Linux Tools for Developers

Name	Purpose
ls	List files
cat	Type out (concatenate) the files
rm	Remove files
mv	Rename (move) files
mkdir	Create directories
rmdir	Remove directories
file	Show file types
ln	Create symbolic and hard links
tail	Look at the tail end of the file
head	Look at the beginning of the file
less	Look at the file, one screenful at a time
more	Look at the file, one screenful at a time
touch	Either create an empty file, or update the file modification time
wc	Count lines, words, and bytes in a file

Finding Files: find and locate

The **find** command line utility provides an extremely powerful and flexible method for locating files based on their properties, including name. It does not search the interior of files for patterns, etc.; that is more the province of grep and its variations.

The general form of a find command is:

```
$ find [location] [criteria] [actions]
```

where there are three classes of arguments, each of which may be omitted. If no location is given, the current directory (.) is assumed; if no criteria are given, all files are displayed; and, if no actions are given, only a listing of the names is given.

There are many logical expressions which can be used for criteria. For example, the command:

```
$ find /etc -name "*.conf"
```

will print out the names of all files in the /etc directory and its descendants, recursively, that end in .conf. To specify a simple action request:

```
$ find /etc -name "*.conf" -ls
```

will print out a long listing, not just the names.

A little more complicated example is the following:

```
$ find /tmp /etc -name "*.conf" -or -newer /tmp/.X0-lock -ls
```

will look in subdirectories under /etc and /tmp for files whose names end in .conf, or are newer than /tmp/.X0-lock and print out a long listing.

You can perform actions with the -exec option, as in:

```
$ find . -name "*~" -exec rm {} ';'
```

where {} is a fill in for the files to be operated on, and ';' indicates the end of the command.

Another method of locating files is provided by the **locate** command, which searches your entire filesystem (except for paths which have been excluded) and works off a database that is updated periodically with updatedb. Thus, it is very fast.

Thus, the command:

```
$ locate .conf
```

will find all files on your system that have .conf in them.

locate will only find files that were already in existence the last time the database was updated. On most systems, this is done by a background cron job, usually daily. To force an update, you need to do:

```
$ sudo updatedb
```

grep Command

grep is a workhorse command line utility whose basic job is to search files for patterns and print out matches according to specified options.

Its name stands for global regular expression print, which points out that it can do more than just match simple strings; it can work with more complicated regular expressions which can contain wildcards and other special attributes.

The simplest example of using grep would be:

```
$ grep pig file

pig
dirty pig
pig food
```

which finds three instances of the string "pig" in file.

grep has many options; some of the most important are:

Option	Meaning
-i	Ignore case
-v	Invert match
-n	Print line number
-H	Print filename
-a	Treat binary files as text
-I	Ignore binary files
-r	Recurse through subdirectories
-l	Print out names of all files that contain matches
-L	Print out names of all files that do not contain matches
-c	Print out number of matching lines only
-e	Use the following pattern; useful for multiple strings and special characters

Command	Usage
grep [pattern] <filename>	Search for a pattern in a file and print all matching lines
grep -v [pattern] <filename>	Print all lines that do not match the pattern
grep [0-9] <filename>	Print the lines that contain the numbers 0 through 9
grep -C 3 [pattern] <filename>	Print context of lines (specified number of lines above and below the pattern) for matching the pattern; here, the number of lines is specified as 3

sed Command

sed stands for stream editor. Its job is to make substitutions and other modifications in files and in streamed output.

Any of the following methods will change all first instances of the string "pig" with "cow" for each line of **file**, and put the results in **newfile**:

```
$ sed s/pig/cow/ file > newfile
$ sed s/pig/cow/ < file > newfile
$ cat file | sed s/pig/cow/ > newfile
```

where the **s** stands for substitute. If you want to change all instances, you have to add the **g** (global) qualifier, as in:

```
$ sed s/pig/cow/g file > newfile
```

The / characters are used to delimit the new and old strings. You can choose to use another character, as in:

```
$ sed s:pig:cow:g file > newfile
```

you can work directly on streams generated from commands, as in:

```
$ echo hello | sed s/"hello"/"goodbye"/g

goodbye
```

If you want to make multiple simultaneous substitutions, you need to use the **-e** option, as in:

```
$ sed -e s/"pig"/"cow"/g -e s/"dog"/"cat"/g < file > newfile
```

If you have a lot of commands, you can put them in a file and apply the **-f** option, as in:

```
$ cat scriptfile

s/pig/cow/g
s/dog/cat/g
s/frog/toad/g

$ sed -f scriptfile < file > newfile
```

Command	**Usage**
sed s/pattern/replace_string/ file	Substitute first string occurrence in every line
sed s/pattern/replace_string/g file	Substitute all string occurrences in every line
sed 1,3s/pattern/replace_string/g file	Substitute all string occurrences in a range of lines
sed -i s/pattern/replace_string/g file	Save changes for string substitution in the same file

cat

cat is short for concatenate and is one of the most frequently used Linux command line utilities. It is often used to read and print files, as well as for simply viewing file contents. To view a file, use the following command:

```
$ cat <filename>
```

For example, **cat readme.txt** will display the contents of **readme.txt** on the terminal. However, the main purpose of **cat** is often to combine (concatenate) multiple files together.

Command	Usage
cat file1 file2	Concatenate multiple files and display the output; i.e. the entire content of the first file is followed by that of the second file
cat file1 file2 > newfile	Combine multiple files and save the output into a new file
cat file >> existingfile	Append a file to the end of an existing file
cat > file	Any subsequent lines typed will go into the file, until ***Ctrl-D*** is typed
cat >> file	Any subsequent lines are appended to the file, until ***Ctrl-D*** is typed

echo

echo displays (echoes) text. It is used simply, as in:

```
$ echo string
```

echo is particularly useful for viewing the values of environment variables (built-in shell variables). For example, **echo $USERNAME** will print the name of the user who has logged into the current terminal.

The following table lists **echo** commands and their usage:

Command	Usage
echo string > newfile	The specified string is placed in a new file
echo string >> existingfile	The specified string is appended to the end of an already existing file
echo $variable	The contents of the specified environment variable are displayed

head

head reads the first few lines of each named file (10 by default) and displays it on standard output. You can give a different number of lines in an option.

For example, if you want to print the first 5 lines from **grub.cfg**, use the following command:

```
$ head -n 5 grub.cfg
```

You can also just say **head -5 grub.cfg**.

tail

tail prints the last few lines of each named file and displays it on standard output. By default, it displays the last 10 lines. You can give a different number of lines as an option. **tail** is especially useful when you are troubleshooting any issue using log files, as you probably want to see the most recent lines of output.

For example, to display the last 15 lines of **somefile.log**, use the following command:

```
$ tail -n 15 somefile.log
```

You can also just say **tail -15 somefile.log**.

To continually monitor new output in a growing log file:

```
$ tail -f somefile.log
```

This command will continuously display any new lines of output in **somefile.log** as soon as they appear. Thus, it enables you to monitor any current activity that is being reported and recorded.

Viewing Compressed Files

Command	Description
$ zcat compressed-file.txt.gz	To view a compressed file
$ zless somefile.gz or $ zmore somefile.gz	To page through a compressed file
$ zgrep -i less somefile.gz	To search inside a compressed file
$ zdiff file1.txt.gz file2.txt.gz	To compare two compressed files

awk

The table below explains the basic tasks that can be performed using **awk**. The input file is read one line at a time, and, for each line, **awk** matches the given pattern in the given order and performs the requested action. The **-F** option allows you to specify a particular field separator character. For example, the **/etc/passwd** file uses "**:**" to separate the fields, so the **-F:** option is used with the **/etc/passwd** file.

The command/action in **awk** needs to be surrounded with apostrophes (or single-quote (')**). awk** can be used as follows:

Command	**Usage**
awk '{ print $0 }' /etc/passwd	Print entire file
awk -F: '{ print $1 }' /etc/passwd	Print first field (column) of every line, separated by a space
awk -F: '{ print $1 $7 }' /etc/passwd	Print first and seventh field of every line

sort

sort is used to rearrange the lines of a text file either in ascending or descending order, according to a sort key. You can also sort by particular fields of a file. The default sort key is the order of the ASCII characters (i.e. essentially alphabetically).

sort can be used as follows:

Syntax	**Usage**
sort <filename>	Sort the lines in the specified file, according to the characters at the beginning of each line
cat file1 file2 | sort	Combine the two files, then sort the lines and display the output on the terminal
sort -r <filename>	Sort the lines in reverse order

sort -k 3 <filename>	Sort the lines by the 3rd field on each line instead of the beginning

uniq

uniq removes duplicate consecutive lines in a text file and is useful for simplifying the text display.

Because **uniq** requires that the duplicate entries must be consecutive, one often runs **sort** first and then pipes the output into **uniq**; if **sort** is used with the **-u** option, it can do all this in one step.

To remove duplicate entries from multiple files at once, use the following command:

```
sort file1 file2 | uniq > file3
```

or

```
sort -u file1 file2 > file3
```

To count the number of duplicate entries, use the following command:

```
uniq -c filename
```

paste

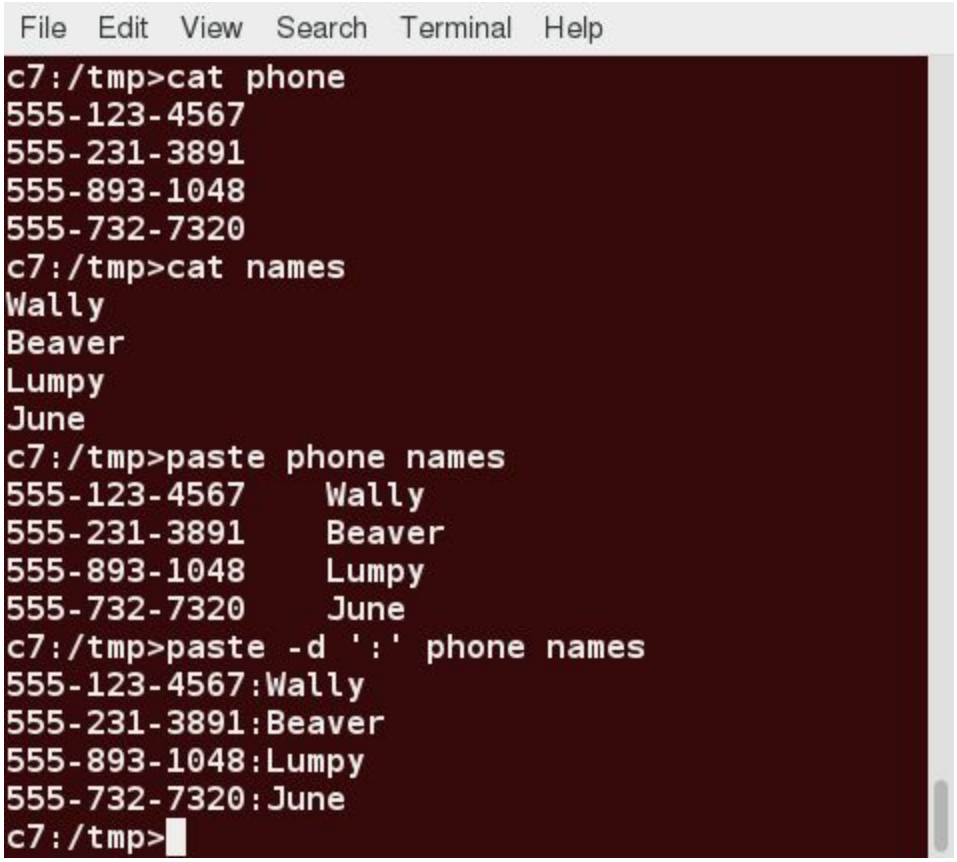
paste can be used to combine fields (such as name or phone number) from different files, as well as combine lines from multiple files. For example, line one from **file1** can be combined with line one of **file2**, line two from **file1** can be combined with line two of **file2**, and so on.

To paste contents from two files, you can do:

```
$ paste file1 file2
```

The syntax to use a different delimiter is as follows:

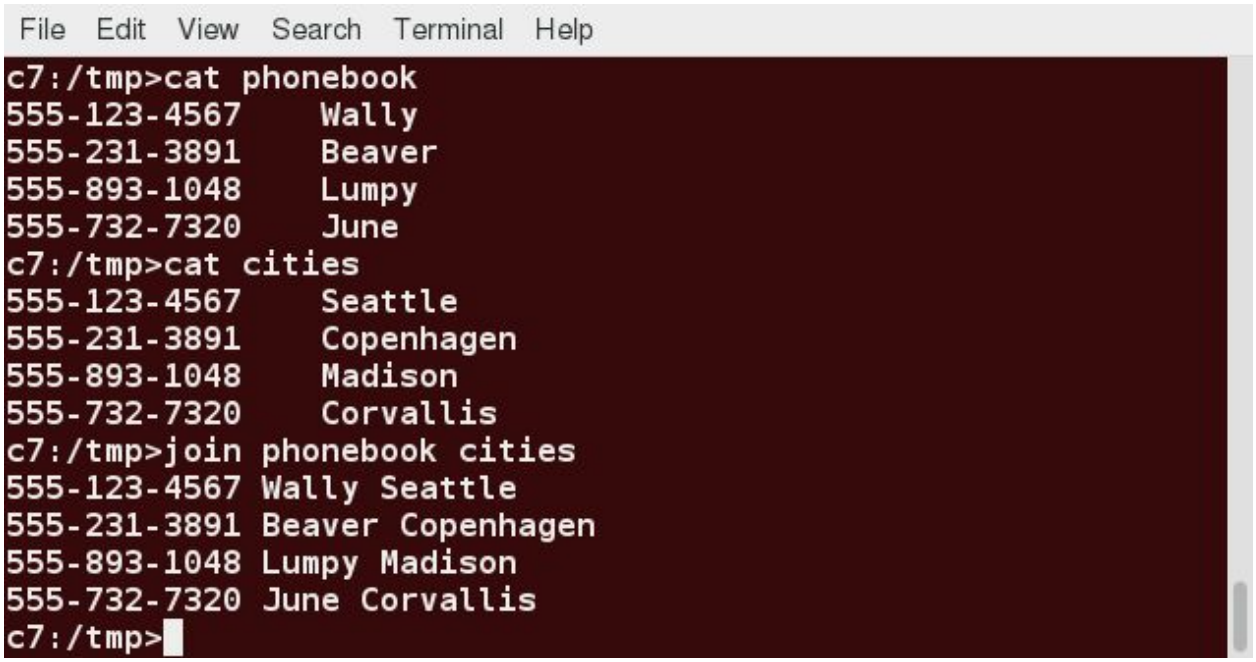
```
$ paste -d ':' file1 file2
```



join

To combine two files on a common field, at the command prompt type **join file1 file2** and press the ***Enter*** key.

For example, the common field (i.e. it contains the same values) among the phonebook and cities files is the phone number, and the result of joining these two files is shown in the screen capture.



split

[document from somewhere else]

tr

The tr utility is used to translate specified characters into other characters or to delete them. The general syntax is as follows:

```
$ tr [options] set1 [set2]
```

The items in the square brackets are optional. tr requires at least one argument and accepts a maximum of two. The first designated set1 in the example lists the characters in the text to be replaced or removed. The second, set2, lists the characters that are to be substituted for the characters listed in the first argument. It is usually safe (and may be required) to use the single-quotes around each of the sets, as you will see in the examples below.

For example, suppose you have a file named city containing several lines of text in mixed case. To translate all lower case characters to upper case, at the command prompt type cat city | tr a-z A-Z and press the Enter key.

Command	Usage
$ tr abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ	Convert lower case to upper case
$ tr '{}' '()' < inputfile > outputfile	Translate braces into parenthesis
$ echo "This is for testing" | tr [:space:] '\t'	Translate white-space to tabs
$ echo "This is for testing" | tr -s [:space:]	Squeeze repetition of characters using **-s**
$ echo "the geek stuff" | tr -d 't'	Delete specified characters using **-d** option
$ echo "my username is 432234" | tr -cd [:digit:]	Complement the sets using **-c** option
$ tr -cd [:print:] < file.txt	Remove all non-printable characters from a file
$ tr -s '\n' ' ' < file.txt	Join all the lines in a file into a single line

tee

tee takes the output from any command, and, while sending it to standard output, it also saves it to a file. In other words, it "tees" the output stream from the command: one stream is displayed on the standard output and the other is saved to a file.

For example, to list the contents of a directory on the screen and save the output to a file, at the command prompt type ls -l | tee newfile and press the Enter key.

wc

wc (word count) counts the number of lines, words, and characters in a file or list of files. Options are given in the table below:

Option	Description
–l	Displays the number of lines
-c	Displays the number of bytes
-w	Displays the number of words

By default, all three of these options are active.

cut

cut is used for manipulating column-based files and is designed to extract specific columns. The default column separator is the **tab** character. A different delimiter can be given as a command option.

Types of Files

Character	Type
-	Normal File
d	Directory
l	Symbolic link
p	Named pipe (FIFO)
s	Unix domain socket
b	Block device node
c	Character device node

Changing Permissions and Ownership

Changing file permissions is done with **chmod**, while changing file ownership is done with **chown**, and changing the group is done with **chgrp**.

There are a number of different ways to use **chmod**. For instance, to give the **owner** and **world** execute permission, and remove the **group** write permission:

```
$ ls -l a_file
-rw-rw-r-- 1 coop coop 1601 Mar  9 15:04 a_file
$ chmod uo+x,g-w a_file
$ ls -l a_file
-rwxr--r-x 1 coop coop 1601 Mar  9 15:04 a_file
```

where **u** stands for user (owner), **o** stands for other (world), and **g** stands for group.

This kind of syntax can be difficult to type and remember, so one often uses a shorthand which lets you set all the permissions in one step. This is done with a simple algorithm, and a single digit suffices to specify all three permission bits for each entity. This digit is the sum of:

- 4 if read permission is desired
- 2 if write permission is desired
- 1 if execute permission is desired.

Thus, 7 means read/write/execute, 6 means read/write, and 5 means read/execute.

When you apply this when using **chmod**, you have to give three digits for each degree of freedom, such as in:

```
$ chmod 755 a_file
$ ls -l a_file
-rwxr-xr-x 1 coop coop 1601 Mar  9 15:04 a_file
```

Changing the group ownership of the file is as simple as doing:

```
$ chgrp aproject a_file
```

and changing the ownership is as simple as:

```
$ chown coop a_file
```

You can change both at the same time with:

```
$ chown coop.aproject a_file
```

where you separate the owner and the group with a period.

All three of these commands can take an **-R** option, which stands for recursive. For example:

```
$ chown -R coop.aproject .
$ chown -R coop.aproject subdir
```

will change the owner and group of all files in the current directory and all its subdirectories in the first command, and in **subdir** and all its subdirectories in the second command.

These commands can only do what you already have the right to do. You cannot change the permissions on a file you do not own, for example, or switch to a group you are not a member of. To do such changes, you must be root, prefixing **sudo** to most of the above commands.