BASH(ish) Hacks: One-Liners and Other Sketchy Ideas

Comments via Google Docs are Welcome!

BASH “History”

- Pull up the Wikipedia page and scroll to the History section. BASH is OLD!

BASH History: Basics

- Issuing *history* lists your previous commands *stored in memory* (more on this later)
- *history 10* lists the last 10 commands (including, oddly enough, 'history 10' itself, so keep that in mind!)
- up and down arrows scroll sequentially through the history
- CTRL-R starts a search, start typing and the latest match will appear
- Notice the numbers? use *!123* to execute that command immediately
- Issue *!-5* to run the 5th prior command in your history.
- *!!* runs the immediate prior command, but why bother?! Just up-arrow-enter.
- Need to modify that numbered command? *!123:p* puts a copy of it at the end of the history, then just up arrow, then you can edit it!

BASH History: Advanced

- Temporarily disable history: *set +o history* (think +omit)
- Re-enable with *set -o history* (think -omit or “don’t omit”)
- Delete a command from history? *history -d 123* -- note that there is no range, so you can only delete one at a time!
- Omit certain commands AND deduplicate your history? Edit your .bashrc and add *HISTCONTROL=ignoreboth* this will omit commands that start with whitespace and omit duplicate commands
- Also in .bashrc, you can set *HISTSIZE=123* and *HISTFILESIZE=246* to limit the in-memory and written to file sizes of your history.
- To completely clear your history AND the history file while logging out, issue *history -c; history -w; exit* -- when next you (or someone else) logs in, that history will be. . . history!

BASH History: File vs Memory

- When you log in, the prior history is loaded from ~/.bash_history
- While logged in, your commands are tracked in memory only
- When you log out, the commands added during that session are appended to the file.
- If your session crashes or the system resets, the history for that session is lost. Think, the root user issues *shutdown -h now*
- When you issue *history -c* in the current session, that only clears the session history -- the history loaded from the file, and the file itself, are preserved.
- *history -w* writes what was read from the file + what was added during the session to the file.
- *history -r* reads the default history file, or a file you specify, into the current session.
- *history -a* appends the current session history to the history file, less the lines read from the file on login.

BASH History: Esoterica

- *history -n* compares the history loaded from the file at login to the current history file (think *diff*). It then appends to the in-memory history any lines that were added since that session started. This is useful if you use *screen* or *tmux* and need to pass or preserve history lines between sessions.
- *!?grep?* Will search for the immediate prior command which contained *grep* within it. You want to call your last command containing a grep, but you're not sure how many commands ago that was.
- *^oopsy^corrected^* acts like a sed inline search and replace on the immediate prior command, but only replaces the first instance of the "oopsy"
- *fc 7 10* opens the selected range of lines for editing prior to executing them.
- And the BASH History rabbit-hole goes much, much deeper than this!

BASH One-Liners:

- What are they? One-off hacks that live for a while in your BASH History. If you use them enough, they live on. If not, they die lonely and sad. If they are really useful, you turn them into bonafide scripts!
- *until ssh myhost.kristau.net; do sleep 3; done*
- *until ping -c 3 -q myhost.kristau.net; do sleep 20; done; mailx -s "Host is Online" kristau@gmail.com < /dev/null*
- *while clear; do uptime; echo; who; echo; dmesg | tail; sleep 20; done*
- *while true; do echo $RANDOM$RANDOM$RANDOM | cut -c 1-8; sleep 1; done*

Sketchy Ideas:

Older PIN Generator (this one is FUGLY)

```
#!/bin/bash
# pin_generator: random number pin generator
# Variables:
default_quantity=50;
default_length=12;
# Main:
if [ $# -eq 1 ]; then default_length=$1; fi
if [ $# -eq 2 ]; then default_length=$1; default_quantity=$2; fi
until [ $default_quantity == 0 ];
do {
   if [ $default_length -ge '1' ]; then my_random=$(($RANDOM*$RANDOM)); fi
   if [ $default_length -ge '6' ]; then my_random=$my_random$(($RANDOM*$RANDOM)); fi
   if [ $default_length -ge '12' ]; then my_random=$my_random$(($RANDOM*$RANDOM)); fi
   if [ $default_length -ge '18' ]; then my_random=$my_random$(($RANDOM*$RANDOM)); fi
   if [ $default_length -ge '25' ]; then echo "Maximum \$default_length is 24."; exit 1; fi
   echo $my_random | head -c $default_length;
   echo '';
   default_quantity=$(($default_quantity - 1));
} done;
```

Check for Updates, YUM Edition:

```
#!/bin/bash
# cron-yum-check-update: See if we have updates and e-mail someone about that.
# Variables
mailto='kristau@gmail.com'
host=`hostname`
tempfile=/tmp/yum-updates.txt
# Main
if ! yum check-update &> $tempfile
then
  mailx -s "Updates Available on $host" -a $tempfile $mailto < /dev/null &> /dev/null
else
  mailx -s "No Updates Available on $host" -a $tempfile $mailto < /dev/null &> /dev/null
fi
```

Apply Updates and Reboot, YUM Edition:

```
#!/bin/bash
# cron-yum-update: Apply all available yum updates and e-mail someone about that
# Variables
mailto='kristau@gmail.com'
mypid=$$
host=`hostname`
tempfile=/tmp/cron-yum-update-$mypid.txt
lockfile=/var/lock/cron-yum-update.lock
# First, make sure another instance isn't running or recently failed:
if [ ! -e $lockfile ]
then
  echo $mypid > $lockfile
else
  mailx -s "Error with cron-yum-update on $host: lock file present" $mailto < /dev/null &>
/dev/null
  exit 1
fi
# Check to see if we have updates available:
yum check-update
status=$?
if [ $status -eq '100' ]
then
  # We have updates! Apply them and (maybe) reboot!
  yum -y update &> $tempfile
  mailx -s "Updates applied to server $host" -a $tempfile $mailto < /dev/null &> /dev/null
  rm -f $lockfile
  # Optional: reboot the server.
  reboot
elif [ $status -eq '0' ]
then
  # We don't have updates! Let someone know about that.
  mailx -s "No Updates applied to server $host" -a $tempfile $mailto < /dev/null &> /dev/null
  rm -f $lockfile
else
  # Something must have gone horribly wrong?!
  mailx -s "Error attempting to update $host" -a $tempfile $mailto < /dev/null &> /dev/null
  # NOTE: lock file has not been removed.
fi
```

MySQL Backup Script

```
#!/bin/bash
# cron-mysql-backup: Perform a daily mysqldump backup of the databases on this server.
# Variables
datestamp=$(date +%Y-%m-%d-%H%M)
mailto='kristau@gmail.com'
backupdir='/root/mysql_backups'
workingdir='/root/bin'
tempdir='/tmp'
tempfile="$tempdir/$datestamp-cron-mysql-backup.tmp"
logfile="$backupdir/$datestamp-cron-mysql-backup.log"
dumpfile="$backupdir/$datestamp-mysql-backup.sql"
mysql_user='root'
#mysql_pw='' # This should be specified in the ~/.my.cnf file [client] section as
password=password
# Functions
## Check current replication status and return 0 if running, 1 if not.
function is_replicating {
  echo "select
performance_schema.replication_connection_status.service_state,performance_schema.replica
tion_applier_status.service_state from performance_schema.replication_connection_status,
performance_schema.replication_applier_status;" | mysql -u $mysql_user -N 2>&1 | sed -e
"s/\t/,/" > $tempfile
  if grep -q "ON,ON" $tempfile;
  then {
        #echo "Replication is running" >> $logfile
        return 0
  } else {
        #echo "Replication is NOT running" >> $logfile
        return 1
  } fi
  rm -f $tempfile
}
## Change the state of replication, Requires either OFF or ON as an argument.
function change_replication {
  if [ $1 == "ON" ]
  then {
        echo "Turning replication $1" >> $logfile
        if is_replicating
        then {
        echo "Replication is already running. No action taken." >> $logfile
```

```
        return 0
        } else {
        echo "Replication is OFF. Changing state to ON." >> $logfile
        echo "start slave;" | mysql -u $mysql_user >>$logfile 2>&1 || exit 2
        sleep 7 # Wait a bit for the status to change
        if is_replicating
        then {
        echo "Replication successfully started." >> $logfile
        return 0
        } else {
        echo "Error starting replication. Exiting." >> $logfile
        echo "Error starting replication. Please check $logfile." >&2
        exit 1
        } fi
        } fi
 } elif [ $1 == "OFF" ]
 then {
        echo "Turning replication $1" >> $logfile
        if is_replicating
        then {
        echo "Replication is ON. Changing state to OFF." >> $logfile
        echo "stop slave;" | mysql -u $mysql_user >>$logfile 2>&1 || exit 2
        sleep 7 # Wait a bit for the status to change
        if is_replicating
        then {
        echo "Error stopping replication. Exiting." >> $logfile
        echo "Error stopping replication. Please check $logfile." >&2
        exit 1
        } else {
        echo "Replication successfully stopped." >> $logfile
        return 0
        } fi
        } else {
        echo "Replication is already OFF. No action taken." >> $logfile
        return 0
        } fi
 } else {
        echo "Something is very wrong here. $1 is not a correct value. Exiting." >> $logfile
        echo "Error changing replication. Please check log file." >&2
        exit 1
 } fi
}
# Main
```

```
echo "Start time and system load averages: $(uptime)" >> $logfile
if ! is_replicating # If replication is stopped at this point, we have a problem!
then {
  echo "Replication was not running when we started. Exiting." >> $logfile
  echo "Error: Replication not running. Please check $logfile for details." >&2
  exit 1
} else { # Otherwise, we are good to go!
  echo "Replication verified in running state." >> $logfile
  change_replication OFF # Stop the replication before we back up
  # Now, we need to actually dump the databases!
  echo "Starting mysqldump to $dumpfile." >> $logfile
  if mysqldump -u $mysql_user --all-databases --add-drop-database 2>> $logfile 1>> $dumpfile
  then {
        echo "Database backup successful to $dumpfile." >> $logfile
  } else {
        echo "Database backups failed. Exiting." >> $logfile
        echo "Error backing up databases. Please check $logfile for details." >&2
        exit 1
  } fi
} fi
# If we made it this far, we should have a good dump of the databases in $dumpfile. First, let's
re-start replication:
change_replication ON
# Then, compress $dumpfile to save space:
echo "Compressing $dumpfile with gzip." >> $logfile
gzip -1 -v $dumpfile >> $logfile 2>&1
# Now, perform some clean-up of older files:
echo "Performing cleanup of older files under $backupdir." >> $logfile
find $backupdir -type f -mtime +7 -exec rm -vf {} \; >> $logfile 2>&1
# Provide a current file listing:
echo "Current contents of $backupdir:" >> $logfile
ls -lah $backupdir >> $logfile
# Finally, let's e-mail $logfile so someone can monitor this sucker:
echo "Stop time and system load averages: $(uptime)" >> $logfile
mailx -q $logfile -s "MySQL Backup on $(hostname) for $datestamp" $mailto < /dev/null
# And we're done!
```