# Shebang

Joe Conway
mail@joeconway.com

June 19, 2015

## What is Shebang?

- Definition: shebang
  - UNIX term, for `#!`
  - Contraction of "hash" (#), and "bang" (!)
- Focus here - `#!/bin/bash` with PostgreSQL

`http://www.urbandictionary.com/define.php?term=shebang&defid=301996`

## Agenda

- Pros/cons of shell scripts
- Overview
    - Function library
- PostgreSQL Specific Techniques
    - Executing SQL
    - Set/get PostgreSQL data from/into script variables
    - Keeping PostgreSQL functions in sync with scripts
- General Techniques
    - Locking
    - Doing work in parallel
    - Ensuring cleanup

## Pros

- Relatively easy learn
- Great for automating repetitive command line tasks
- Reasonably easy to debug
- Pervasive support in modern Posix systems
- Leverage huge ecosystem of tools

## Cons

- Not good for complex tasks
- Relatively slow
- Tend to be less robust
- Portability issues

## Common Functions Library

- Aggregate commonly used bash functions
- Prevent proliferation of redundant code
- Centralize for ease of testing and version control

## Common Functions Library: Header

```
set -u
set +x              # -x for debug
readonly DEBUG=0    #  1 for debug

# base name of outer script
if [[ "${0:0:1}" == "-" || "${0}" == "/bin/bash" ]]
then
  readonly BASENAME="loginshell"
  readonly BASEDIR="./"
else
  readonly BASENAME="$(basename $0)"
  readonly BASEDIR="$(readlink -m $(dirname $0))"
  set -e
fi
...
```

## Common Functions Library: Header

```
...
# location for any output files
readonly OUTDIR="${BASEDIR}/output"
# make sure it exists
mkdir -p ${OUTDIR}

# location for any sql files
readonly SQLDIR="${BASEDIR}/sql"
# make sure it exists
mkdir -p ${SQLDIR}

# Interlock ENUM
readonly UNLOCK=0
readonly NOWAIT=1
readonly BLOCK=2
...
```

## Common Functions Library: Header

```
...
# often useful for log output and filenames
NOW=$(date +"%Y%m%d-%H%M%S")

# to print all output to both log file and stdout
# change LOGOUTPUT to 1
LOGOUTPUT=0
if [[ -t 1 ]] && [[ LOGOUTPUT -eq 1 ]]
then
    OUTPUTLOG="${OUTDIR}/${BASENAME}_${NOW}.log"
    exec > >(tee -a "${OUTPUTLOG}")
    exec 2>&1
fi
...
```

## Common Functions Library: Body

```
...
# Description of my_func1
function my_func1
{
  local VAR1="$1"
  ...
}

# Description of my_func2
function my_func1
{
  local VAR1="$1"
  ...
}

...
```

## Common Functions Library: Usage

```
#!/bin/bash
# Description of this script

# resolve canonical script directory name
BASEDIR=$(readlink -m $(dirname $0))

# find the function library and use it
COMMON_LIB="${BASEDIR}/common.sh"
if [[ -r "${COMMON_LIB}" ]]; then
  source "${COMMON_LIB}"
else
  echo "ERROR: unable to source file ${COMMON_LIB}"
  exit 1
fi
...
```

## Executing SQL - String: Function

```
function exec_sql
{
    local PGHOST="$1"
    local PGPORT="$2"
    local DBNAME="$3"
    local PGUSER="$4"
    local DLM="$5"
    local SQL="$6"

    echo "${SQL}" | \
         psql -v ON_ERROR_STOP=1 \
         -qAt -F "${DLM}" \
         -h ${PGHOST} -p ${PGPORT} \
         -U ${PGUSER} -d ${DBNAME}
}
```

## Executing SQL - String: Explanation

```
echo "${SQL}" | psql -v ON_ERROR_STOP=1 -qAt -F "${DLM}" \
     -h ${PGHOST} -p ${PGPORT} -U ${PGUSER} -d ${DBNAME}
```

- Pipe SQL to psql versus `psql -c`
  - `psql -c` does not honor ON_ERROR_STOP
- `-v ON_ERROR_STOP=1`
  - Multi-statement SQL will stop on first ERROR
- `-qAt -F "${DLM}"`
  - Quiet, unaligned, tuples-only
  - Field separator set to `${DLM}`
- `-h ${PGHOST} -p ${PGPORT} -U ${PGUSER} -d ${DBNAME}`
  - Connection info

## Executing SQL - String: Usage

```
# see sqlstr.sh
### Load Common Functions Library ###
PGHOST="/tmp"; PGPORT="55605"; DBNAME="pgcon2015"
PGUSER="postgres"; DLM=" "

SQL="select pid, now() - state_change as age
     from pg_stat_activity
     where datname = current_database()
     and state = 'idle in transaction'"
exec_sql "${PGHOST}" "${PGPORT}" "${DBNAME}" \
         "${PGUSER}" "${DLM}" "${SQL}"
```

## Executing SQL - File: Function

```
function exec_sql_file ()
{
    local PGHOST="$1"
    local PGPORT="$2"
    local DBNAME="$3"
    local PGUSER="$4"
    local DLM="$5"
    local SQLFILE="$6"

    psql -v ON_ERROR_STOP=1 \
         -qAt -F "${DLM}" \
         -h ${PGHOST} -p ${PGPORT} \
         -U ${PGUSER} -d $DBNAME \
         -f "${SQLFILE}"
}
```

## Executing SQL - File: Explanation

```
psql -v ON_ERROR_STOP=1 -qAt -F "${DLM}" \
     -h ${PGHOST} -p ${PGPORT} -U ${PGUSER} -d $DBNAME \
     -f "${SQLFILE}"
```

- Same as string version except source from file
- Useful if dynamically generated SQL needs persistence
- Easier to avoid nested quote and shell expansion issues
- Convenient to manage SQL separately from script

## Executing SQL - File: Usage

```
# see sqlfile.sh
### Load Common Functions Library ###
PGHOST="/tmp"; PGPORT="55605"; DBNAME="pgcon2015"
PGUSER="postgres"; DLM=" "

SQLFILE="$(mktemp ${SQLDIR}/test.XXXXXX.sql)"

echo "select pid, now() - state_change as age
from pg_stat_activity
where datname = current_database()
and state = 'idle in transaction'" > "$SQLFILE"

exec_sql_file "${PGHOST}" "${PGPORT}" "${DBNAME}" \
              "${PGUSER}" "${DLM}" "${SQLFILE}"
```

## Assigning SQL Results: Scalar

```
# see sqlassign.sh
### Load Common Functions Library ###
PGHOST="/tmp"; PGPORT="55605"; DBNAME="pgcon2015"
PGUSER="postgres"; DLM=" "

SQL="select 42 as the_answer"

the_answer="$(exec_sql "${PGHOST}" "${PGPORT}" \
                "${DBNAME}" "${PGUSER}" \
                "${DLM}" "${SQL}")"

echo "The answer is: ${the_answer}"
```

## Assigning SQL Results: Set

```
# see sqlassign.sh
### Load Common Functions Library ###
PGHOST="/tmp"; PGPORT="55605"; DBNAME="pgcon2015"
PGUSER="postgres"; DLM=" "

SQL="select pid, now() - state_change as age
     from pg_stat_activity
     where datname = current_database()
     and state = 'idle in transaction'"

while read pid age
do
  echo "pid/age: ${pid}/${age}"
done <<< "$(exec_sql "${PGHOST}" "${PGPORT}" \
                     "${DBNAME}" "${PGUSER}" \
                     "${DLM}" "${SQL}")"
```

## Managing SQL Functions: sqlfunc_match

```
function sqlfunc_match
{
    local PGHOST="$1"; local PGPORT="$2"; local DBNAME="$3"
    local PGUSER="$4"; local DLM="$5"; local SCHEMA="$6"
    local FNAME="$7"; local ARGTYPS="$8"; local FUNCMD5="$9"
    local SQL="
        select count(1) from pg_catalog.pg_proc
        where pronamespace=(select oid from pg_catalog.pg_namespace
                            where nspname='${SCHEMA}')
        and proname='${FNAME}'
        and proargtypes = '${ARGTYPS}'::oidvector
        and md5(pg_get_functiondef(oid)) = '${FUNCMD5}'
    "

    echo $( exec_sql "${PGHOST}" "${PGPORT}" "${DBNAME}" \
                     "${PGUSER}" "${DLM}" "${SQL}" )
}
```

## Managing SQL Functions: Explanation

```
select count(1) from pg_catalog.pg_proc
where pronamespace=(select oid from pg_catalog.pg_namespace
                    where nspname='${SCHEMA}')
and proname='${FNAME}'
and proargtypes = '${ARGTYPS}'::oidvector
and md5(pg_get_functiondef(oid)) = '${FUNCMD5}'
```

- SCHEMA+FNAME+ARGTYPS $\Rightarrow$ function signature
- `md5(pg_get_functiondef(oid))` $\Rightarrow$ function version

## Managing SQL Functions: cr_sql_func

```
function cr_sql_func
{
  local PGHOST="$1"; local PGPORT="$2"; local DBNAME="$3"
  local PGUSER="$4"; local DLM="$5"; local SQLFILE="$6"
  local FQ_SQLFILE="${SQLDIR}/${SQLFILE}"

  local SCHEMA=$(cut -d "." -f 1 <<< ${SQLFILE})
  local FNAME=$((cut -d "." -f 2 <<< ${SQLFILE}) |\
                cut -d "-" -f 1)
  local ARGTYPS=$((cut -d "." -f 2 <<< ${SQLFILE}) |\
                  cut -d "-" -f 2 | tr '_' ' ')
  local FUNCMD5=$(md5sum ${FQ_SQLFILE} | cut -d " " -f 1)
  ...
```

## Managing SQL Functions: Explanation

```
local SCHEMA=$(cut -d "." -f 1 <<< ${SQLFILE})
local FNAME=$((cut -d "." -f 2 <<< ${SQLFILE}) |\
              cut -d "-" -f 1)
local ARGTYPS=$((cut -d "." -f 2 <<< ${SQLFILE}) |\
                cut -d "-" -f 2 | tr '_' ' ')
local FUNCMD5=$(md5sum ${FQ_SQLFILE} | cut -d " " -f 1)
```

- Parse SCHEMA+FNAME+ARGTYPS from SQLFILE basename
- Calculate MD5 of SQLFILE

## Managing SQL Functions: cr_sql_func

```
  ...
  funcmatch=$(sqlfunc_match "${PGHOST?}" "${PGPORT?}" \
                            "${DBNAME?}" "${PGUSER?}" \
                            "${DLM}" "${SCHEMA}" "${FNAME}" \
                            "${ARGTYPS}" "${FUNCMD5}" )
  if [[ $funcmatch -ne 1 ]]; then
    echo "Executing ${SQLFILE} to install/replace ${SCHEMA}.${FNAME}"
    sqlout=$(exec_sql_file "${PGHOST}" "${PGPORT}" "${DBNAME}" \
                           "${PGUSER}" "${DLM}" "${FQ_SQLFILE}")
  fi
}
```

## Managing SQL Functions: Explanation

```
funcmatch=$(sqlfunc_match "${PGHOST?}" "${PGPORT?}" \
                          "${DBNAME?}" "${PGUSER?}" \
                          "${DLM}" "${SCHEMA}" "${FNAME}" \
                          "${ARGTYPS}" "${FUNCMD5}" )
if [[ $funcmatch -ne 1 ]]; then
  ...
```

- SCHEMA+FNAME+ARGTYPS+MD5 match?
- If not, `CREATE OR REPLACE ...`

## Managing SQL Functions: Usage

```
CREATE OR REPLACE FUNCTION testfunc(arg int)
RETURNS text LANGUAGE sql AS $$
select 'The answer is ' || arg::text as the_answer
$$;
```

## Managing SQL Functions: Usage

```
# see sqlfdump.sh
### Load Common Functions Library ###
PGHOST="/tmp"; PGPORT="55605"; DBNAME="pgcon2015"
PGUSER="postgres"; DLM=" "

SCHEMA="$1"; FNAME="$2"; ARGTYPS="$3"; SQLFILE="$4"
SQL="select pg_get_functiondef(p.oid)
from pg_catalog.pg_proc p
join pg_catalog.pg_namespace n on n.oid = p.pronamespace
where nspname = '${SCHEMA}'
and proname = '${FNAME}'
and proargtypes = '${ARGTYPS}'"

FDEF=$(exec_sql "${PGHOST}" "${PGPORT}" "${DBNAME}" \
                "${PGUSER}" "${DLM}" "${SQL}")

echo "${FDEF}" > "${SQLDIR}/${SQLFILE}"
```

## Managing SQL Functions: Usage

```
# see sqlcheck.sh
### Load Common Functions Library ###
PGHOST="/tmp"; PGPORT="55605"; DBNAME="pgcon2015"
PGUSER="postgres"; DLM=" "

SQLFILE="public.testfunc-23.sql"

# verify the correct version of the SQL function is installed
cr_sql_func "${PGHOST}" "${PGPORT}" "${DBNAME}" \
            "${PGUSER}" "${DLM}" "${SQLFILE}"
# now use it
SQL="select public.testfunc(42)"
exec_sql "${PGHOST}" "${PGPORT}" "${DBNAME}" \
         "${PGUSER}" "${DLM}" "${SQL}"
```

## Concurrent Execution Interlock

- Ensure script only runs once at any time
- Especially important if script:
  - might run long
  - is executed from periodic cron job
  - concurrent execution might cause undue load or damage
- Examples:
  - Forcing table vacuum based on custom criteria
  - Maintaining data in external (non-pg) systems
  - Dependence on starting state

## Concurrent Execution Interlock

```
function interlock
{
  local LOCKNAME="$1"; local LOCKTYPE="$2"; set +e
  if [[ ${LOCKTYPE} -ne ${UNLOCK} ]]; then
    if [[ ${LOCKTYPE} -eq ${NOWAIT} ]]; then
      action="--nonblock"
    else
      action=""
    fi
    exec {FD}> "${LOCKNAME}"
    flock --exclusive ${action} ${FD}
    rv=$?
    if [[ $rv -ne 0 ]]; then
      echo "could not obtain lock - ${LOCKNAME}"
    fi
  else
  ...
```

## Concurrent Execution Interlock

```
    ...
    else
      flock --unlock ${FD}
      rv=$?
      if [[ $rv -ne 0 ]]
      then
        echo "could not unlock ${LOCKNAME}"
      fi
    fi
    return $rv
  }
```

## Concurrent Execution Interlock

```
# see flock.sh
### Load Common Functions Library ###
SLEEPTIME=10
LOCKFILE="${OUTDIR}/${BASENAME}.lock"
echo "Attempt to grab lock in non-blocking mode..."
interlock "${LOCKFILE}" ${NOWAIT}
rv=$?
if [[ $rv -ne 0 ]]
then
  echo "Attempt to grab lock in blocking mode..."
  interlock "${LOCKFILE}" ${BLOCK}
fi

echo "sleeping ${SLEEPTIME} seconds locked"
sleep ${SLEEPTIME}
echo "done ${SLEEPTIME} seconds"
```

## Parallel Work

- Launch multiple async processes and wait until completion
- Examples:
    - pg_dump across multiple servers
    - multiple long running queries
- Example script:
    - Three queries, each launched as separate background task
    - The `wait` command blocks until all three complete
    - Without parallelism, would take >= 15 seconds
    - With parallelism, takes ~ 5 seconds

## Parallel Work - Simple Method

- Cannot gather results easily
- Still useful for long autonomous tasks

```
### Load Common Functions Library ###
PGHOST_1="/tmp"; PGHOST_2="/tmp"; PGHOST_3="/tmp"
PGPORT="55605"; DBNAME="pgcon2015"; PGUSER="postgres"
DLM=" "; SLEEPTIME=5
SQL="select 42 as the_answer, pg_sleep(30)"

exec_sql "${PGHOST_1}" ... "${SQL}" &
exec_sql "${PGHOST_2}" ... "${SQL}" &
exec_sql "${PGHOST_3}" ... "${SQL}" &
wait
...
```

## Parallel Work - More Complex

- Able to gather results, but coproc has issues

```
# see parallel.sh
### Load Common Functions Library ###
PGHOST_1="/tmp"; PGHOST_2="/tmp"; PGHOST_3="/tmp"
PGPORT="55605"; DBNAME="pgcon2015"; PGUSER="postgres"
DLM=" "; SLEEPTIME=5
SQL1="select 42 as the_answer, pg_sleep(${SLEEPTIME})"
SQL2="select 43 as the_answer, pg_sleep(${SLEEPTIME})"
SQL3="select 44 as the_answer, pg_sleep(${SLEEPTIME})"
coproc p1 { exec_sql "${PGHOST_1}" ... "${SQL1}" ; }
coproc p2 { exec_sql "${PGHOST_2}" ... "${SQL2}" ; }
coproc p3 { exec_sql "${PGHOST_3}" ... "${SQL3}" ; }
coproc p4 { sleep 1 ; }  #work around coproc bug
wait
read buf1 <&${p1}; echo "buf1=${buf1}"
read buf2 <&${p2}; echo "buf2=${buf2}"
read buf3 <&${p3}; echo "buf3=${buf3}"
```

## Parallel Work - More Complex

- Another coproc issue

```
time ./parallel.sh
./parallel.sh: line 29: warning: execute_coproc: ...
./parallel.sh: line 30: warning: execute_coproc: ...
./parallel.sh: line 31: warning: execute_coproc: ...
buf1=42
buf2=43
buf3=44

real    0m5.083s
user    0m0.106s
sys     0m0.063s
```

## Automated Cleanup

- Register trap
- Examples:
  - Undo partial changes after error or SIGINT (Ctl-c)
  - Clean up on any EXIT, e.g. temporary files
  - Keep track of progress/provide ability to restart
- Example script:
  - Set undo traps
    $\rightarrow$ TRUNCATE t1; INSERT INTO t2;
  - Perform some DML
    $\rightarrow$ INSERT INTO t1; TRUNCATE t2;
  - On ERROR or SIGINT, trap runs
  - On success, trap does not run

## Automated Cleanup

```
# see trap.sh
### Load Common Functions Library ###
PGHOST="/tmp"; PGPORT="55605"; DBNAME="pgcon2015"
PGUSER="postgres"; DLM=" "

if [[ $# -eq 1 ]]
then THROWERR="1"
else THROWERR="0"
fi

function clean_up
{
    for (( i=${#ARRAY[@]} - 1; i>=0; i-- )); do
        eval ${ARRAY[$i]}
    done; exit 1
}
...
```

## Automated Cleanup

```
...
declare -a ARRAY
SQL="truncate table t1"
ELEMENT="echo 'exit trap: truncating table t1';
         exec_sql \"${PGHOST}\" \"${PGPORT}\" \"${DBNAME}\" \
                  \"${PGUSER}\" \"${DLM}\" \"${SQL}\""
ARRAY[0]="${ELEMENT}"

SQL="insert into t2 values(1),(2),(3),(4)"
ELEMENT="echo 'exit trap: inserting into table t2';
         exec_sql \"${PGHOST}\" \"${PGPORT}\" \"${DBNAME}\" \
                  \"${PGUSER}\" \"${DLM}\" \"${SQL}\""
ARRAY[${#ARRAY[@]}]="${ELEMENT}"

trap 'clean_up' SIGINT ERR
...
```

## Automated Cleanup

```
...
SQL="insert into t1 values(1),(2),(3)"
echo "script: inserting into table t1"
exec_sql "${PGHOST}" "${PGPORT}" "${DBNAME}" \
         "${PGUSER}" "${DLM}" "${SQL}"
SQL="truncate table t2"
echo "script: truncating table t2"
exec_sql "${PGHOST}" "${PGPORT}" "${DBNAME}" \
         "${PGUSER}" "${DLM}" "${SQL}"
SQL="select count(1) from t1"
echo "t1 count: $(exec_sql "${PGHOST}" "${PGPORT}" \
         "${DBNAME}" "${PGUSER}" "${DLM}" "${SQL}")"
SQL="select count(1) from t2"
echo "t2 count: $(exec_sql "${PGHOST}" "${PGPORT}" \
         "${DBNAME}" "${PGUSER}" "${DLM}" "${SQL}")"
...
```

## Automated Cleanup

```
...
if [[ $THROWERR -eq 1 ]]
then
    echo "ctl-c to get SIGINT, after 5 seconds force error"
    sleep 5
    false
else
    echo "Completed successfully"
fi
```

## Automated Cleanup

```
./trap.sh err
script: inserting into table t1
script: truncating table t2
t1 count: 3
t2 count: 0
ctl-c to get SIGINT, after 5 seconds force error
exit trap: inserting into table t2
exit trap: truncating table t1

echo "select (select count(1) from t1) as t1,
             (select count(1) from t2) as t2" | \
     psql -h /tmp -p 55605 -U postgres -d pgcon2015
 t1 | t2
----+----
  0 |  4
(1 row)
```

## Automated Cleanup

```
./trap.sh
script: inserting into table t1
script: truncating table t2
t1 count: 3
t2 count: 0
Completed successfully

echo "select (select count(1) from t1) as t1,
             (select count(1) from t2) as t2" | \
      psql -h /tmp -p 55605 -U postgres -d pgcon2015
 t1 | t2
----+----
  3 |  0
(1 row)
```

## Questions?

Thank You!
mail@joeconway.com