1.111.2
Tune the user environment and system environment variables
Weight 3

Linux Professional Institute Certification — 102

Nick Urbanik <nicku@nicku.org>

2005 October

Outline

Contents

1 Context

Topic 111 Administrative Tasks [21]

1.111.1 Manage users and group accounts and related system files [4]

1.111.2 <u>**Tune the user environment and system environment variables [3]**</u>

1.111.3 Configure and use system log files to meet administrative and security needs [3]

1.111.4 Automate system administration tasks by scheduling jobs to run in the future [4]

1.111.5 Maintain an effective data backup strategy [3]

1.111.6 Maintain system time [4]

2 Objectives

Description of Objective

Candidate should be able to modify global and user profiles. This includes setting environment variables, maintaining skel directories for new user accounts and setting command search path with the proper directory.

Key files, terms, and utilities include:

`/etc/profile` — To export environment variables for all users when they log in using a `bash`, `sh`, or `ksh` (and other) shell

`/etc/skel` — directory from which new home directories get a copy of files

`env` — display environment variables, or run a command with a modified environment

`export` — make environment variables available to commands

`set` — display environment, or control operation of the bash shell

`unset` — completely remove variables or functions from environment

3 What things can we set?

What things can we set?

`PATH` — a colon-separated list of directories that the shell should search to look for a command.

other environment variables — there are many, including the handy `export RSYNC_RSH=ss`

aliases, functions — discussed in topic 1.109.1 Customize and use the shell environment

shell prompts — customise the shell prompt(s) `PS1`, … in `/etc/bashrc` or `/etc/bash.bas`

`umask` — determines the default permissions when you create a file

`ulimit` — places limits on resources; in particular: core file sizes

`set` — we can set various shell options with the built-in command `set`

3.1 Setting the `PATH`

Setting the `PATH`

- The `PATH` will have already been set with initial values:

  Debian/Ubuntu in `/etc/login.defs`

  Red Hat/Fedora in `/etc/profile`

  - though on my system the PATH /usr/local/bin:/bin:/usr/bin exists when `/etc/pro` is sourced
- You need to *append* or *prefix* your existing `PATH` with other directories:
- append: `PATH="$PATH:/new/dir/bin"`
- prefix: `PATH="/new/dir/bin:$PATH"`

3.2 Prompts: `PS1`

Prompts: `PS1`

- The prompts you set go into `PS1`
- Set in `/etc/bashrc` or `/etc/bash.bashrc`
- Highly customisable
- At UNSW in mid 80's, I spent too much time making prompts that did somersaults or printed something quickly that immediately disappeared, to give subliminal messages.
  - Depended on having a 2400 bps connection to a DEC PDP11 for the delay in animation
- In `$ man bash ←`, search for `PROMPTING`
- There are also other prompts: `PS2`, `PS3`, `PS4`.

3.3 `umask`

`umask`

- Determines the default permissions of any file or directory you create
- Example: this in `/etc/bashrc` or `/etc/bash.bashrc`: `umask 022`
- … ensures that any ordinary file will have permissions `-rw-r--r--`, a directory or compiled executable will have permission `-rwxr-xr-x`

3.4 `ulimit`

`ulimit`

- To see the limits you have: `$ ulimit -a ←`
- Documentation: `$ help ulimit ←`

4 `export`

`export`

- Every *environment variable* must be *exported* if other commands are to inherit its value
- A variable only needs to be exported once
- The default startup scripts will have exported `PATH`, unless something is strangely wrong
- In `bash`, we can export variables when we define them, or separately, so we can put:

  ```
  export RSYNC_RSH=ssh
  ```

  or

  ```
  RSYNC_RSH=ssh
  export RSYNC_RSH
  ```

5 Setting options in **`bash`** with **`set`**

Setting options in `bash` with `set`

- The `bash` builtin command `shopt` controls some `bash` options, but the exam doesn't ask about it.
  - do `$ help shopt` ↩
- The builtin `bash` command `set` is also used to set many options in `bash`
- `$ set -o` ⟨*option*⟩ ↩
  - … turns ⟨*option*⟩ *on*
- `$ set +o` ⟨*option*⟩ ↩
  - … turns ⟨*option*⟩ *off*

`bash` options you can set with `set`

`emacs` or `vi` — choose whether you want `emacs`-like or `vi`-like editing of the command line.

`history` — enable/disable command history

- important for junior to use before viewing porn to avoid being sprung my mum or dad

`noclobber` — If set, disallow existing regular files to be overwritten by redirection of output.

- Override this setting with:

```
$ command >| file-to-be-clobbered-regardless.txt ↩
```

Quick Quiz

- Okay, junior wants to execute the command `$ xine -f porn-movie.wmv` ↩ without it going into `~/.bash_history`, where mum or dad might find it.
- What command should junior execute first?

6 Startup Scripts

6.1 The order in which `bash` executes scripts

login `hash` shell

- A *login shell* has '`-`' as the first character of the command name,

```
$ ps o pid,user,cmd p $$ ↩
  PID USER     CMD
 8892 nickl    -bash
```

  or has the option `--login`.
- When a *login* shell starts up, the following files are *sourced*:
  - `/etc/profile`, if it exists
  - it sources the first of these that it finds, searching for them in this order: `~/.bash_profile`, `~/.bash_login`, `~/.profile`
  - When the login shell exits, it sources `~/.bash_logout`, if it exists.

Interactive `bash` shell

- An *interactive* shell has standard input and error both connected to terminals
  - it is not being used to run a command such as `$ sh -c command` ↩ or `$ sh script.sh` ↩
- Behaviour is different on Fedora and Ubuntu systems (Why???)

  Fedora/Red Hat — If the shell is not a login shell, then it will source `~/.bashrc`, if it exists.

  Ubuntu/Debian — If the shell is not a login shell, then it will source both `/etc/bash.bashrc` and `~/.bashrc`, if each of them exists.

Noninteractive shells

- A non-interactive shell (e.g., one that has been started to execute a command) will source the file whos name is in the environment variable `BASH_ENV`

6.2 What Sources What

What sources what

- `~/.bash_profile` sources `~/.bashrc`
- `~/.bashrc` sources `/etc/bashrc`
- `/etc/bashrc` sources `/etc/profile.d/*.sh` if this is not a login shell
- `/etc/profile` sources `/etc/profile.d/*.sh`
- This means:
  - `/etc/profile` and `~/.bash_profile` are sourced *only* when a user logs in where their shell is `bash`, `sh`, `ksh`, `ash` and a few other shells, by *whatever means*
  - `~/.bashrc`, `/etc/bashrc` and `/etc/profile.d/*.sh` are sourced for *every* new interactive shell, including login shells.

What sources what

- `/etc/profile` sources `/etc/bash.bashrc`
- `/etc/bash.bashrc` sources `/etc/bashrc.local` ✔
- `~/.bash_profile` sources `~/.bashrc`
- This means:
  - `/etc/profile` and `~/.bash_profile` are sourced *only* when a user logs in where their shell is `bash`, `sh`, `ksh`, `ash` and a few other shells, by *`ssh` and a text console only*
  - `/etc/bash.bashrc` and `~/.bashrc` and `/etc/bashrc.local` are sourced for *every* new interactive shell, including login shells.

6.3 Weird stuff

Weird stuff

- The file `/etc/bashrc` is not read directly by `bash`
  - Red Hat, Fedora systems source `/etc/bashrc` from `~/.bashrc`
- Red Hat, Fedora systems source `~/.bashrc` from `~/.bash_profile`
- When you log into an Ubuntu system via `gdm`, it will *not* source `/etc/profile`!
  - However, the file `/etc/bash.bashrc` *does* (somehow) get read!
  - The file `/etc/profile` *is* sourced when you log in via `ssh` or at a text console!
  - You can define environment variables in `/etc/environment`, but do not use `export` there, since it is not parsed by the shell.
  - It gets curiouser and curiouser.

6.4 Executive Summary

Executive Summary for the suit on the go

- Export variables and the `PATH` from `/etc/profile` on a Fedora/Red Hat system for all users, since it is sourced once only, when logging in, via `gdm`, `kdm`, `ssh` or a console;
- define aliases and functions and the prompts `PS1`, `PS2`, ... in `/etc/bashrc` on Red Hat/Fedora systems, since all `~/.bashrc` scripts will source it by default whenever a new interactive shell is started
- A better place for aliases and function definitions is a file in `/etc/profile.d/` — you might call it `local.sh` — since upgrades will not affect it.

Executive Summary for the suit on the go

- Export variables and the `PATH` from `/etc/bashrc.local`, since

  `/etc/bash.bashrc` sources `/etc/bashrc.local` and `/etc/profile` sources `/etc/bash.bashrc`, if you want them set the same for all logins, since `/etc/profile` will not be read when you log in via `gdm`. In fact, `/etc/bashrc.local` will be read whenever you start a new interactive `bash` shell, so it is also the place to define aliases and functions and local customisations to prompts `PS1`, `PS2`, ...
- You can add global environment variables to `/etc/environment`, but just assign variables, do not use export.
- If someone can explain the rationale for not reading `/etc/profile` from `gdm`, please let me know. There are issues of security, and setting environment variables independently of shell.

7 Other places to put settings

7.1 `/etc/login.defs`

`/etc/login.defs`

- `/etc/login.defs` appears to have different roles on Red Hat/Fedora systems from Debian/Ubuntu systems.
- On Debian systems, `/etc/login.defs` appears to be read when a user logs in or changes settings. The `umask` value is set there, as is the initial value of `PATH`.
- See `$ man login.defs` ←↩ on Debian.
- Red Hat/Fedora systems read `/etc/login.defs` when creating user accounts with `shadow-utils` commands including `useradd`, `usermod`, `groupadd`, …
- There is no man page on Fedora, but it is mentioned in the man pages for the `shadow-utils` commands.

8 The `/etc/skel` directory

The `/etc/skel` directory

- When a user's home directory is created using tools such as `useradd` or `adduser`, the contents of `/etc/skel` are all copied to the new directory
- You can customise the login scripts
- You can create a `/etc/skel/bin` directory, so each new user will have a `~/bin` directory
- See topic 1.111.1 Manage users and group accounts and related system files for how `useradd`, … use `/etc/skel`

9 License Of This Document

License Of This Document