Shell Scripting for MySQL Administration

An Introduction

Bob Burgess
radian6 Technologies

MySQL User Conference 2009

Who Are *You?*

- Programmed before, at least a little
- Starting out with Unix/Linux
- Minimal exposure to "shell"
- Probably moving from Windows

Windows "cmd" → Bash shell

Scope

- Linux *Bash* shell – Beginner Level
  - Variables
  - Parameters
  - Tests
  - Flow Control
  - File Redirection

Scope

- Common Utilities/Commands
  - hostname
  - tr
  - cut
  - ls
  - read
  - nail
  - sed

Scope

- Script Interface to MySQL
  - command-line parameters
  - retrieving a single value
  - iterating through a result set
  - backup control
  - security

Scope

- Examples
  - Monitoring database status variables
  - Monitoring/controlling replication
  - Monitoring performance
  - Alarming and alarm emails/pages
  - Controlling backup
  - Monitoring environment metrics such as disk space

Scope – NOT

- Complete Course on Shell
- Advanced Shell Techniques
- Amazing Tricks
- Under The Hood (MySQL / Linux)

The Shell

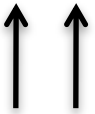
The Linux Shell

- Command Processor
- Bash
- Bourne Shell *(Steve Bourne, 1975)*
- **B**ourne-**A**gain **Sh**ell
- BASh

Variables

- Named with letters, numbers, “_”
- Names start with letters or “_”
- Holds strings
- Math
  - built-in does 64-bit signed ints
  - utility “bc” does floating-point
- Example:

```
TIMEOUT=60
```



Variables

- Referenced with “$”

  `$TIMEOUT`

- Surround name with { } when confusion is possible

  `${TIMEOUT}`

- Can represent programs too

```
MYSQL=/usr/local/mysql/bin/mysql
$MYSQL -e”select 1”
```

Variable Scope

- Default: within current shell or script
- Use **`export`** to make variable visible within commands launched from this shell or script
- No way* to make variables visible to “parent” shell

** unless you “source” the script which sets the variables*

Script Parameters

- $*n*
- $0 – script name itself
- $1 – first parameter
- $2 – second parameter
- ${10} – tenth parameter
- etc.

Built-In “Variables”

- **`$#`** – number of parameters
- **`$?`** – return value from last command
- **`$$`** – Process ID of this shell

Default I/O Channels

- Standard In – 0
- Standard Out – 1
- Standard Error – 2

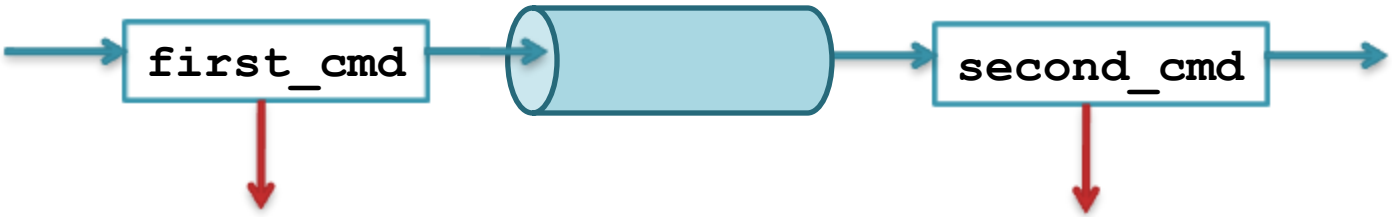
File Redirection

- Use non-default sources or destinations for program input and output

```
$ echo "hello"
hello
$ echo "hello" >some_file
$ cat some_file
hello
```

Pipes

- Daisy-chain commands
  `first_cmd | second_cmd`
- Connects *stdout* of one command to *stdin* of the next



File Redirection

- Use a file instead of keyboard input
  `some_command <input_file`
- Append the output to an existing file
  `echo "new line" >>my_file`
- Save the error output
  `mycmd 2>err_file >log_file`
- Save error and normal together
  `mycmd >log_file 2>&1`

Tests

- Two types: String or Numeric
- Useful in **`if/then`** and **`while`**

Tests: String

- `[ "$ANSWER" = "NO" ]`

- `[ "$ANSWER" \> "A" ]`
- `[ "$ANSWER" \< "A" ]`

- `[ -z "$ANSWER" ]` *(null)*
- `[ -n "$ANSWER" ]` *(not null)*

Tests: Numeric

- `[ $VAL -eq 1 ]`
- `[ $VAL -ne 1 ]`

- `-le`, `-lt`, `-ge`, `-gt`

- `[ 0$VAL -eq 1 ]`

Tests: File

- File exists:

```
[ -e "$MYFILE" ]
```

- File does not exist:

```
[ ! -e "$MYFILE" ]
```

- File has >0 bytes:

```
[ -s "$MYFILE" ]
```

- File exists and is a directory:

```
[ -d "$DIR" ]
```

Flow Control!

- if / then
- while
- for

IF / THEN

```
if [ $VAL -eq 1 ]; then
   statements
else
   statements
fi
```

IF / THEN

```
if command; then
    statements if command succeeds
else
    statements if command fails
fi
```

While

```
while [ $VAL -eq 1 ]; do
   statements
done
```

For

```
ALLPORTS="3306 3308 3319"
for PORT in $ALLPORTS; do
   echo $PORT
   statements
done
```

For

```
cd /var/log/mysql
for LOGFILE in mysql-bin.[0-9]*
do
   echo $LOGFILE
   statements
done
```

Two handy shell features

- Run *this* and insert the result *here*
  `` `command` `` *(backticks)*

```
$ echo "Script started at `date`" >$LOGFILE
$ cat $LOGFILE
Script started at Mon Apr  6 15:37:48 ADT 2009
```

- Run a script and come back
  `.` *other_script.sh* (source with a dot)

Writing a Bash Script

- First line: which executable to use:

```
#!/bin/bash
```

Writing a Bash Script

- Suggested Comments...

```
#   script_name.sh [{maxrows}]
# Send an email if 'queue' table
# exceeds {maxrows} rows.
# Parameters:
# Input: {maxrows}(defaults to 100)
# Output: email is sent to alert@job.com
```

Error Reporting

- An Error Log

```
ERRLOG=/opt/mysql/log/script.err

if [ $OHNO -eq 1 ]; then
  echo "script failed" >>$ERRLOG
  exit 1
fi
```

Error Reporting

- Output to *stderr*

```
if [ $OHCRAP -eq 1 ]; then
  echo “script failed” >&2
  exit 1
fi
```

Exit Status

- 0 = “good”

```
exit 0
```

- Not 0 = “Not good”

```
ERR_DBDOWN=900
exit $ERR_DBDOWN
```

- or -

```
exit 1
```

Running the Script

- Turn on the “execute” bit

  `chmod +x` *script.sh*

- ***dot slash*** *or* complete path

  `./`*script.sh*

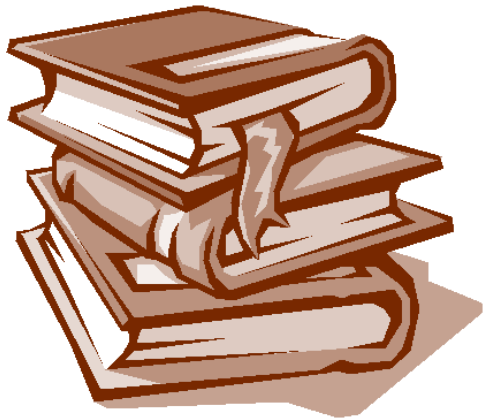
- See under the hood:

  `bash -x` *script.sh*

Handy Utilities

Utilities: man

- The “M” in RTFM



Utilities: info

- A newer **`man`**

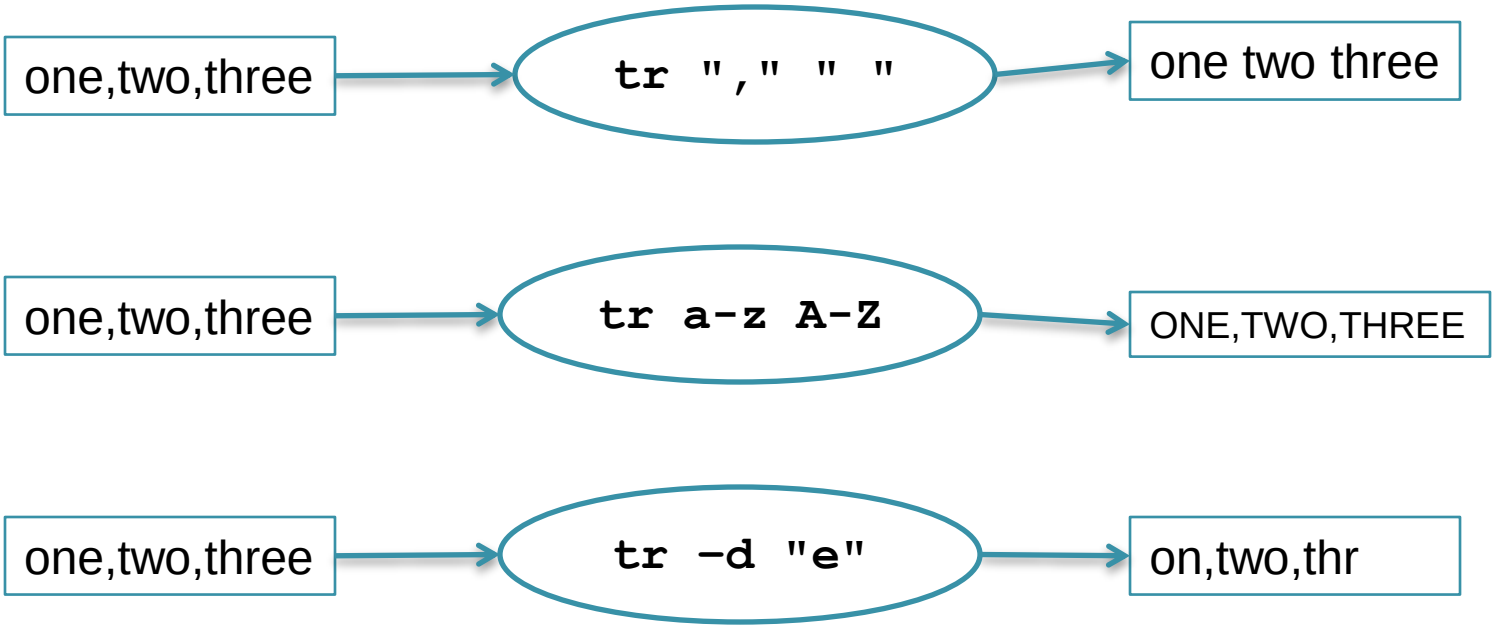
Utilities: hostname

- “Who Am I ?”
- Handy within backticks:

```
HOST=`hostname -s`
$MYSQL -h${HOST}
```

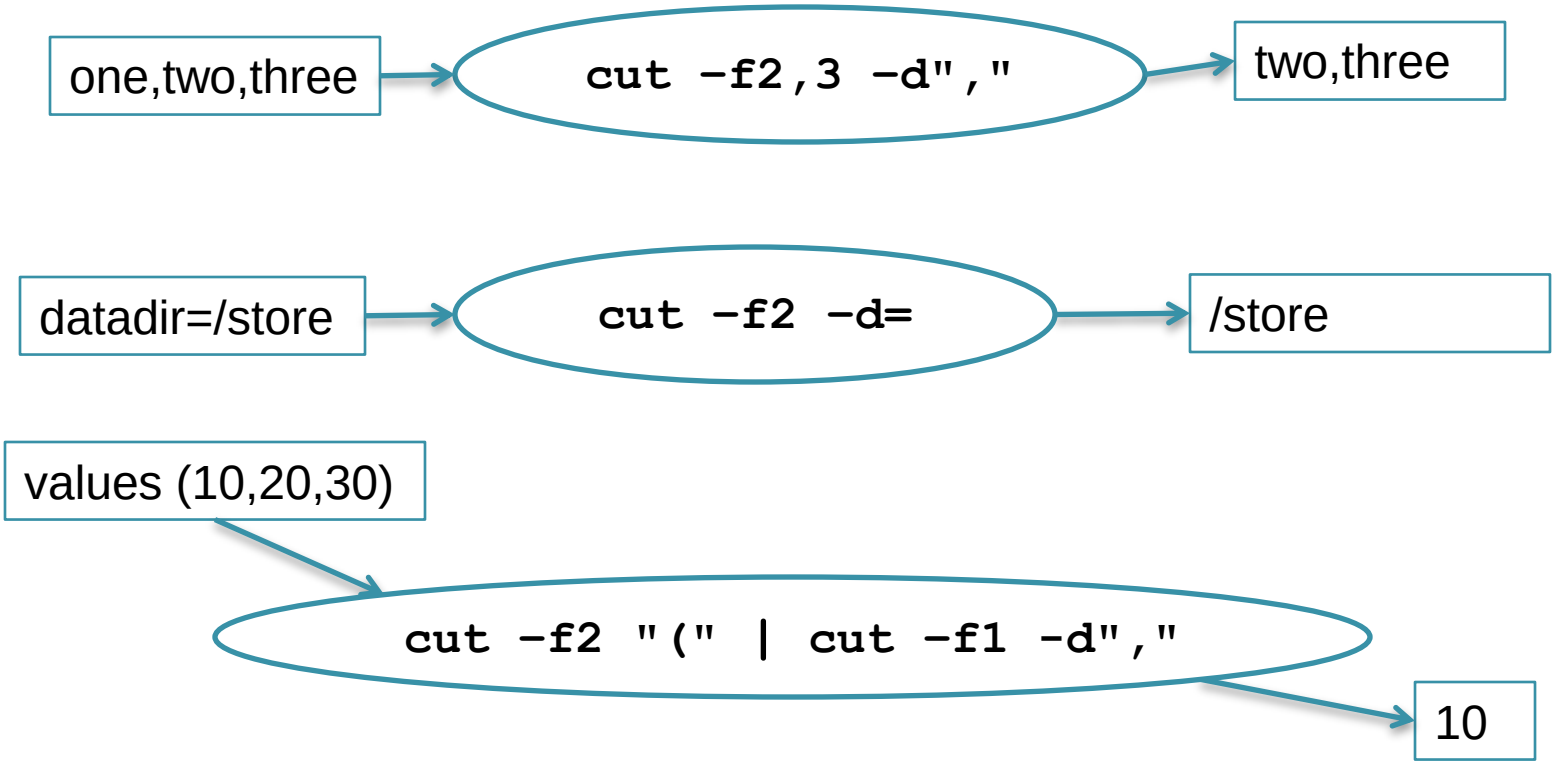
Utilities: tr

- Translate characters
- Delete characters



Utilities: cut

- Return sections from a line



Utilities: ls

- Options make it powerful
- List files, one on each line

```
ls -1
```

- List files with details, latest modify time at the end

```
ls -ltr
```

- Example: latest binlog file number

```
ls -1tr /var/log/mysql/mysql-bin.[0-9]*|tail -1|cut -f2 -d.
```

Utilities: nail

- Email!
- Assumes the server is configured to send emails to the outside world
- Example: email contents of one variable:

```
echo $MSG |
 nail -s "Subject" bob@job.com
```

Utilities: nail

- Email contents of a file in the body and attach another file

```
nail -a ${ATT_FILE} -s "$SUBJ"
$SENDTO <$MSGBODY
```

- Email a file in the body and alias the From address

```
nail -s "The List" -S
from="santa@claus.com" $SENDTO
<list.txt
```

Utilities: sed

- Very powerful
- Many commands
- Well documented

Utilities: sed

- Translations too complex for ‘tr’

```
echo $VX |
 sed 's/Error://' >$LFILE
```

- Delete 1st three lines from input

```
cat $LF | sed '1,3d'
```

Utilities: date

- Writing to log files

```
echo "Starting at `date`" >$LOGFILE
Starting at Mon Apr  6 23:07:07 ADT 2009
```

- Dated filenames

```
LOGFILE=/logs/myscript.`date +%Y%m%d`.log
```

myscript.20090416.log

- Rotating filenames

```
LOGFILE=/logs/myscript.`date +%a`.log
```

myscript.Wed.log

Utilities: date

- Timing

```
T_START=`date +%s`
    do stuff
T_END=`date +%s`
T_ELAPSED=$((T_END-T_START))
```

Utilities: date

- When was **$x** seconds ago?

```
UPTIME=$((`date +%s` - $x))
UP_AT=`date -d "1970-01-01 UTC $UPTIME seconds"`
```

Utilities: read

- Accepts a line, puts it into variables
- Useful with While loops

```
while read LINE; do
   echo $LINE
done <$FILE

$MYSQL -e"select a from T" | \
while read ROW ; do
   echo $ROW
done
```

Utilities: grep

- Returns lines matching a pattern
- From a file
  `grep "SUMMARY" $TMPFILE`
- From *stdin*
  `some_cmd` **`| grep $PATTERN`**

Utilities: grep

- Just count the matches
  ```
  grep -c "Error" $LOGFILE
  ```
- Put the count in a variable
  ```
  CNT=`grep -c "Error" $LOGFILE`
  ```
- Show lines that don't match
  ```
  ps -ef|grep mysqld|grep -v grep
  ```

Utilities: grep

Other Options

- Case-insensitive match

  `grep -i`

- Match only ***n*** lines then stop

  `grep -m` *`n`*

- Show matching line plus ***n*** lines after

  `grep -A` *`n`*

- Quiet (only output is **`$?`**) – good for **`if`**

  `grep -q`

Utilities: grep

Match Patterns

- Any character: `.`
- Range of characters: `[2-8]`
- Set of characters: `[E,H,Q]`
- Character repeated: `*`
- Limit pattern to start of line: `^`
- Limit pattern to end of line: `$`

Utilities: awk

- Powerful pattern-matching language
- Common use: getting values from a line

- Similar to **`cut`**
- Can specify the field separator (normally “whitespace”) with **`-f`**

Utilities: awk

- Pluck data from randomly-spaced columns
  Example: get status and time from all mysql sessions:

```
mysql -e"show processlist" |
  awk '{print $5,$6}'
```

```
19516545   theuser   server1:41083   sphere  Sleep   0      NULL
```


```
Sleep,0
```

Utilities: awk

- Pluck data from randomly-spaced columns, *with a condition*
  Example: get status and time from all mysql sessions:

```
mysql -e"show processlist" \
| awk '{print $5,$6}' \
| grep -v Sleep \
| grep -v Connect \
| awk "{if (\\$2 > $MAX_TIME) print $2}"
```

mysql Command Line

mysql Command Line

Relevant options

- connection info: **`-u -p -h -P`**
- execute a command: **`-e`**
- change output format: **`-s -ss -B`**

mysql Command Line

Default options in my.cnf

```
[client]
port    = 3307
socket = /var/run/mysqld/mysqld.sock

[mysql]
user          = theuser
password    = xyzxyz
database    = mydb
default-character-set = utf8
```

mysql Command Line

Output Format: Interactive

```
mysql> select * from util.sample;
+-------+-------+
| digit | name  |
+-------+-------+
|     1 | one   |
|     2 | two   |
|     3 | three |
+-------+-------+
3 rows in set (0.00 sec)
```

mysql Command Line

Output Format: SQL from stdin

```
$ echo "select * from util.sample"|mysql
digit    name
1        one
2        two
3        three
```

mysql Command Line

Output Format:

SQL as `-e` option, output to console

```
$ mysql -e"select * from util.sample"
+-------+-------+
| digit | name  |
+-------+-------+
|     1 | one   |
|     2 | two   |
|     3 | three |
+-------+-------+
```

mysql Command Line

Output Format:

SQL as `-e` option,

output to another program

```
$ mysql -e"select * from util.sample"|grep ".*"
digit    name
1        one
2        two
3        three
```

No table lines when stdin/stdout are redirected.

mysql Command Line

Output Format: `-s` adds silence

```
+-------+-------+
| digit | name  |
+-------+-------+
|     1 | one   |
|     2 | two   |
|     3 | three |
+-------+-------+
```

`-s`

```
digit   name
1       one
2       two
3       three
```

`-s`
`-ss`

```
1       one
2       two
3       three
```

mysql command line

- Retrieving one value

```
V=`$MYSQL -e"select min(digit)
  from util.sample"`

echo $V
min(digit) 1
```

mysql command line

- Retrieving one value

```
V=`$MYSQL -s -e"select min(digit)
  from util.sample"`

echo $V
1
```

mysql command line

- Retrieving several values *from one line*

```
L=`$MYSQL -s -e"select min(digit),
  max(digit), min(name) from
  util.sample"`
echo $L
 1 3 one
set - $L
echo "Highest digit is $2"
 Highest digit is 3
```

mysql command line

- Iterating through several lines

```
L=`$MYSQL -s -e"select digit,name
  from util.sample"`
echo $L
  1 one 2 two 3 three
```

Nope.

mysql command line

- Iterating through several lines

```
$MYSQL -s -e"select digit,name
  from util.sample" \
  | while read L; do
      echo $L
    done
```

```
1 one
2 two
3 three
```

mysql command line

- Iterating through several lines

```
$MYSQL -s -e"select digit,name
  from util.sample" \
  | while read D N; do
      echo "Digit $D is called $N"
    done
```

```
Digit 1 is called one
Digit 2 is called two
Digit 3 is called three
```

mysql command line

- Hiding the password

File *mysql_access*:

```
MYSQL_USER="root"
MYSQL_PASSWORD="secret"
```

```
chmod 400 mysql_access
```

```
# ls -l mysql_access
-r-------- 1 root root 44 Nov 17  2007 mysql_access
```

mysql command line

- Hiding the password

In your script:

- **mysql_access**

```
MYSQL="/usr/local/mysql/bin/mysql
  -u$MYSQL_USER -p$MYSQL_PASSWORD"
```

cron

- Wakes up every minute to run stuff
- **`crontab -e`** to edit the to-do list
- **`crontab -l`** to list the list
- Has its own execution path!

cron

Schedule format

minute	0-59
hour	0-23
day of month	1-31
month	1-12
day of week	0-7 (0 or 7 is Sunday)

cron

Schedule examples

- Every day at 1:15pm:
  `15 13 * * *`
- Every 5 minutes:
  `*/5 * * * *`
- Every Monday at 7 am:
  `0 7 * * 1`
- Every 10 minutes, 2-3am, 8-9am, and 2-7pm, on weekends in February:
  `*/15 2,8,14-18 * 2 6,7`

cron

- Run SQL and discard output:

```
* * * * * /usr/local/mysql/bin/mysql -
e"update tbl set col=col*1.2"
>/dev/null 2>&1
```

- Run SQL. Log output and errors:

```
* * * * * /usr/local/mysql/bin/mysql -
e"update tbl set col=col*1.2"
>>/log/update.log 2>&1
```

cron

- Run a script from cron:
  * * * * * /opt/mysql/script/myscript.sh
- Don’t rely on cron’s $PATH! Use absolute paths for everything!

Example: MySQL Status variables

Send mail when Threads_connected reaches 1000:

```
TC=`$MYSQL -s -e"show status like
  'Threads_connected' "|cut -f2`
if [ $TC -ge 1000 ]; then
   echo "Thread count at $TC" | \
  nail -s"Thread emergency"
  admin@job.com
fi
```

Example: Replication

See how far behind or if we're stopped:

```
$MYSQL -e'show slave status\G' >$TMP
SB=`grep Seconds_Behind_Master $TMP |
  cut -f2 -d: | tr -d " "`
SBNUM=$SB
if [ "$SB" = "NULL" ]; then
  SBNUM=99999999
fi
if [ $SBNUM -le $MAXBEHIND ]; then
  exit 0
fi
```

Example: Replication

Check IO and SQL Running:

```
IO_R=`grep Slave_IO_Running
$TMPFILE |cut -f2 -d: |tr -d " "`
SQL_R=`grep Slave_SQL_Running
$TMPFILE | cut -f2 -d: |tr -d " "`
```

Example: Replication

Test and Alarm:

```
ERRMSG=""
if [ "$SB" = "NULL" ]; then
  ERRMSG="Replication Down."
elsif [ $SBNUM -ge $MAXBEHIND ]; then
  ERRMSG="Replication behind ${SBNUM}s."
fi
if [ -n "$ERRMSG" ]; then
  echo "$ERRMSG IO:${IO_R}, SQL:${SQL_R}"\
 | nail -s"Replication" admin@job.com
fi
```

Example: Performance Monitor

```
T0=`date +%s`
$MYSQL -e"select avg(x) from
  bigtable" >/dev/null
T1=`date +%s`
T=$((T1-T0))
if [ $T -gt $TMAX ]; then
   echo "Test query took ${T}s."\
   | nail -s "Slow DB"
admin@job.com
fi
```

Alarming

- Email directly from script.

```
echo "Error message" | nail
```

- Easy to write, but *hard to maintain*.

Alarming

- Use an alarm log.

```
echo "WARNING:monitor_repldb:Prod:
  ${port}:${host}:`date +%Y%m%d-%H%M%S`:
  Error message." >>$ALARM_LOG
```

- One alarm log file per server
  (or one for many – NFS)
- Record of all alarm events
- Read / Email with a separate script

Backup – Components

- Shutdown
- Copy (tar / gzip)
- Startup

- Generate config files
- Call ibbackup (test for success)

- Call mysqldump (test for success)

Backup – Shutdown

```
# Start shutdown:
echo "Starting shutdown." >$BACKUP_LOG
$MYSQLADMIN shutdown 2>>$BACKUP_LOG >>$BACKUP_LOG


# Wait for shutdown to complete:
STOP_WAITING_TIME=$((`date +%s` + ($SHUTDOWN_TIMEOUT * 60) ))
MYSQLD_RUNNING=`ps -ef|grep "$PS_PATTERN"|grep -v grep|wc -l`
while [ $MYSQLD_RUNNING -eq 1 -a `date +%s` -lt
$STOP_WAITING_TIME ]; do
  sleep 3
  MYSQLD_RUNNING=`ps -ef|grep "$PS_PATTERN"|grep -v grep|wc -l`
done


# Error if it didn't shut down:
if [ $MYSQLD_RUNNING -eq 1 ]; then
  echo "Shutdown did not finish in time.  Ending." >>$BACKUP_LOG
  echo "WARNING:cold_backup:`date +%Y%m%d-%H%M%S`:Cold Backup of
       $HOST:$DB_PORT Failed: Shutdown timed out." >>$ALARM_LOG
  exit 1
else
  echo "Shutdown Complete at `date`" >>$BACKUP_LOG
fi
```

Backup – Cold Copy

```
# echo "Start backup to $BACKUP_FILE at `date`" >>$LOG
if tar czf $BACKUP_FILE ${SOURCE_DIR}/* >>$LOG 2>&1
then
  echo "File copy (Tar) complete at `date`" >>$LOG
else
  echo "Error from Tar at `date`" >>$LOG
  echo "WARNING:cold_backup:`date +%Y%m%d-%H%M%S`:Cold
       Backup of $HOST:$DB_PORT Failed: Tar error."
       >>$ALARM_LOG
  exit 1
fi
```

Backup – Startup

```
echo "Restarting database." >>$BACKUP_LOG
/etc/init.d/mysql start
STOP_WAITING_TIME=$((`date +%s` + ($STARTUP_TIMEOUT * 60) ))
$MYSQLADMIN ping >/dev/null 2>&1
DB_UP=$?
while [ $DB_UP -ne 0 \
      -a `date +%s` -lt $STOP_WAITING_TIME ]
do
  sleep 3
  $MYSQLADMIN ping >/dev/null 2>&1
  DB_UP=$?
done
```

Backup – Generate Config

```
# Write the source config
echo "[mysqld]" >$SOURCE_CNF
$MYSQL -e"show variables like 'innodb%';show variables like
'datadir'" \
   | awk '{print $1"="$2}' >>$SOURCE_CNF
# Write the destination config
echo "datadir=$DESTINATION"                        > $DEST_CNF
echo "innodb_data_home_dir=$DESTINATION"           >>$DEST_CNF
echo "innodb_log_group_home_dir=$DESTINATION"      >>$DEST_CNF
grep "innodb_data_file_path" $SOURCE_CNF           >>$DEST_CNF
grep "innodb_log_files_in_group" $SOURCE_CNF       >>$DEST_CNF
grep "innodb_log_file_size" $SOURCE_CNF            >>$DEST_CNF
```

Backup – Call backup or dump

```
if $IBBACKUP $SOURCE_CNF $DEST_CNF >>$LOG 2>&1; then
  echo "# EDIT THIS FILE: CHANGE DIRECTORIES TO RECOVERY
LOCATION" >$DESTINATION/recover.cnf
  cat $DEST_CNF >>$DESTINATION/recover.cnf
  echo "* ibbackup finished successfully. " >>$LOG
  echo "INFO:hot_backup:`date +%Y%m%d-%H%M%S`: Successful
finish." >>$ALARM_LOG
  EXITCODE=0
else
  echo "* ibbackup: nonzero return code. *" >>$LOG
  echo "WARNING:hot_backup:`date +%Y%m%d-%H%M%S`: ERROR from
ibbackup." >>$ALARM_LOG
  EXITCODE=1
fi
```

- Calling mysqldump is basically the same.

Environment Monitoring

- Important one: disk space

```
DFTEMP=/tmp/monitor_disk.$$
df -HP >$DFTEMP &
sleep $DF_WAIT
if [ ! -s $DFTEMP ]
then  # df is blocked
  echo "WARNING:monitor_disk:$HOST:`date
      +%Y%m%d-%H%M%S`: Disk Space cannot be
      measured. (Usual cause: NFS error.)"
      >> $ALARM_LOG
else  # df is not blocked...
```

Environment Monitoring

```
cat $DFTEMP | awk '{ print $5 " " $1 " " $6}' | \
while read output;  do
  used=$(echo $output | awk '{ print $1}' | cut -d'%' -f1  )
  partition=$(echo $output | awk '{ print $3 }' )
  if [ $used -ge $DISK_ERROR ]; then
    echo "ERROR:monitor_disk:$HOST:`date +%Y%m%d-%H%M%S`:
         Disk Space on $partition at $used% (Threshold=
         $DISK_ERROR%)" >> $ALARM_LOG
  else
    if [ $used -ge $DISK_WARNING ]; then
      echo "WARNING:monitor_disk:$HOST:`date +%Y%m%d-
          %H%M%S`:Disk Space on $partition at $used%
          (Threshold=$DISK_WARNING%)" >> $ALARM_LOG
    fi
  fi
done
fi
```

Resources

- **`man`** pages
- **`info`** pages
- MySQL Docs
- Advanced Bash-Scripting Guide
  Mendel Cooper www.tldp.org/LDP/abs
- Google

Thank you!

Bob Burgess

bob.burgess@radian6.com

www.radian6.com/mysql