Bash Shell: Take Control of PS1, PS2, PS3, PS4 and PROMPT_COMMAND

by RAMESH NATARAJAN *on* SEPTEMBER 4, 2008

Your interaction with Linux Bash shell will become very pleasant, if you use PS1, PS2, PS3, PS4, and PROMPT_COMMAND effectively. PS stands for prompt statement. This article will give you a jumpstart on the Linux command prompt environment variables using simple examples.

1. PS1 – Default interaction prompt

The default interactive prompt on your Linux can be modified as shown below to something useful and informative. In the following example, the default PS1 was "\s-\v\$", which displays the shell name and the version number. Let us change this default behavior to display the username, hostname and current working directory name as shown below.

```
-bash-3.2$  export PS1="\u@\h \w> "

ramesh@dev-db ~> cd /etc/mail
ramesh@dev-db /etc/mail>
[Note: Prompt changed to "username@hostname current-dir>" format]
```

Following PS1 codes are used in this example:

- **\u** – Username
- **\h** – Hostname
- **\w** – Full pathname of current directory. Please note that when you are in the home directory, this will display only ~ as shown above
- Note that there is a space at the end in the value of PS1. Personally, I prefer a space at the end of the prompt for better readability.

Make this setting permanent by adding export PS1="\u@\h \w> " to either .bash_profile (or) .bashrc as shown below.

```
ramesh@dev-db ~> vi ~/.bash_profile (or)

ramesh@dev-db ~> vi ~/.bashrc

[Note: Add export PS1="\u@\h \w> " to one of the above files]
```

In the next post, I'll write about several practical examples of PS1 usage in detail.

2. PS2 – Continuation interactive prompt

A very long unix command can be broken down to multiple line by giving \ at the end of the line. The default interactive prompt for a multi-line command is "> ". Let us change this default behavior to display "continue->" by using PS2 environment variable as shown below.

```
ramesh@dev-db ~> myisamchk --silent --force --fast --update-state \
> --key_buffer_size=512M --sort_buffer_size=512M \
> --read_buffer_size=4M --write_buffer_size=4M \
> /var/lib/mysql/bugs/*.MYI
[Note: This uses the default ">" for continuation prompt]

ramesh@dev-db ~> export PS2="continue-> "

ramesh@dev-db ~> myisamchk --silent --force --fast --update-state \
continue-> --key_buffer_size=512M --sort_buffer_size=512M \
continue-> --read_buffer_size=4M --write_buffer_size=4M \
continue-> /var/lib/mysql/bugs/*.MYI
[Note: This uses the modified "continue-> " for continuation prompt]
```

I found it very helpful and easy to read, when I break my long commands into multiple lines using \. I have also seen others who don't like to break-up long commands. What is your preference? Do you like breaking up long commands into multiple lines?

3. PS3 – Prompt used by "select" inside shell script

You can define a custom prompt for the select loop inside a shell script, using the PS3 environment variable, as explained below.

Shell script and output WITHOUT PS3:

```
ramesh@dev-db ~> cat ps3.sh

select i in mon tue wed exit
do
  case $i in
    mon) echo "Monday";;
    tue) echo "Tuesday";;
    wed) echo "Wednesday";;
    exit) exit;;
  esac
done

ramesh@dev-db ~> ./ps3.sh

1) mon
2) tue
3) wed
4) exit
#? 1
Monday
#? 4
[Note: This displays the default "#?" for select command prompt]
```

Shell script and output WITH PS3:

```
ramesh@dev-db ~> cat ps3.sh

PS3="Select a day (1-4): "
select i in mon tue wed exit
do
  case $i in
    mon) echo "Monday";;
    tue) echo "Tuesday";;
    wed) echo "Wednesday";;
    exit) exit;;
  esac
done

ramesh@dev-db ~> ./ps3.sh
1) mon
2) tue
3) wed
4) exit
Select a day (1-4): 1
Monday
Select a day (1-4): 4
[Note: This displays the modified "Select a day (1-4): "
       for select command prompt]
```

4. PS4 – Used by “set -x” to prefix tracing output

The PS4 shell variable defines the prompt that gets displayed, when you execute a shell script in debug mode as shown below.

Shell script and output WITHOUT PS4:

```
ramesh@dev-db ~> cat ps4.sh

set -x
echo "PS4 demo script"
ls -l /etc/ | wc -l
du -sh ~

ramesh@dev-db ~> ./ps4.sh

++ echo 'PS4 demo script'
PS4 demo script
++ ls -l /etc/
++ wc -l
243
++ du -sh /home/ramesh
48K     /home/ramesh
[Note: This displays the default "++" while tracing the output using set -x]
```

Shell script and output WITH PS4:

The PS4 defined below in the ps4.sh has the following two codes:

- $0 – indicates the name of script
- $LINENO – displays the current line number within the script

```
ramesh@dev-db ~> cat ps4.sh
```

```
export PS4='$0.$LINENO+ '
set -x
echo "PS4 demo script"
ls -l /etc/ | wc -l
du -sh ~

ramesh@dev-db ~> ./ps4.sh
../ps4.sh.3+ echo 'PS4 demo script'
PS4 demo script
../ps4.sh.4+ ls -l /etc/
../ps4.sh.4+ wc -l
243
../ps4.sh.5+ du -sh /home/ramesh
48K     /home/ramesh
[Note: This displays the modified "{script-name}.{line-number}+"
       while tracing the output using set -x]
```

5. PROMPT_COMMAND

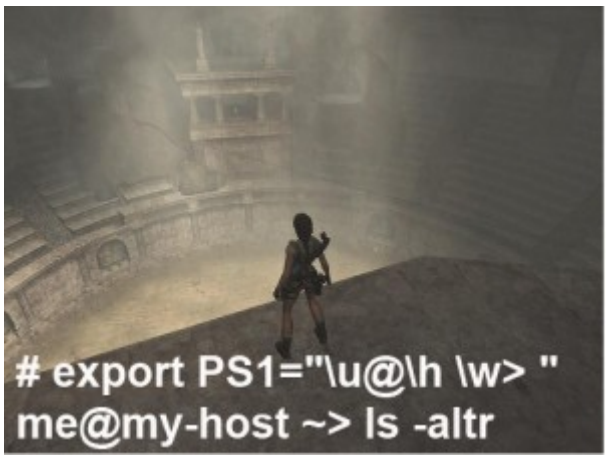
Bash shell executes the content of the PROMPT_COMMAND just before displaying the PS1 variable.

```
ramesh@dev-db ~> export PROMPT_COMMAND="date +%k:%m:%S"


22:08:42


ramesh@dev-db ~>


[Note: This displays the PROMPT_COMMAND and PS1 output on different lines]
```

If you want to display the value of PROMPT_COMMAND in the same line as the PS1, use the echo -n as shown below.

```
ramesh@dev-db ~> export PROMPT_COMMAND="echo -n [$(date +%k:%m:%S)]"
[22:08:51]ramesh@dev-db ~>
[Note: This displays the PROMPT_COMMAND and PS1 output on the same line]
```

Bash Shell PS1: 10 Examples to Make Your Linux Prompt like Angelina Jolie



by RAMESH NATARAJAN *on* SEPTEMBER 8, 2008

In the previous article, we discussed about Linux environment variables PS[1-4] and PROMPT_COMMAND. If used effectively, PS1 can provide valuable information right on the command prompt.

In Tomb Raider, Angelina Jolie has all the gadgets and weapons at her finger tips to solve the mystery in style. While the gadget and style of Angelina Jolie is hard to match, at least let us try to make the good old Linux prompt very functional and stylish using the 10 examples provided in this article.

1. Display username, hostname and current working directory in the prompt

The PS1 in this example displays the following three information in the prompt:

- **\u** – Username
- **\h** – Hostname
- **\w** – Full path of the current working directory

```
-bash-3.2$ export PS1="\u@\h \w> "

ramesh@dev-db ~> cd /etc/mail
ramesh@dev-db /etc/mail>
```

2. Display current time in the prompt

In the PS1 environment variable, you can directly execute any Linux command, by specifying in the format $(linux_command). In the following example, the command $(date) is executed to display the current time inside the prompt.

```
ramesh@dev-db ~> export PS1="\u@\h [\$(date +%k:%M:%S)]> "

ramesh@dev-db [11:09:56]>
```

You can also use \t to display the current time in the hh:mm:ss format as shown below:

```
ramesh@dev-db ~> export PS1="\u@\h [\t]> "
```

```
ramesh@dev-db [12:42:55]>
```

You can also use \@ to display the current time in 12-hour am/pm format as shown below:

```
ramesh@dev-db ~> export PS1="[\@] \u@\h> "
[04:12 PM] ramesh@dev-db>
```

3. Display output of any Linux command in the prompt

You can display output of any Linux command in the prompt. The following example displays three items separated by | (pipe) in the command prompt:

- **\!:** The history number of the command
- **\h:** hostname
- **$kernel_version:** The output of the uname -r command from $kernel_version variable
- **\$?:** Status of the last command

```
ramesh@dev-db ~> kernel_version=$(uname -r)
ramesh@dev-db ~> export PS1="\!|\h|$kernel_version|\$?> "
473|dev-db|2.6.25-14.fc9.i686|0>
```

4. Change foreground color of the prompt

Display prompt in blue color, along with username, host and current directory information

```
$ export PS1="\e[0;34m\u@\h \w> \e[m"

[Note: This is for light blue prompt]

$ export PS1="\e[1;34m\u@\h \w> \e[m"
[Note: This is for dark blue prompt]
```

- **\e[** – Indicates the beginning of color prompt
- **x;ym** – Indicates color code. Use the color code values mentioned below.
- **\e[m** – indicates the end of color prompt

Color Code Table:

```
Black 0;30

Blue 0;34

Green 0;32

Cyan 0;36
```

```
Red 0;31

Purple 0;35

Brown 0;33

[Note: Replace 0 with 1 for dark color]
```

Make the color change permanent by adding the following lines to .bash_profile or .bashrc

```
STARTCOLOR='\e[0;34m';
ENDCOLOR="\e[0m"
export PS1="$STARTCOLOR\u@\h \w> $ENDCOLOR"
```

5. Change background color of the prompt

Change the background color by specifying \e[{code}m in the PS1 prompt as shown below.

```
$ export PS1="\e[47m\u@\h \w> \e[m"
[Note: This is for Light Gray background]
```

Combination of background and foreground

```
export PS1="\e[0;34m\e[47m\u@\h \w> \e[m"
[Note: This is for Light Blue foreground and Light Gray background]
```

Add the following to the .bash_profile or .bashrc to make the above background and foreground color permanent.

```
STARTFGCOLOR='\e[0;34m';

STARTBGCOLOR="\e[47m"

ENDCOLOR="\e[0m"

export PS1="$STARTFGCOLOR$STARTBGCOLOR\u@\h \w> $ENDCOLOR"
```

Play around by using the following background color and choose the one that suites your taste:

- \e[40m
- \e[41m

- \e[42m
- \e[43m
- \e[44m
- \e[45m
- \e[46m
- \e[47m

6. Display multiple colors in the prompt

You can also display multiple colors in the same prompt. Add the following function to .bash_profile

```
function prompt {

  local BLUE="\[\033[0;34m\]"

  local DARK_BLUE="\[\033[1;34m\]"

  local RED="\[\033[0;31m\]"

  local DARK_RED="\[\033[1;31m\]"

  local NO_COLOR="\[\033[0m\]"

  case $TERM in

    xterm*|rxvt*)

      TITLEBAR='\[\033]0;\u@\h:\w\007\]'

      ;;

    *)

      TITLEBAR=""

      ;;

  esac

  PS1="\u@\h [\t]> "

  PS1="${TITLEBAR}\

  $BLUE\u@\h $RED[\t]>$NO_COLOR "

  PS2='continue-> '
```

```
  PS4='$0.$LINENO+ '

}
```

You can re-login for the changes to take effect or source the .bash_profile as shown below.

```
$. ./.bash_profile

$ prompt

ramesh@dev-db [13:02:13]>
```

7. Change the prompt color using tput

You can also change color of the PS1 prompt using tput as shown below:

```
$ export PS1="\[$(tput bold)$(tput setb 4)$(tput setaf 7)\]\u@\h:\w $ \[$(tput
sgr0)\]"
```

tput Color Capabilities:

- **tput setab [1-7]** – Set a background color using ANSI escape
- **tput setb [1-7]** – Set a background color
- **tput setaf [1-7]** – Set a foreground color using ANSI escape
- **tput setf [1-7]** – Set a foreground color

tput Text Mode Capabilities:

- **tput bold** – Set bold mode
- **tput dim** – turn on half-bright mode
- **tput smul** – begin underline mode
- **tput rmul** – exit underline mode
- **tput rev** – Turn on reverse mode
- **tput smso** – Enter standout mode (bold on rxvt)
- **tput rmso** – Exit standout mode
- **tput sgr0** – Turn off all attributes

Color Code for tput:

- 0 – Black
- 1 – Red
- 2 – Green
- 3 – Yellow
- 4 – Blue
- 5 – Magenta
- 6 – Cyan
- 7 – White

8. Create your own prompt using the available codes for PS1 variable

Use the following codes and create your own personal PS1 Linux prompt that is functional and suites your taste. Which code from this list will be very helpful for daily use? Leave your comment and let me know what PS1 code you've used for your Linux prompt.

- **\a** an ASCII bell character (07)
- **\d** the date in "Weekday Month Date" format (e.g., "Tue May 26")
- **\D{format}** – the format is passed to strftime(3) and the result is inserted into the prompt string; an empty format results in a locale-specific time representation. The braces are required
- **\e** an ASCII escape character (033)
- **\h** the hostname up to the first part
- **\H** the hostname
- **\j** the number of jobs currently managed by the shell
- **\l** the basename of the shell's terminal device name
- **\n** newline
- **\r** carriage return
- **\s** the name of the shell, the basename of $0 (the portion following the final slash)
- **\t** the current time in 24-hour HH:MM:SS format
- **\T** the current time in 12-hour HH:MM:SS format
- **\@** the current time in 12-hour am/pm format
- **\A** the current time in 24-hour HH:MM format
- **\u** the username of the current user
- **\v** the version of bash (e.g., 2.00)
- **\V** the release of bash, version + patch level (e.g., 2.00.0)
- **\w** the current working directory, with $HOME abbreviated with a tilde
- **\W** the basename of the current working directory, with $HOME abbreviated with a tilde
- **\!** the history number of this command
- **\#** the command number of this command
- **\$** if the effective UID is 0, a #, otherwise a $
- **\nnn** the character corresponding to the octal number nnn
- \\ a backslash
- **\[** begin a sequence of non-printing characters, which could be used to embed a terminal control sequence into the prompt
- **\]** end a sequence of non-printing character

9. Use bash shell function inside PS1 variable

You can also invoke a bash shell function in the PS1 as shown below.

```
ramesh@dev-db ~> function httpdcount {

>  ps aux | grep httpd | grep -v grep | wc -l

> }
```

```
ramesh@dev-db ~> export PS1="\u@\h [`httpdcount`]> "

ramesh@dev-db [12]>

[Note: This displays the total number of running httpd processes]
```

You can add the following line to .bash_profile or .bashrc to make this change permanent:

```
function httpdcount {

  ps aux | grep httpd | grep -v grep | wc -l

}

export PS1='\u@\h [`httpdcount`]> '
```

10. Use shell script inside PS1 variable

You can also invoke a shell script inside the PS1 variable. In the example below, the ~/bin/totalfilesize.sh, which calculates the total filesize of the current directory, is invoked inside the PS1 variable.

```
ramesh@dev-db ~> cat ~/bin/totalfilesize.sh

for filesize in $(ls -l . | grep "^-" | awk '{print $5}')
do
  let totalsize=$totalsize+$filesize
done
echo -n "$totalsize"

ramesh@dev-db ~> export PATH=$PATH:~/bin
ramesh@dev-db ~> export PS1="\u@\h [\$(totalfilesize.sh) bytes]> "
ramesh@dev-db [534 bytes]> cd /etc/mail
ramesh@dev-db [167997 bytes]>
[Note: This executes the totalfilesize.sh to display the total
       file size of the current directory in the PS1 prompt]
```

How much customization have you done to your PS1? Can your PS1 beat Angelina Jolie? Leave a comment and **share your PS1** prompt value.

Recommended Reading

Bash Cookbook, by Carl Albing, JP Vossen and Cameron Newham. Bash is a very powerful shell. This book will help you to master the bash shell and become highly

productive. Whether you are a sysadmin, DBA or a developer, you have to write shell script at some point. A wise sysadmin knows that once you've mastered the shell-scripting techniques, you can put your servers on auto-pilot mode by letting the shell-scripts do the grunt work. To get to the auto-pilot mode of sysadmin, you definitely need to master the examples provided in this cookbook. There are quiet few Bash shell books out there. But, this books tops them all by giving lot of detailed examples.

Additional Linux book recommendation: 12 Amazing and Essential Linux Books To Enrich Your Brain and Library