "google",
set    out
```

コマンドの解説

- ✓ set term gif　　出力形式を GIF に切り替える
- ✓ set out 'ファイル名'　　出力ファイルを指定
- ✓ plot 'ファイル名'　　データファイルを読み取り、折れ線グラフを作成
  - ➢ u x:y　　データファイルの X、Y 軸に用いる項目位置(use の略)
  - ➢ w l　　折れ線を指定（with line の略)
  - ➢ t 'タイトル'　　グラフの凡例に表示するタイトル
- ✓ set out　　ファイルのクローズ（終了処理）

plot で、複数の要素（ファイル）からグラフを生成する場合は、カンマ(,)で区切り、ファイル名から続く要素を記述します。

今回のデータファイルはログ・ファイルの先頭に cat –n で行番号を付与しています。

```
     1   2009/10/26(Mon)    13:42:05    www.google.com    4.382
     2   2009/10/26(Mon)    13:43:05    www.google.com    4.408
                                  :
     ▲(1)                                                  ▲(5)
```

つまり、行番号と平均 RTT 時間をそれぞれ X 軸、Y 軸の値として折れ線グラフを作成します。折れ線の色は gnuplot が自動的に割り当てます。

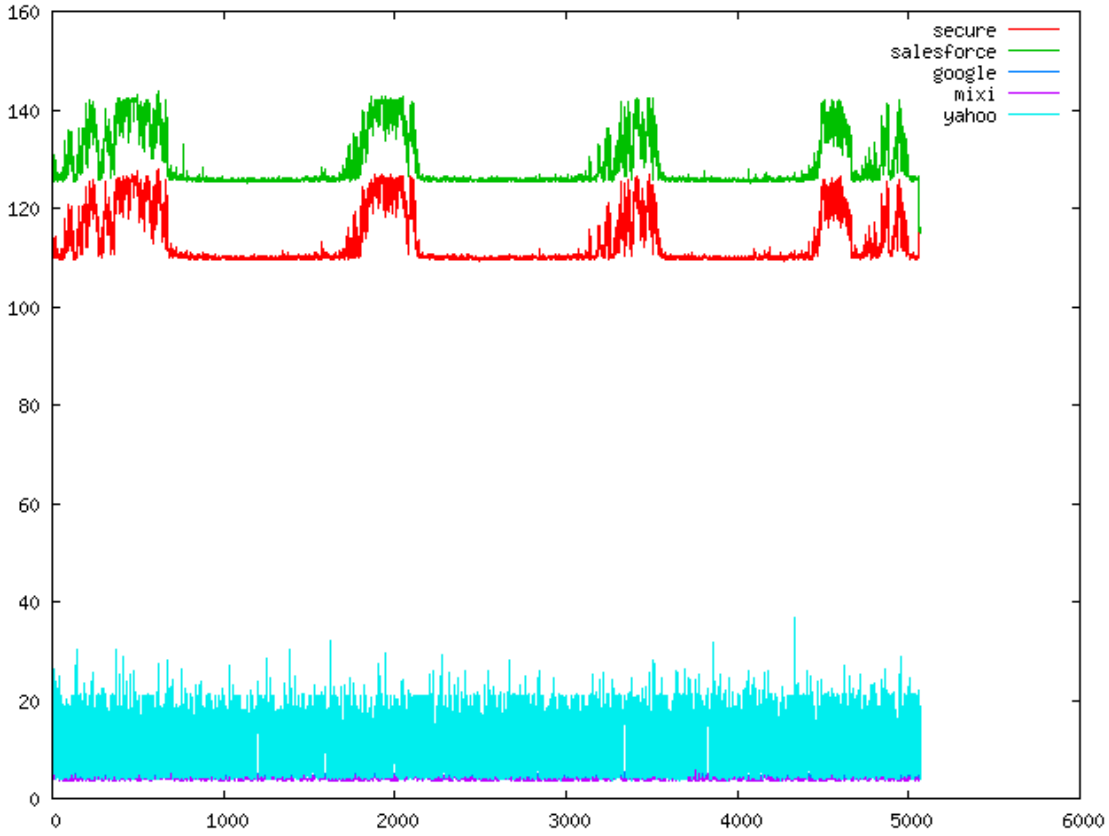
凡例は、ログ・ファイル名（≒FQDN）をドット区切った２つめの要素をタイトルとします。例えば mkgraph$$/www.**mixi**.jp.log の場合、タイトルは mixi となります。

このコマンドファイルを、gnuplot へ投入すると自動処理が行われ、GIF ファイルが生成されます。gnuplot を引数なしで実行すると、別ウインドが表示され対話モードに移行します。

```
$　gnuplot　　ping.plot
```

### 3.5 実行例

mkgraph を実行すると、today.gif ファイルが生成されます。



付録：実習キットに含まれるスクリプト一覧

| 名称 | シェル | 概要 |
|---|---|---|
| allip | bash | 自設定をもとにネットワーク中の機器の IP アドレス一覧を生成 |
| box | bash | 文字列を罫線で囲む |
| calc | bash | bc のフロントエンド |
| chkpsent | bash | パスワードの妥当性チェック |
| cntr | csh | 文字列のセンタリング |
| del | bash | ファイル削除（別名保管） |
| dice | ksh | 乱数（１～６表示、サイコロ） |
| dtree | csh | ディレクトリ構造の表示 |
| fbiff | csh | 指定されたファイルができたら返事する |
| fromwin | ksh | ログイン元のホスト名表示 |
| gap | ksh | 数列の中から間隙を表示する |
| htbl | bash | テキストファイルを元に HTML 形式の表を作成する |
| just | ksh | 文字列の片寄（左寄せ、センタリング、右寄せ） |
| last_cnt | bash | last(1) 出力を集計 |
| log | bash | 作業の記録 |
| mkindx | bash | テキストファイルから爪見出しを作成 |
| mn | bash | 数あてゲーム |
| myinfo | ksh | システム状況の表示 |
| repeat | bash | コマンドの繰り返し |
| rn | bash | ファイル名の変更 |
| rpmlist | bash | RPM の一覧（パッケージ名付き） |
| scale | bash | スケール(....+....1..)の表示 |
| shf | ksh | ファイル中の指定されたキーワードを強調表示 |
| spr | bash | 印刷しやすいようテキストファイルを成形 |
| trunk | bash | ファイルの行を切り出す（開始行・終了行を指定） |
| userpid | csh | 指定したユーザで動作している PID の一覧 |
| wdate | bash | 世界時計 |
| xpmdraw | bash | イメージ生成ツール（test.xd はコマンドファイル例） |
| ydup | bash | ファイルの各行を連結または横方向に複写 |
| ysub | ksh | ファイルの特定キーワードを、別のコマンド実行結果で置換 |

Source)　http://ycos.sakura.ne.jp/LA