TechNews Letter

July 2014 | Vol 3 | UNIX

From the Architect's Desk

Zoheb Shaikh

Technology Architect
10.5 years of experience in Data Warehousing & Automation
Specializations: Data Warehousing, Hadoop HDFS, Informatica, Microstrategy, IBM CDC, Enterprise Schedulers

Tell us a little bit about yourself

For almost my entire professional career I have been with Infy, and since my first project I have been associated with Data Warehouses.

I love problem solving, it comes naturally to me, even when someone is explaining some issue my mind keep giving one solution after another J. My interests are in automations and reporting/dashboards. In my free time I like to spend some time with my DSLR, I love taking landscape and star trails pictures.

How do you perceive the role of Unix/Linux in Cloud Platforms?

Linux might have missed the desktop adoption but when it comes to big corporate data center it is the hands down the default OS. And the same has been the case with cloud backend servers. The reliability, scalability and economic benefits outshine its competitors.

RHEL 7's new Container format which is integrated into the operating system also helps users build a "layered" cloud application that can be moved across physical hosts which would be an awesome feature for the Cloud.

Red hat has just launched the RHEL7 – the latest version of Linux. What are the new features that we can look forward to in the new OS version?

Some of the key features that I look forward to in the new release

- *Upgrades in the File System, the new XFS (default from previous ext4) can now support 500TB disk sizes. Also the old ext4 can now support 50TB. This would help further in Big Data adoptions for RHEL.*
- *Support for 40GB Ethernet Link speed Data -* ***Hadoop Core, Beehive and Pig*** *and take up Technical certifications.*
- *Improvements in time keeping and global synchronization.*

Any recent technical challenge you faced and your learning from it?

While I installed Hadoop 2.0 HDFS on a RHEL system, there were a lot of UNIX administration activities that I was not aware of. So learning and solving issues on the system is more of an on-going activity as of now. BTW this system is available for anyone who wants to practice Hadoop map-reduce jobs on HDFS

What are your technology goals?

In my new role I plan to understand all the internal workings and limitation that a data warehouse architect should be aware while designing a warehouse. I am particularly interested in MPP databases and their leverage in answering complex corporate discussions. I am also interested in the BI layer's Dashboards, designing a next generation interactive dashboards is something I am currently working on.

How do you keep yourself abreast with latest in Technology?

I have a very long commute to work and most of that time I spend listen to podcasts. One of my favorites is NPR's "TED Radio Hour" these are talks about the mind blowing researches that are happing around the world. Other times I read online forums and rumors websites.

Your advice to fellow professionals.

Understand your strengths and weaknesses; this is very important that you should act accordingly. If you need help talk to your managers/friends and discuss with them to get their perception about you. Its only after that will you be able to understand which field you should strive to progress in.

90% of the world's most powerful supercomputers are using **GNU/Linux** (including the top ten). **33.8%** of the world runs on **Linux** servers compared to **7.3%** running **Microsoft Windows** operating system.

CRON Jobs in UNIX

Shilpa Borele
Technology Lead – SOA Dev
8 years of experience in Java/J2EE development

The **CRON** utility is a time-based job scheduler in Unix-like computer operating systems. It is used to schedule jobs in terms of **commands** or **shell scripts** to run periodically at fixed times, dates, or intervals.

Crontab is a configuration file that specifies shell commands to run periodically on a given schedule. Each line of a crontab file represents a job which is composed of a CRON expression, followed by a shell command or script to be executed.

For commands that need to be executed repeatedly you need to use crontab. Each entry in a **crontab** file consists of six fields, specified in the following order:

Crontab File- syntax:

```
minute(s) hour(s) day(s) month(s) weekday(s)
command(s)
```

Where, details to be given for command sequence executes

- minute 0-59 - the exact minute
- hour 0-23 - hour of the day
- day 1-31 - day of the month
- month 1-12 - month of the year
- weekday 0-6 - day of the week; Sunday=0, Monday = 1, Tuesday = 2, and so forth.
- command - The complete command sequence variable that is to be executed.

Crontab Example:

```
0 0 1,15 * 1 /big/dom/xdomain/cgi-
bin/scriptname.cgi
```

*The cron job would run the program **scriptname.cgi** in the cgi-bin directory on the 1st and 15th of each month, as well as on every Monday.*

To schedule one-time only tasks with cron we can use *at* or *batch*. For eg. the below job will run at noon the same day if submitted.

```
at noon
tar -cf /users/dvader dvader.tar
Ctrl-d
```

The job will run at noon the same day if submitted in the in the morning, or noon the next day if submitted in the afternoon. When the task is performed, a tarball of the /users/dvader directory will be created.

Crontab Commands:

- crontab filename (Install filename as crontab file)
- crontab -e (Edit crontab file, or create one if it doesn't already exist)
- crontab -l (Display crontab file)
- crontab -r (Remove crontab file)

Crontab Environment:

Cron invokes the command from the user's HOME directory with the shell, (/usr/bin/sh).

Cron supplies a default environment for every shell, defining:

```
HOME=user's-home-directory
LOGNAME=user's-login-id
PATH=/usr/bin:/usr/sbin:.
SHELL=/usr/bin/sh
```

Crontab Restrictions:

- **Cron.allow**: User can execute crontab if his name is present in /usr/lib/cron/cron.allow or this file doesn't exist.
- **Cron.deny**: User can execute crontab if his name is not present in cron.deny file.
- If cron.deny doesn't exist and is empty then all users can use crontab.
- If neither of these files exist only root user can user crontab.

Disable Email:

By default cron jobs sends an email to the user account executing the cronjob. If this is not needed put the following command at the end of the cron job line.

```
>/dev/null 2>&1
```

Generate log file:

To collect the cron execution log in a file:

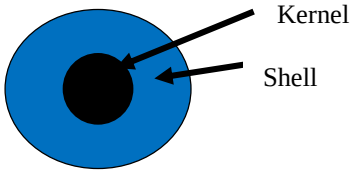
```
30 18 * * * rm /home/someuser/tmp/*
>
/home/someuser/cronlogs/clean_tmp_dir.log
```

Linux Kernel was written by a 21 year Finnish college student as a part of his hobby. Yup! His name is Linus Torvalds.

Shell Script in UNIX

Simranjeet Singh
Technology Analyst
4.8 years of experience in Java/J2EE development

What is Shell?
Shell is a command language interpreter which accepts the instruction from the user and executes them if they are valid instruction. Shell is not a part of the Kernel but uses kernel to execute the commands/instruction. It's an interface between the User and the Kernel.



History of Shell
Shell was developed over the time and it has four generation and its development is still going on.

- ***First Generation Shell:*** Thomson shell and Mashey shell were the first generation shells. The main feature of the Thomson Shell was the pipe feature wherein the output of the first command can be passed onto the second command. If or goto commands were not a part of this generation.
- ***Second Generation Shell:*** Bourne and C Shell are the second generation Shell which significantly improved the first generation shell. Switch, for, if and variable substitution was introduced in the Second Generation shell.
- ***Third Generation Shell:*** TCSH (extension of C-Shell), Korn Shell and POSIX shell were part of the third generation of the Shell. Korn shell is backward compatible with Bourne Shell. Korn shell introduced various attributes like read-only, uppercase, lowercase attributes for Strings. Korn Shell is licensed and hence its use was limited to commercial environment.
- ***Fourth Generation Shell:*** Z-Shell and Ksh93 were the shell introduced in the Fourth generation which were majorly influenced by Per. Z-Shell was created by merging the features of C-Shell and Korn Shell. Ksh93 is the most powerful shell for UNIX and can be considered the superset of POSIX.

What is a Shell Script?
Shell Script is a program which contains set of instructions/commands which are executed sequentially.

How to write a Shell Script?

- Create a script file using the vi editor or any other editor.

```
#! /bin/bash - Choose the shell
# Hash is used for comments
echo "Hello World!!!"
exit 1
```

- Modify the permissions of the script as shown below :

```
$ chmod +x your-script-name
$ chmod 755 your-script-name
```

- Execute the script as shown below

```
$ bash your-script-name
        Or
$ sh your-script-name
        Or
$ ./your-script-name
```

Functions in Shell Script
To increase the readability of a shell script functions can be added in the script. Below are examples of adding functions in the shell script.

```
#!/bin/sh
# Define function here
MyFunction () {
  echo "Hello World $1 $2"
}
# Invoke your function – Passing parameters to the function
MyFunction Simranjeet Singh.
# calling the function.
```

Executing the above script would produce the following output:

```
$./functionexample.sh
Hello World Simranjeet Singh
```

References
http://www.freeos.com/guides/lsst/
http://en.wikipedia.org/wiki/Shell_script

AWK Utility - Data Processing in file

Vivek Agrawal
Technology Analyst
5.8 years of experience in Oracle Apps

AWK is an interpreted programming language designed for text processing and typically used as a data extraction and reporting tool. It is a standard feature of most Unix-like operating systems.

AWK treats a file as a sequence of records, and by default each line is a record. Each line is broken up into a sequence of fields. An AWK program is of a sequence of pattern-action statements. A line is scanned for each pattern in the program, and for each pattern that matches, the associated action is executed.

The essential organization of an AWK program follows the form:

AWK String Functions - *pattern { action }*	
Name	**Variant**
index(string,search)	asort(string,[d])
length(string)	asorti(string,[d])
split(string,array,separator)	gensub(r,s,h [,t])
substr(string,position)	strtonum(string)
substr(string,position,max)	match(string,regex)
sub(regex,replacement)	tolower(string)
sub(regex,replacement,string)	toupper(string)

Below are some examples used for data formating in file which is helpful for preparing data files for **SQL LOADER**

1. **Convert Windows/DOS newlines (CRLF) to Unix newlines (LF) from Unix**

```
awk '{ sub(/\r$/,""); print }'
```

- This one-liner uses the *sub(regex, repl, [string])* function. This function substitutes the first instance of regular expression "regex" in string "string" with the string "repl". If "string" is omitted, variable $0 is used. Variable $0 contains entire line.

The one-liner replaces '\r' (CR) character at the end of the line with nothing, i.e., erases CR at the end. Print statement prints out the line and appends ORS variable, which is '\n' by default.

2. **Print and sort the login names of all users**

- This is the first time we see the *-F* argument passed to AWK. This argument specifies a character, a string or a regular expression that will be used to split the line into fields contains entire line.

- */etc/passwd* is a text file that contains a list of the system's accounts, along with some useful information like login name, user ID, group ID, home directory, shell, etc. The entries in the file are separated by a colon ":".

```
awk -F ":" '{ print $1 | "sort" }'
/etc/passwd
```

- The one-liner does just that - it splits the line on ":", then forks the "*sort*" program and feeds it all the usernames, one by one. After AWK has finished processing the input, sort program sorts the usernames and outputs them.

3. **Remove duplicate, nonconsecutive lines**

```
awk '!a[$0]++'
```

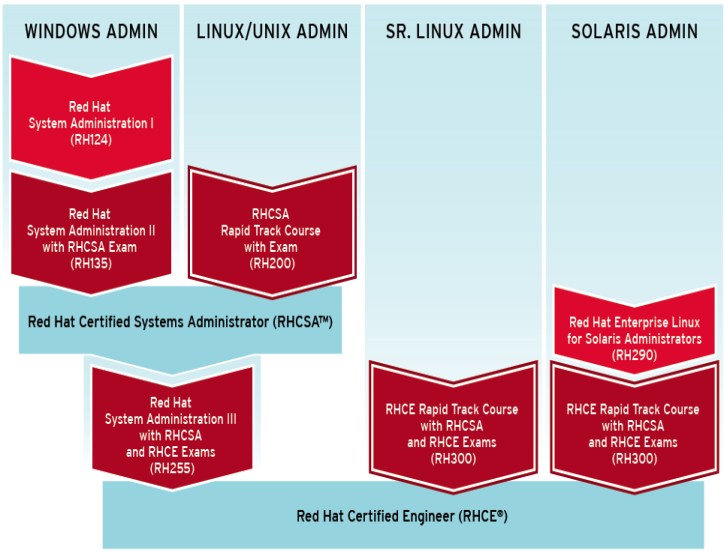
- This one-liner is very idiomatic. It registers the lines seen in the associative-array "*a*" (arrays are always associative in AWK).
- For example, suppose the input is:
  foo
  bar
  foo
- When AWK sees the first "foo", it evaluates the expression "*!a["foo"]++*". "a["foo"]" is false, but "!a["foo"]" is true - AWK prints out "foo". Then it increments "a["foo"]" by one with "++" post-increment operator. Array "a" now contains one value "a["foo"] == 1".
- Next AWK sees "bar", it does exactly the same what it did to "foo" and prints out "bar". Array "a" now contains two values "a["foo"] == 1" and "a["bar"] == 1".

Now AWK sees the second "foo". This time "a["foo"]" is true, "!a["foo"]" is false and AWK does not print anything! Array "a" still contains two values "a["foo"] == 2" and "a["bar"] == 1".

Android operating system is primarily based off of Linux kernel and Google has made several changes to make it go above and beyond the original basis of Linux kernel.

Certifications & Training

Red Hat Linux Certification Track



Oracle Solaris Certification Track

Training @ Infosys

Introduction to Unix	Introduction to Perl Scripting	Introduction to UNIX Shell Scripting
21 Jul - 22 Jul	31 Jul - 05 Aug	11 Aug - 13 Aug
VirtualClassroom	Classroom, Pune	Classroom, Pune

Technical Architects

Indranil Dharap

Core skills-
- Oracle Database
- Big Data
- Mongodb
- Pentaho Integrator

Contact details-
8378971004
Indranil_dharap@infosys.com

Zoheb Shaikh

Core skills-
- Data Warehousing
- Informatica
- Microstrategy
- IBM CDC, Schedulers
- Hadoop HDFS

Contact details-
8600152688
Zoheb_shaikh@infosys.com

Srinivasan Duraiswamy

Core skills-
- J2EE
- Application Integration
- SOAP based WebServices
- Oracle, Websphere

Contact details-
8095078962
Srinivasan_D@infosys.com

Android operating system is primarily based off of Linux kernel and Google has made several changes to make it go above and beyond the original basis of Linux kernel.

Quiz & Facts

UNIX Quiz

1. How do you get help about the command "cp"?

a) *help cp*
b) *man cp*
c) *cp ?*

2. How do you list 'all' the files that are in the current directory?

a) *list all*
b) *ls -full*
c) *ls –a*

3. How to create a new file "new.txt" that is a concatenation of "file1.txt" and "file2.txt"?

a) *cat file1.txt file2.txt > new.txt*
b) *make new.txt=file1.txt+file2.txt*
c) *tail file1.txt | head file2.txt > new.txt*

4. How do you visualize the content of file "not_empty"?

a) *type not_empty*
b) *cat not_empty*
c) *more not_empty*

5. How do you create a new directory called "flower"?

a) *newdir flower*
b) *mkdir flower*
c) *crdir flower*

6. What is the command to search all files in your current directory for the word "plasmodium"?

a) *grep plasmodium **
b) *find plasmodium -all*
c) *lookup plasmodium **

7. How do you print the first 15 lines of all files ending by ".txt"?

a) *print 15 .txt*
b) *cat * .txt -length = 15*
c) *head -15 *.txt*

8. How do you uncompress and untar an archive called "lot_of_thing.tar.Z"?

a) *tar lot_of_thing.tar.Z | decomp*
b) *zcat lot_of_thing.tar.Z | tar xvf -*
c) *tar xvf lot_of_thing.tar.Z*

(Verify your answers on last page)

$ Up/Down Arrows: The up and down arrows on your keyboard move through your last used commands. So, if you wanted to run the second to last command you ran, just hit the up arrow twice and hit Enter. You can also edit the command before you run it.

$ Tab: One of everyone's favorite shortcuts employs Tab to autocomplete a line of text. So, say you wanted to type of ~/Dropbox/, you could just type cd ~/Dr, hit Tab to autocomplete opbox, and continue on with your day.

$ Ctrl+left and Ctrl+right: Hitting Ctrl and the left or right arrow keys jumps between arguments in your command. So, if you had a typo in the middle of the command, you could jump to it quickly with Ctrl and a few taps of the left arrow key.

Case Study of Named Pipes to the Rescue

Ravikant Tiwari
Technology Lead – EIM Stability
8 years of experience in Development and Production Support

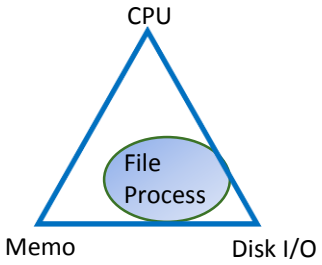
"The man with insight enough to admit his limitations comes nearest to perfection."
--Johann Wolfgang von Goethe

Our case study is about understanding your system limitations and making design changes to work around it.

Problem Statement
Raw Data file are provided to our warehouse in vendor format. It contains some unwanted columns and some columns that need to be added as well as validating of other fields before data is loaded in the database.

With increasing data volumes, file size has also been growing. System processes that were designed earlier used to process these files line by line and there were different steps involved. At every step physical file were being created on the disks. This was causing the processing time to increase exponentially.

When we analyzed the process it showed that there was a lot of disk reliance. And the effective I/O speeds that were achieved were below par. This was making the overall process to slowdown. On the CPU, Memory and Disk IO triangle this process can be mapped as below relying more on disk I/O



The task therefore was to see how we could improve the system performance and still work around the system limitation of a slow disk I/O.

Strategy
As discussed before the problem area identified was the intermediate physical files that were created on the disks. So the typical flow was -

```
cat input_file.dat | Some_Process1 > temp1.dat
cat temp1.dat | Some_Process2 > temp2.dat
cat temp2.dat | Some_Process3 > temp3.dat
cat temp3.dat | Some_Process4 > final_file.dat
```

It was decided to use named pipes instead of creating intermediate files and reducing/removing the disk I/O from the equation. In this way we were able to reduce I/O and increase use of Memory and CPU.

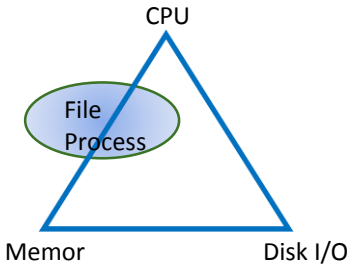
"A named pipe is a special kind of file (eg: FIFO file) on the drive. Unlike a regular file, a FIFO file does not contain any user information. Instead, it allows two or more processes to communicate with each other by reading/writing to/from this file. FIFO here stands for First record In will be the First record OUT"

With this change the code now became

```
cat input_file.dat | Some_Process1 > temp1.fifo &
cat temp1.fifo | Some_Process2 > temp2.fifo &
cat temp2.fifo | Some_Process3 > temp3.fifo &
cat temp3.fifo | Some_Process4 > final_file.dat
```

Only the input and the output files were actual physical files. All the other intermediate files were virtual pipes and in that way we only relied on the CPU and memory and quickly created the final file. With this example mentioned above the process saved 3 disk writes and 3 disk reads and reducing the disk IO in by 75%. This resulted in the reduction of the overall process from hours to a few minutes.

If depicted the new processing impact triangle was



Reference :
http://en.wikipedia.org/wiki/Named_pipe
http://www.linuxjournal.com/article/2156

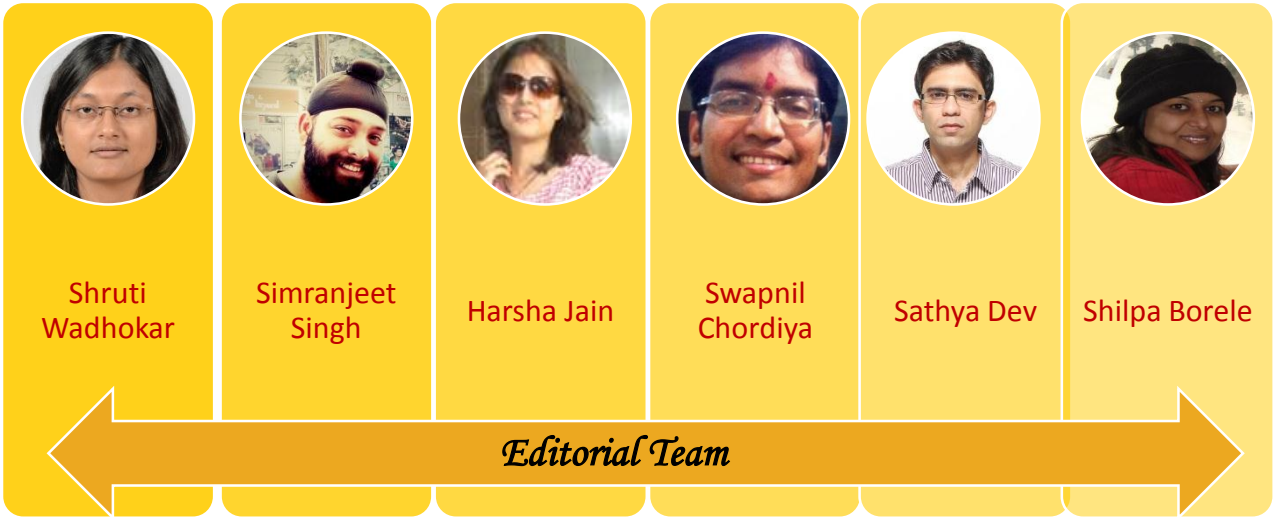
The Linux kernel's official mascot is a penguin named Tux, abbreviation of tuxedo. The idea that Linux had a pet penguin comes from Linus Torvalds himself.

Important Links

1. **Linux -** http://www.linux.org/
2. **Solaris -** http://sysunconfig.net/unixtips/solaris.html
3. **Linux Forum -** http://www.linuxforums.org/
4. **Unix, Linux & Solaris -** Sparsh-> PRIDE

From Editorial Team…

"We would like to thank all the teams and especially the participants and Tech board team for reviewing TechNews letter. We are expecting more participation to put your skills in any category in forthcoming editions."



Answers for UNIX quiz

1.b	2. c	3. a	4. b	5. b	6. a	7. c	8. c

We'd love to hear from you! Feedback and Suggestions are most welcome. Write to us

Coming Soon Testing Edition…

"Our greatest glory is not in never failing, but in rising up every time we fail"
– Ralph Waldo Emerson