

**Converting rc scripts for 10.x****Dillon Pyron****Advanced Micro Devices, Inc.****6800 Burleson Rd, Bldg 312 M/S 606****Austin, TX 78741****512 602 2368**

Until it starts, a computer is just a bunch of expensive components, in a neat little box. And the OS is just magnetic anomalies on a disk. So, startup is a critical part of a computer's life. In Unix, the startup is arranged so that we can control various portions of the software configuration and how it will deal with the external hardware. In previous releases of HP-UX, this was handled by a script known as rc, which contained significant portions of the startup code, and called other related scripts for specialized functions such as networking.

Under HP-UX 10.x, however, this structure has changed. As HP has embraced the SVR4 model more closely, we find a startup with more structure, modularity and maintainability. As well as one that is clearer. Gone is the monolithic rc. In its place is a trim rc script that calls execution scripts as needed. The execution scripts each have their own config files. By editing the config files, the sys admin can control the behavior of the execution scripts, without having to edit (or understand) the scripts. More important, new software can be "plugged in" without perturbing existing installations.

The rc we all knew and despised is gone. And the new rc is in a new location. All of the startup scripts can be found under the /sbin directory. In fact, during startup, /sbin is one of the few directories that can be guaranteed to be available. Also of interest in /sbin are the directories rc{0-4}.d and init.d. The rc{0-4}.d directories are called "link" directories, because the files in them are actually links to the execution scripts, which are located in init.d. (Fig. 1)

rc2.d:

```
lrwxr-xr-x 1 root sys      16 Jul 22 1996 S008net.sd -> /sbin/init.d/net
lrwxr-xr-x 1 root sys      21 Jul 22 1996 S100swagentd -> /sbin/init.d/swagentd
lrwxr-xr-x 1 root sys      21 Jul 22 1996 S120swconfig -> /sbin/init.d/swconfig
lrwxr-xr-x 1 root sys      21 Jul 22 1996 S200clean_ex -> /sbin/init.d/clean_ex
lrwxr-xr-x 1 root sys      23 Jul 22 1996 S202clean_uucp -> /sbin/init.d/clean_uucp
lrwxr-xr-x 1 root sys      23 Jul 22 1996 S204clean_tmpps -> /sbin/init.d/clean_tmpps
lrwxr-xr-x 1 root sys      22 Jul 22 1996 S206clean_adm -> /sbin/init.d/clean_adm
lrwxr-xr-x 1 root sys      20 Jul 22 1996 S220syslogd -> /sbin/init.d/syslogd
lrwxr-xr-x 1 root sys      22 Jul 22 1996 S230ptydaemon -> /sbin/init.d/ptydaemon
lrwxr-xr-x 1 root sys      18 Jul 22 1996 S300nettl -> /sbin/init.d/nettl
lrwxr-xr-x 1 root sys      20 Jul 22 1996 S320hpether -> /sbin/init.d/hpether
lrwxr-xr-x 1 root sys      16 Jul 22 1996 S340net -> /sbin/init.d/net
lrwxr-xr-x 1 root sys      18 Jul 22 1996 S370named -> /sbin/init.d/named
lrwxr-xr-x 1 root sys      21 Jul 22 1996 S400nfs.core -> /sbin/init.d/nfs.core
lrwxr-xr-x 1 root sys      23 Jul 22 1996 S410nis.server -> /sbin/init.d/nis.server
lrwxr-xr-x 1 root sys      23 Jul 22 1996 S420nis.client -> /sbin/init.d/nis.client
lrwxr-xr-x 1 root sys      23 Jul 22 1996 S430nfs.client -> /sbin/init.d/nfs.client
```

Fig. 1

The link scripts have a simple naming convention that must be observed to make things function properly. Within each run level (as denoted by the directory), lower number scripts are executed first on startup, and last on shutdown. After this comes the subsystem name, which is usually the same as the execution script name. This is seen in Fig. 1. Which scripts are run are startup and shutdown is discussed below.

At this point, most readers are starting to watch the room spin, so let's take a quick look at a skeletal startup. The purpose here is not to describe a startup in detail, but rather, to list the elements of one for further discussion.

In /etc/inittab, there is an entry for the sequencer (/sbin/rc):

```
sqnc::wait:/sbin/rc </dev/console >/dev/console 2>&1      # system init
```

At this point, /sbin/rc starts up and determines its environment. We are assuming system startup, although it can also be called for transitions between run-levels on a running system. It identifies the target level, and collects a list of all of the scripts to run. It is at this point that the checklist is printed. (Fig 2) Now rc starts executing the scripts. It does this in order of level, and then based on sequence number inside each level. For instance, syncer (in rc1.d) is started before nis client (in rc2.d). (Fig 3) And, nis client is started before nfs client, due to sequence numbers. (Fig 3)

### HP-UX Start-up in progress

---

```
Mount file systems . . . . . [ OK ]
Setting hostname . . . . . [ OK ]
Set privilege group . . . . . [N/A ]
.
.
.
Start the snoopy daemon . . . . . [Fail]
```

Fig. 2

The sequence numbers have a particular form and function. Each component has an entry in the appropriate run level directory, with either a start (S) or kill (K) indicator, followed by a number which indicates its relative position in the start/stop process. In the above example, nis client is S420nis.client and nfs client is S430nfs.client. (Fig 3) So nis client is executed first.

As each component is executed in sequence, rc changes the status in the checklist display to reflect state of the execution script. This status consists of OK (in green on color monitors), FAIL (in red), N/A (in yellow) and an alternating BUSY/WAIT (also in yellow). When a script completes, it returns a value 0 to indicate success, a 1 to indicate a failure and a 2 to indicate that the script skipped execution.

Now that we have a basic understanding of the mechanics, let's take a look at how to write a functional, compliant startup script for your own component. As stated above, the entries in the rc\*.d directories are just links to the actual execution scripts in init.d. This execution script needs to handle one of four input parameters, start\_msg, start, stop\_msg and stop. Of these, start\_msg and stop\_msg simply print a message (the checklist display) to the console. Notice that, if there is not an entry to start (S) or kill (K) a component, then there will not be a checklist display for that component.

rc1.d:				
K186pv	K380xf	K440SnmpMaster	K580nis.client	S100hfsmount
K230audio	K390bootd	K450ddfa	K590nis.server	S320hostname
K240auditing	K400i4lmd	K460sendmail	K600nfs.core	S400set_prvgrp
K270cron	K410nfs	K470rwhod	K630named	S420set_date
K280lp	K420dfs	K480rpd	K660net	S440savecore
K290hparray	K430dce	K490gated	K700nettl	S500swap_start
K300acct	K435OspfMib	K500inetd	K770ptydaemon	<u>S520syncer</u>
K340xntpd	K435SnmpHpunix	K510mouted	K780syslogd	
K370vt	K435SnmpMib2	K570nfs.client	K900swagentd	
rc2.d:				
K100dtlogin.rc	S300nettl	S510gated	S590nfs	S760auditing
K900nfs.server	S320hpether	S520rpd	S600i4lmd	S770audio
S008net.sd	S340net	S530rwhod	S610rbootd	S800spa
S100swagentd	S370named	S540sendmail	S620xf	S814pv
S120swconfig	S400nfs.core	S550ddfa	S630vt	S820prm
S200clean_ex	S410nis.server	S560SnmpMaster	S660xntpd	S880swcluster
S202clean_uucp	<u>S420nis.client</u>	S565OspfMib	S700acct	
S204clean_tmpps	<u>S430nfs.client</u>	S565SnmpHpunix	S710hparray	
S206clean_adm	S431amd.client	S565SnmpMib2	S720lp	
S220syslogd	S490mouted	S570dce	S730cron	
S230ptydaemon	S500inetd	S580dfs	S740suprtinfo	

Fig. 3

HP generously supplies us with a template file in /sbin/init.d/template. This provides all of the generic utilities used by most of the standard execution scripts, as well as a workable structure. The user is responsible for filling in “the guts”, but this is typically the easiest part, once the boilerplate required to interface with startup is covered.

There are some rules, or guidelines, to writing your script. Never assume too much. Do not expect filesystems such as /usr to be mounted. Don’t write messages to the console. This will mess up the checklist display. Instead, write to stdout and stderr.

One of the additional benefits of this system is that we no longer need to edit the script to modify its behavior. Instead, the script is affected by variables in config files for each of the execution scripts. To, for instance, turn off nfs client mode, set the variable NFS\_CLIENT to 0 in /etc/rc.config.d/nfsconf. (Fig. 4) This is not only easier, but safer, the trying to edit a script and get all of the correct lines. In fact, HP recommends NOT editing the scripts.

```
#
NFS_CLIENT=1
NFS_SERVER=1
NUM_NFSD=4
NUM_NFSIOD=4
#
```

Fig. 4

The config files can be sourced in one of two methods. The execution script can source in just its files, or it can source in /etc/rc.config, which then sources all of the config files. The second method is recommended for applications, as it guarantees a fully defined environment. The config files are executed by a POSIX shell, and must be written with that in mind.

To write you own scripts, you will need sequence numbers. For general purpose usage, HP has assigned /sbin/rc2.d/S900xxxxx for startup scripts, and /sbin/rc1.d/K100xxxxx for shutdown scripts. HP may also use these same numbers, so use very distinctive names.

With these points in mind, the following is a trivial example, which starts an application called snoopy.

```
#!/sbin/sh
#
# @(#) $Revision: 72.11 $
#
# NOTE: This script is not configurable! Any changes made to this
# script will be overwritten when you upgrade to the next
# release of HP-UX.
#
# WARNING: Changing this script in any way may lead to a system that
# is unbootable. Do not modify this script.

#
# This will start the snoopy daemon Put your comments here.
#

# Allowed exit values:
# 0 = success; causes "OK" to show up in checklist.
# 1 = failure; causes "FAIL" to show up in checklist.
# 2 = skip; causes "N/A" to show up in the checklist.
# Use this value if execution of this script is overridden
# by the use of a control variable, or if this script is not
# appropriate to execute for some other reason.
# 3 = reboot; causes the system to be rebooted after execution.

# Input and output:
# stdin is redirected from /dev/null
#
# stdout and stderr are redirected to the /etc/rc.log file
# during checklist mode, or to the console in raw mode.

PATH=/usr/sbin:/usr/bin:/sbin
export PATH

# NOTE: If your script executes in run state 0 or state 1, then /usr might
# not be available. Do not attempt to access commands or files in
# /usr unless your script executes in run state 2 or greater. Other
# file systems typically not mounted until run state 2 include /var
# and /opt.

rval=0

# Check the exit value of a command run by this script. If non-zero, the
# exit code is echoed to the log file and the return value of this script
# is set to indicate failure.

set_return() {
    x=$?
    if [ $x -ne 0 ]; then
        echo "EXIT CODE: $x"
        rval=1 # script FAILED
    fi
}
```

```

}

# Kill the named process(es).
# $1=<search pattern for your process>

killproc() {
    pid=`ps -e | awk '$NF~/\"$1\"/ {print $1}`
    if [ "X$pid" != "X" ]; then
        if kill "$pid"; then
            echo "$1 stopped"
        else
            rval=1
            echo "Unable to stop $1"
        fi
    fi
}

case $1 in
'start_msg')
    # Emit a _short_ message relating to running this script with
    # the "start" argument; this message appears as part of the checklist.
    echo "Start the snoopy daemon" This is the message echoed to the console
    ;;

'stop_msg')
    # Emit a _short_ message relating to running this script with
    # the "stop" argument; this message appears as part of the checklist.
    echo "Stopping the snoopy subsystem"
    ;;

'start')
    # source the system configuration variables
    if [ -f /etc/rc.config ] ; then
        . /etc/rc.config
    else
        echo "ERROR: /etc/rc.config defaults file MISSING"
    fi

    # Check to see if this script is allowed to run...
    if [ "$SNOOPY_RUN" != 1 ]; then Set in the snoopy config file
        rval=2
    else
        killproc snoopyd take advantage of provided utilities
    #make sure the daemon isn't already running

    /sbin/naughty/snoopy Most of the work is already done, just start the thing!
    set_return and return a value of (hopefully) 0 for success.

    fi
    ;;

```

```

'stop')
    # source the system configuration variables
    if [ -f /etc/rc.config ] ; then
        . /etc/rc.config
    else
        echo "ERROR: /etc/rc.config defaults file MISSING"
    fi

    # Check to see if this script is allowed to run...
    if [ "$SNOOPY_RUN" != 1 ]; then
        rval=2
    else
        :
        killproc snoopyd
        # we don't care if it really succeeds, we're just trying to be nice

    fi
    ;;

*)
    echo "usage: $0 {start|stop|start_msg|stop_msg}"
    rval=1
    ;;
esac

exit $rval

```

As a note to the observant, the above is an example and would not likely run due to some scripting problems.

The config file for snoopy, residing in /etc/rc.config.d, look like this:

```

# ***** File: /etc/rc.config.d/snoopy *****
#
# SNOOPY_RUN: Set to 1 to start snoopy daemon
#

```

*SNOOPY\_RUN=1 This value tells the script to run it*

If you are planning on writing scripts that will be used by external (hopefully paying) customers, you can request a unique startup sequence number from HP. This will also be of value if a) you need to start prior to run level 3 or b) you need to start prior to other subsystems. You can contact the PA-RISC Developer's Program at [pard@apollo.hp.com](mailto:pard@apollo.hp.com).

By following a straightforward set of rules, and taking advantage of the tools and templates provided, almost all of the conversion work is done, and the resulting scripts will be easier to manage and use. At the same time, however, you may continue using 9.0x scripts, wrapped in a simple script. This will allow you to make use of existing scripts, and convert them as the opportunity arises.

Converting rc scripts for 10.x  
2160-7