The alias Command

The *alias command* makes it possible to launch any command or group of commands (inclusive of any options, arguments and redirection) by entering a pre-set *string* (i.e., sequence of characters).

That is, it allows a user to create simple names or abbreviations (even consisting of just a single character) for commands regardless of how complex the original commands are and then use them in the same way that ordinary commands are used.

A command is an instruction given by a user to tell a computer to do something. Commands are generally issued by typing them in at the *command line* (i.e., an all-text user interface) and then pressing the ENTER key, which passes them to the *shell*. A shell is a program that provides the traditional, text-only user interface for a Unix-like operating systems. Its primary function is to read commands and then *execute* (i.e., run) them.

The alias command is built into a number of shells including *ash, bash* (the default shell on most Linux systems), *csh* and *ksh*. It is one of several ways to customize the shell (another is setting *environmental variables*). Aliases are recognized only by the shell in which they are created, and they apply only for the user that creates them, unless that user is the *root* (i.e., administrative) user, which can create aliases for any user.

Listing and Creating Aliases

The general syntax for the alias command varies somewhat according to the shell. In the case of the bash shell it is

```
alias [-p] [name="value"]
```

When used with no arguments and with or without the *-p* option, alias provides a list of aliases that are in effect for the current user, i.e.,

```
alias
```

Some of the aliases listed are likely to be system-wide aliases that apply to all users and are created automatically for each new user for a particular shell. Aliases for any other shell can be seen by first switching to that shell and then using the alias command as above.

name is the name of the new alias and *value* is the command(s) which it initiates. The alias name and the replacement text can contain any valid shell input except for the *equals* sign (=).

The commands, including any options, arguments and redirection operators, are all enclosed within a single pair of quotation marks, which can be single quotes or double quotes. No spaces are permitted before or after the equals sign. Any number of aliases can be created simultaneously by enclosing the name in each name-value pair in quotes.

As a trivial example of alias creation, the alias *p* could be created for the commonly used *pwd* command, which shows the current location of the user in the directory structure (and which is an abbreviation for *present working directory*), by typing the following command and then pressing the ENTER key:

```
alias p="pwd"
```

Then, to show the current location, instead of typing *pwd*, the user would only have to type the letter *p* and press the ENTER key, i.e.,

```
p
```

An alias can be created with the same name as the core name of a command (i.e., a command without any options or arguments). In such case, it is the alias that is called (i.e., activated) first when the name is used, rather than the command with the same name. For example, an alias named *ls* could be created for the command *ls -al* as follows:

```
alias ls="ls -al"
```

ls is a commonly used command that by default lists the names of the files and directories within the *current directory* (i.e., the directory in which the user is currently working). The *-a* option instructs *ls* to also show any hidden files and directories, and the *-l* option tells it to provide detailed information about each file and subdirectory.

Such an alias can be disabled temporarily and the core command called by preceding it directly (i.e., with no spaces in between) with a backslash, i.e.,

```
\ls
```

It makes no difference whether double or single quotes are used when creating an alias. It can be slightly easier to use single quotes because this obviates the need to simultaneously use the shift key. Thus, the above example could have been written as

```
alias ls='ls -al'
```

This example could be simplified even further, again with no adverse effect on performance, by using a single character instead of two characters for the alias name, for example:

```
alias l='ls -al'
```

In addition to options, arguments can also be included in alias values. For example, to have the *ls* alias always display the contents of the */etc* directory, it could be rewritten as:

```
alias l='ls -al /etc'
```

Multiple commands can be included in the same alias by inserting them within the same pair of quotation marks and separating them with semicolons. For example, the alias *pl* could be created to first launch pwd and then immediately launch ls:

```
alias pl='pwd; ls'
```

Aliases can even be created to call other aliases. For example, if the alias ls as shown earlier had already been created, then pl will launch it in the above example, otherwise it will launch the conventional ls command. If the aliases p and l shown in earlier examples have already been created, then the above example could alternatively be written as

```
alias pl='p; l'
```

The following is an example of creating two separate aliases simultaneously, as contrasted with creating a single alias that launches two separate commands:

```
alias p="pwd"; l="ls -al"
```

As an example of an alias for a series of commands linked by a *pipe* (represented by a vertical line), the alias *dir* can be created to generate a list of the names of and information about all of the subdirectories in the current directory:

```
alias dir="ls -al | grep ^d"
```

Here ls -al obtains a listing of all files and directories in the current directory. Its output is sent by the pipe to the filter *grep*, which then searches for lines beginning with the letter *d* (as all directories have a line returned by ls -al that begins with d). The *caret* (i.e., upward-pointing

angular character) before d tells grep to search only for lines *beginning* with that letter.

Not only can options and arguments be used in the command(s) that an alias can substitute for, but they can also be used with an alias that has already been created. As a trivial example, supposing the alias l is created for the ls command:

```
alias l="ls -a"
```

Then, the alias l could be used with any argument that the command ls could be used with. For example, to list the files and directories in the the /etc directory:

```
l /etc
```

The alias l could also be used with any option that the command ls could be used with. For example:

```
l -l /etc
```

The alias command is unusual in that it only has a single option. That option, *-p*, tells it to display a list of the aliases for the current user on the current shell. This might be helpful if used when creating an alias, but it is, of course, redundant when the alias command is used without arguments.

Uses For Aliases

There are several types of uses for aliases. They include:

(1) Reducing the amount of typing that is necessary for commands or groups of commands that are long and/or tedious to type. These commands could include opening a file that is frequently used for studying or editing.

For example, if a user often accesses the Apache web server configuration file, which is */etc/httpd/conf/httpd.conf* on Red Hat Linux 9, and uses the *gedit* text editor to read it, such user might type each time:

```
gedit /etc/httpd/conf/httpd.conf
```

However, this could quickly become tedious. It would be much easier to make this command into an alias and give it a short name, perhaps even a single letter, such as *a*:

```
alias a="gedit /etc/httpd/conf/httpd.conf"
```

Then, whenever the user wants to open the Apache configuration file using gedit, all that is necessary is to type the following single-letter command and press the ENTER key:

```
a
```

Note that the location of the Apache configuration file might be different on different systems. Also, ordinary users will likely only be able to open it for reading, and only the root user will be able to edit it (unless the permission settings are changed).

(2) A second type of use for aliases is specifying the default version of a program that exists in several versions on a system or specifying default options for a command. For example, the following alias would have the ls command always list all items in a directory rather than just the non-hidden ones:

```
alias ls="ls -a"
```

Another example of setting a more convenient default option for a command is *df*, which is used to show information about each partition on the system, including name, size, amount of space used, amount of space available and *mountpoint* (i.e., where it is attached to the system). The default version of df shows this data in terms of one kilobyte blocks (a holdover from the days when 1K was considered a lot of storage), but the *-h* (i.e., *human readable*) option makes the data easier to read by expressing it in terms of MB (megabytes) and GB (gigabytes). -h can be set as the default option with the following command:

```
alias df="df -h"
```

(3) A third use of aliases is correcting common misspellings of commands. For example, if a user has a habit of accidentally typing *pdw* instead of *pwd*, the following command will create an alias so that either spelling will work:

```
alias pdw="pwd"
```

(4) A fourth use of aliases is increasing the safety of the system by making commands interactive. This forces the user to confirm that it is desired to perform a specific action and thereby reduces the risk from accidental or impulsive abuse of powerful commands. For example, the *rm* command, which can remove files and directories and make them virtually unrecoverable, can be made interactive as follows:

```
alias rm="rm -i"
```

Likewise, the risks associated with the *cp* command, which is used to copy the contents of one file to another file, can also be reduced by making it interactive by default. If the name for the file to be written to does not exist in the specified directory (by default the current directory), it will be created, but if it already exists, its contents will be overwritten. Thus, creating the following alias will reduce the chances of an unintended overwriting:

```
alias cp="cp -i"
```

(5) Another use of aliases is standardizing the name of a command across multiple operating systems. For example, different versions of Linux or other operating systems contain different versions of the *vi* text editor (i.e., vi or its clones *vim*, *nvi*, *elvis*, etc.), but issuing the *vi* command on any of these divergent systems will generally launch the particular vi clone that is resident on that system (assuming that the appropriate alias has been created). For instance, Red Hat Linux installs vim by default, but issuing the command vi launches vim because the alias *alias vi="vim"* is also installed by default for all users for the bash, csh and tcsh shells. Of course, the command *vim* can also be used, but *vi* is easier for most people to remember.

For people accustomed to MS-DOS commands, the following aliases can be defined so that a Unix-like operating system appears to behave more like MS-DOS:

```
alias dir="ls"
alias copy="cp"
alias rename="mv"
alias md="mkdir"
alias rd="rmdir"
alias del="rm -i"
```

However, some experienced users of Unix-like systems contend that this may not be a good idea and that it might just make Linux seem more confusing, rather than simpler. Instead, they advocate having Linux users become accustomed to the UNIX terminology right from the start.

Making Aliases Permanent

The main disadvantage with the alias command is that any alias set up with it remains in effect only during the current login session (i.e., until the user logs out or the computer is shut down). Although this might not be much of a problem for systems which are *rebooted* (i.e., restarted) only infrequently (such as corporate database servers), it can be a nuisance for systems that are frequently rebooted (e.g., home computers).

Fortunately, however, any alias can be made more enduring (i.e., until it is explicitly removed) by writing it to the appropriate configuration file with a text editor. The name and location of such file can vary according to the system. In the case of Red Hat Linux, an alias for any user can be added to the *.bashrc* file in that user's home directory. Because this file is read at login, the change will not take effect until the user has logged in again.

Aliases for the root user can be made permanent by entering them in the .bashrc file in the root's home directory, i.e., in */root/.bashrc*. System-wide aliases can be put in the */etc/bashrc* file. The system needs to be restarted before system-wide aliases can take effect.

Removing Aliases

The command *unalias*, which is likewise built into bash and some other shells, is used to remove entries from the current user's list of aliases. Its syntax is

```
unalias [-a] name(s)
```

For example, the following would remove the alias *rm* which was created in an earlier example:

```
unalias rm
```

unalias removes not only aliases created during the current session but also *permanent* aliases that are listed in system configuration files. The *-a* option tells unalias to remove all aliases for the current user for the current shell.

A second way to remove an alias is by using the alias command to create a new alias with the same name. This overwrites the existing alias with that name.

A third way is to delete the alias from the appropriate configuration file using a text editor. For example, in the case of Red Hat, deleting an alias in the bash shell for a user named joe would involve removing the appropriate line in the file /home/joe/.bashrc. Likewise, an alias can be modified by editing the appropriate line in the configuration file.