sourcemind

Working in Linux

Mocktar ISSA
Full-Stack Software Engineer
SBIN SA[CELTIIS],

sourcemind

Compressing Files

Compression reduces the amount of data needed to store or transmit a file while storing it in such a way that the file can be restored. A file with human-readable text might have frequently used words replaced by something smaller, or an image with a solid background might represent patches of that color by a code. The compressed version of the file is not typically viewed or utilized, instead, it is decompressed before use.

The compression algorithm is a procedure the computer uses to encode the original file, and as a result, make it smaller. Computer scientists research these algorithms and come up with better ones that can work faster or make the input file smaller.

When talking about compression, there are two types:

- *Lossless:* No information is removed from the file. Compressing a file and decompressing it leaves something identical to the original.
- *Lossy:* Information might be removed from the file. It is compressed in such a way that uncompressing a file will result in a file that is slightly different from the original. For instance, an image with two subtly different shades of green might be made smaller by treating those two shades as the same. Often, the eye can’t pick out the difference anyway.

Linux provides several tools to compress files; the most common is gzip. Here we show a file before and after compression:

```
adavtyan@artur-lpt:~/tmp$ ls -l kern.log
-rw-r----- 1 adavtyan adavtyan 176788 Ապր 26 14:55 kern.log
adavtyan@artur-lpt:~/tmp$ gzip kern.log
adavtyan@artur-lpt:~/tmp$ ls -l kern.log.gz
-rw-r----- 1 adavtyan adavtyan 28015 Ապր 26 14:55 kern.log.gz
```

The gzip command will provide this information, by using the -l option, as shown here:

```
adavtyan@artur-lpt:~/tmp$ gzip -l kern.log.gz
         compressed        uncompressed  ratio uncompressed_name
              28015              176788  84.2% kern.log
```

Compressed files can be restored to their original form using either the gunzip command or the gzip -d command. This process is called decompression. After gunzip does its work, the longfile.txt file is restored to its original size and file name.

```
adavtyan@artur-lpt:~/tmp$ gunzip kern.log.gz
adavtyan@artur-lpt:~/tmp$ ls -l kern.log
-rw-r----- 1 adavtyan adavtyan 176788 Ապր 26 14:55 kern.log
```

Note: There are other commands that operate virtually identically to gzip and gunzip. There is bzip2 and bunzip2, as well as xz and unxz.

Archiving Files

If you had several files to send to someone, you could choose to compress each one individually. You would have a smaller amount of data in total than if you sent uncompressed files, however, you would still have to deal with many files at one time.

Archiving is the solution to this problem. Archive files are typically used for a transfer (locally or over the internet) or make a backup copy of a collection of files and directories which allow you to work with only one file (if compressed, it has a lower size than the sum of all files within it) instead of many. Likewise, archives are used for software application packaging. This single file can be easily compressed for ease of transfer while the files in the archive retain the structure and permissions of the original files.

The traditional UNIX utility to archive files is called tar, which is a short form of TAPE Archive. It was used to stream many files to a tape for backups or file transfer. The tar command takes in several files and creates a single output file that can be split up again into the original files on the other end of the transmission.

The tar command has three modes that are helpful to become familiar with:

Mode	**Function**
Create	Make a new archive out of a series of files.
Extract	Pull one or more files out of an archive.
List	Show the contents of the archive without extracting.

Remembering the modes is key to figuring out the command line options necessary to do what you want. In addition to the mode, remember where to specify the name of the archive, as you may be entering multiple file names on a command line.

Archiving Create Mode

To create an archive with tar, use the ‘-c’ (“create”) option, and specify the name of the archive file to create with the ‘-f’ option. It’s common practice to use a name with a ‘.tar’ extension. Note that unless specifically mentioned otherwise, all commands and command parameters used in the remainder of this article are used in lowercase.

```
tar -c [ -f ARCHIVE ] [ OPTION ] [ FILE ... ]
```

Creating an archive with the tar command requires two named options:

Option	Function
-c	Create an archive
-f archive	Use archive file. The argument archive will be the name of the resulting archive file

Example:

```
adavtyan@artur-lpt:~/tmp$ tar -cf kern.log.tar kern.log
adavtyan@artur-lpt:~/tmp$ ls -l kern.log.tar
-rw-rw-r-- 1 adavtyan adavtyan 184320 Ապր 26 16:33 kern.log.tar
```

Normally, tarball files are slightly larger than the combined input files due to the overhead information on recreating the original files. Tarballs can be compressed for easier transport, either by using gzip on the archive or by having tar do it with the -z option.

```
adavtyan@artur-lpt:~/tmp$ tar -czf kern.log.tar.gz kern.log
adavtyan@artur-lpt:~/tmp$ ls -l  kern.log.tar.gz
-rw-rw-r-- 1 adavtyan adavtyan 28134 Ապր 26 16:37 kern.log.tar.gz
adavtyan@artur-lpt:~/tmp$ gzip -l kern.log.tar.gz
         compressed        uncompressed  ratio uncompressed_name
              28134              184320  84.7% kern.log.tar
```

sourcemind

Archiving List Mode

To list the contents of a tar archive without extracting them, use tar with the ‘-t’ option.

tar -t [-f ARCHIVE] [OPTION]

The next example uses three options:

Option	Function
-t	List the files in an archive
-j	Decompress with bzip2 command
-f archive	Use archive file. The argument archive will be the name of the resulting archive file

Example: Create new tar file

```
adavtyan@artur-lpt:~/tmp$ tar -c -f kern.log.tar2 kern.log.tar.gz kern.log.tar kern.log
adavtyan@artur-lpt:~/tmp$ ls
kern.log  kern.log.tar  kern.log.tar2  kern.log.tar.gz
```

List the archive file

```
adavtyan@artur-lpt:~/tmp$ tar -t -f kern.log.tar2
kern.log.tar.gz
kern.log.tar
kern.log
adavtyan@artur-lpt:~/tmp$
```

Note: The tar command will recurse into subdirectories automatically when compressing and will store the path info inside the archive.

sourcemind

Archiving Extract Mode

To extract (or *unpack*) the contents of a tar archive, use tar with the ‘-x’ (“extract”) option.

tar -x [-f ARCHIVE] [OPTION]

The next example uses three options:

Option	Function
-x	Extract files from an archive
-j	Decompress with bzip2 command
-f archive	Use archive file. The argument archive will be the name of the resulting archive file

Example:

```
adavtyan@artur-lpt:~/tmp/tar$ ls -la
total 220
drwxrwxr-x 2 adavtyan adavtyan   4096 Մայ  6 09:25 .
drwxrwxr-x 3 adavtyan adavtyan   4096 Մայ  6 09:25 ..
-rw-rw-r-- 1 adavtyan adavtyan 215040 Մայ  6 09:25 kern.log.tar
adavtyan@artur-lpt:~/tmp/tar$ tar -xf kern.log.tar
adavtyan@artur-lpt:~/tmp/tar$ ls -la
total 424
drwxrwxr-x 2 adavtyan adavtyan   4096 Մայ  6 09:26 .
drwxrwxr-x 3 adavtyan adavtyan   4096 Մայ  6 09:25 ..
-rw-r----- 1 adavtyan adavtyan 176788 Ապր 26 14:55 kern.log
-rw-rw-r-- 1 adavtyan adavtyan 215040 Մայ  6 09:25 kern.log.tar
-rw-rw-r-- 1 adavtyan adavtyan  28134 Ապր 26 16:37 kern.log.tar.gz
adavtyan@artur-lpt:~/tmp/tar$ tar -tf kern.log.tar
kern.log
kern.log.tar.gz
```

Note: Add the -v flag and you will get a verbose output of the files processed, making it easier to keep track of what's happening:

sourcemind

Zip Files

The de facto archiving utility in Microsoft is the ZIP file. ZIP is not as prevalent in Linux but is well supported by the zip and unzip commands. Albeit, with tar and gzip/gunzip the same commands and options can be used interchangeably to do the creation and extraction, but this is not the case with zip. The same option has different meanings for the two different commands.

The default mode of zip is to add files to an archive and compress it.

```
zip [ OPTIONS ] [ zipfile [ file … ] ]
```

The first argument zipfile is the name of the archive to be created, after that, a list of files to be added. The following example shows a compressed archive called files.zip being created:

```
adavtyan@artur-lpt:~/tmp/tar$ zip files.zip kern.log kafka.json cp-kafka-0.sh
  adding: kern.log (deflated 84%)
  adding: kafka.json (deflated 94%)
  adding: cp-kafka-0.sh (deflated 77%)
adavtyan@artur-lpt:~/tmp/tar$ ls -la
total 344
drwxrwxr-x 2 adavtyan adavtyan   4096 Մայ  6 09:32 .
drwxrwxr-x 3 adavtyan adavtyan   4096 Մայ  6 09:25 ..
-rw-rw-r-- 1 adavtyan adavtyan   2480 Մայ  6 09:32 cp-kafka-0.sh
-rw-rw-r-- 1 adavtyan adavtyan  35976 Մայ  6 09:32 files.zip
-rw-rw-r-- 1 adavtyan adavtyan 122485 Մայ  6 09:32 kafka.json
-rw-r----- 1 adavtyan adavtyan 176788 Ապր 26 14:55 kern.log
adavtyan@artur-lpt:~/tmp/tar$ 
```

It should be noted that tar requires the -f option to indicate a filename is being passed, while zip and unzip require a filename and therefore don’t need you to inform the command a filename is being passed.

The zip command will not recurse into subdirectories by default, which is different behavior than the tar command. If you want tar like behavior, you must use the -r option to indicate recursion is to be used:

Viewing Files in the Terminal

The cat command, short for concatenate, is a simple but useful command whose functions include creating and displaying text files, as well as combining copies of text files. One of the most popular uses of cat is to display the content of text files. To display a file in the standard output using the cat command, type the command followed by the filename:

```
adavtyan@artur-lpt:~/tmp/Documents$ cat food.txt
Food is good.
```

While viewing small files with the cat command poses no problems, it is not an ideal choice for large files. The cat command doesn't provide any easy ways to pause and restart the display, so the entire file contents are dumped to the screen.

For larger files, use a pager command to view the contents. Pager commands display one page of data at a time, allowing you to move forward and backward in the file by using movement keys.

There are two commonly used pager commands:

- The less command provides a very advanced paging capability. It is usually the default pager used by commands like the man command.
- The more command has been around since the early days of UNIX. While it has fewer features than the less command, however, the less command isn't included with all Linux distributions. The more command is always available.

The more and less commands allow users to move around the document using keystroke commands. Because developers based the less command on the functionality of the more command, all of the keystroke commands available in the more command also work in the less command.

The focus of our content is on the more advanced less command. The more command is still useful to remember for times when the less command isn't available. Remember that most of the keystroke commands provided work for both commands.

Pager Movement Commands

To view a file with the less command, pass the file name as an argument:

```
less filename*
```

There are many movement commands for the less command, each with multiple possible keys or key combinations. While this may seem intimidating, it is not necessary to memorize all of these movement commands. When viewing a file with the less command, use the **H** key or **Shift+H** to display a help screen:

```
                   SUMMARY OF LESS COMMANDS

      Commands marked with * may be preceded by a number, N.
      Notes in parentheses indicate the behavior if N is given.
      A key preceded by a caret indicates the Ctrl key; thus ^K is ctrl-K.

  h  H                 Display this help.
  q  :q  Q  :Q  ZZ     Exit.
 ---------------------------------------------------------------------------

                           MOVING

  e  ^E  j  ^N  CR  *  Forward  one line   (or N lines).
  y  ^Y  k  ^K  ^P  *  Backward one line   (or N lines).
  f  ^F  ^V  SPACE  *  Forward  one window (or N lines).
  b  ^B  ESC-v      *  Backward one window (or N lines).
  z                 *  Forward  one window (and set window to N).
  w                 *  Backward one window (and set window to N).
  ESC-SPACE         *  Forward  one window, but don't stop at end-of-file.
  d  ^D             *  Forward  one half-window (and set half-window to N).
  u  ^U             *  Backward one half-window (and set half-window to N).
```

There are two ways to search in the less command: searching forward or backward from your current position.

- To start a search to look forward from your current position, use the slash **/** key. Then, type the text or pattern to match and press the **Enter** key.
- To search backward from your current position, press the question mark **?**key, then type the text or pattern to match and press the **Enter** key. The cursor moves backward to the first match it can find or reports that the pattern cannot be found.

sourcemind

Head and Tail

The head and tail commands are used to display only the first few or last few lines of a file, respectively (or, when used with a pipe, the output of a previous command). By default, the head and tail commands display ten lines of the file that is provided as an argument.

For example, the following command displays the first ten lines of the /etc/sysctl.conf file:

```
adavtyan@artur-lpt:~/tmp/Documents$ head /etc/sysctl.conf
#
# /etc/sysctl.conf - Configuration file for setting system variables
# See /etc/sysctl.d/ for additional system variables.
# See sysctl.conf (5) for information.
#

#kernel.domainname = example.com

# Uncomment the following to stop low-level messages on console
#kernel.printk = 3 4 1 3
```

Passing a number as an option will cause both the head and tail commands to output the specified number of lines, instead of the standard ten. For example to display the last five lines of the /etc/sysctl.conf file use the -5 option:

```
adavtyan@artur-lpt:~/tmp/Documents$ tail -5 /etc/sysctl.conf
# 0=disable, 1=enable all, >1 bitmask of sysrq functions
# See https://www.kernel.org/doc/html/latest/admin-guide/sysrq.html
# for what other values do
#kernel.sysrq=438

adavtyan@artur-lpt:~/tmp/Documents$
```

Live file changes can be viewed by using the -f option to the tail command-useful when you want to see changes to a file as they are happening.

Input/Output Redirection

Input/Output (I/O) redirection allows for command line information to be passed to different streams. Before discussing redirection, it is important to understand the standard streams.

Input and output in the Linux environment is distributed across three streams. These streams are:

1. *STDIN*
   Standard input, or STDIN, is information entered normally by the user via the keyboard. When a command prompts the shell for data, the shell provides the user with the ability to type commands that, in turn, are sent to the command as STDIN.
2. *STDOUT*
   Standard output, or STDOUT, is the normal output of commands. When a command functions correctly (without errors) the output it produces is called STDOUT. By default, STDOUT is displayed in the terminal window where the command is executing. STDOUT is also known as stream or channel #1.
3. *STDERR*
   Standard error, or STDERR, is error messages generated by commands. By default, STDERR is displayed in the terminal window where the command is executing. STDERR is also known as stream or channel #2.

The redirection capabilities built into Linux provide you with a robust set of tools used to make all sorts of tasks easier to accomplish. Whether you’re writing complex software or performing file management through the command line, knowing how to manipulate the different I/O streams in your environment will greatly increase your productivity.

I/O redirection allows the user to redirect STDIN so that data comes from a file and STDOUT/STDERR so that output goes to a file. Redirection is achieved by using the arrow < > characters.

sourcemind

STDOUT

STDOUT can be directed to files. To begin, observe the output of the following echo command which displays to the screen and Using the > character, the output can be redirected to a file instead:

```
adavtyan@artur-lpt:~/tmp/Documents$ echo "Line 1"
Line 1
adavtyan@artur-lpt:~/tmp/Documents$ echo "Line 1" > example.txt
adavtyan@artur-lpt:~/tmp/Documents$ ls
alpha-first.txt  example.txt  food.txt  kafka.json  new-home.txt  newhome.txt  profile.txt  red.txt  spelling.txt
adavtyan@artur-lpt:~/tmp/Documents$ cat example.txt
Line 1
```

It is important to realize that the single arrow overwrites any contents of an existing file:

```
adavtyan@artur-lpt:~/tmp/Documents$ echo "New line 1" > example.txt
adavtyan@artur-lpt:~/tmp/Documents$ cat example.txt
New line 1
```

The original contents of the file are gone, replaced with the output of the new echo command.

It is also possible to preserve the contents of an existing file by appending to it. Use two arrow >> characters to append to a file instead of overwriting it:

```
adavtyan@artur-lpt:~/tmp/Documents$  echo "Another line" >> example.txt
adavtyan@artur-lpt:~/tmp/Documents$ cat example.txt
New line 1
Another line
```

Instead of being overwritten, the output of the echo command is added to the bottom of the file.

STDERR

STDERR can be redirected similarly to STDOUT. When using the arrow character to redirect, stream #1 (STDOUT) is assumed unless another stream is specified. Thus, stream #2 must be specified when redirecting STDERR by placing the number 2 preceding the arrow > character.

To demonstrate redirecting STDERR, first observe the following command which produces an error because the specified directory does not exist:

```
adavtyan@artur-lpt:~/tmp/Documents$ ls /fake
ls: cannot access '/fake': No such file or directory
```

Note that there is nothing in the example above that implies that the output is STDERR. The output is clearly an error message, but how could you tell that it is being sent to STDERR? One easy way to determine this is to redirect STDOUT:

```
adavtyan@artur-lpt:~/tmp/Documents$ ls /fake > output.txt
ls: cannot access '/fake': No such file or directory
```

In the example above, STDOUT was redirected to the output.txt file. So, the output that is displayed can't be STDOUT because it would have been placed in the output.txt file instead of the terminal. Because all command output goes either to STDOUT or STDERR, the output displayed above must be STDERR.

The STDERR output of a command can be sent to a file:

```
adavtyan@artur-lpt:~/tmp/Documents$  ls /fake 2> error.txt
adavtyan@artur-lpt:~/tmp/Documents$ cat error.txt
ls: cannot access '/fake': No such file or directory
```

sourcemind

Redirecting Multiple Streams

It is possible to direct both the STDOUT and STDERR of a command at the same time. The following command produces both STDOUT and STDERR because one of the specified directories exists and the other does not:

```
adavtyan@artur-lpt:~/tmp/Documents$ ls /fake /etc/ppp
ls: cannot access '/fake': No such file or directory
/etc/ppp:
chap-secrets  ip-down  ip-down.d  ip-up  ip-up.d  ipv6-down  ipv6-down.d  ipv6-up  ipv6-up.d  options  options.pptp  pap-secrets  peers  resolv.conf
```

If only the STDOUT is sent to a file, STDERR is still printed to the screen and If only the STDERR is sent to a file, STDOUT is still printed to the screen:

```
adavtyan@artur-lpt:~/tmp/Documents$ ls /fake /etc/ppp > example.txt
ls: cannot access '/fake': No such file or directory
adavtyan@artur-lpt:~/tmp/Documents$ ls /fake /etc/ppp 2> error.txt
/etc/ppp:
chap-secrets  ip-down  ip-down.d  ip-up  ip-up.d  ipv6-down  ipv6-down.d  ipv6-up  ipv6-up.d  options  options.pptp  pap-secrets  peers  resolv.conf
adavtyan@artur-lpt:~/tmp/Documents$ cat error.txt
ls: cannot access '/fake': No such file or directory
adavtyan@artur-lpt:~/tmp/Documents$ cat example.txt
/etc/ppp:
chap-secrets
ip-down
ip-down.d
```

Both STDOUT and STDERR can be sent to a file by using the ampersand & character in front of the arrow > character. The &> character set means both 1> and 2> or If you don't want STDERR and STDOUT to both go to the same file, they can be redirected to different files by using both > and 2>. For example, to direct STDOUT to example.txt and STDERR to error.txt execute the following:

```
adavtyan@artur-lpt:~/tmp/Documents$ ls /fake /etc/ppp /junk /etc/sound &> all.txt
adavtyan@artur-lpt:~/tmp/Documents$ ls /fake /etc/ppp > example.txt 2> error.txt
adavtyan@artur-lpt:~/tmp/Documents$
```

STDINN

The concept of redirecting STDIN is a difficult one because it is more difficult to understand why you would want to redirect STDIN. With STDOUT and STDERR, their purpose is straightforward; sometimes it is helpful to store the output into a file for future use.

Most Linux users end up redirecting STDOUT routinely, STDERR on occasion, and STDIN very rarely.

There are very few commands that require you to redirect STDIN because with most commands if you want to read data from a file into a command, you can specify the filename as an argument to the command.

For some commands, if you don't specify a filename as an argument, they revert to using STDIN to get data. For example, consider the following cat command:

```
adavtyan@artur-lpt:~/tmp/Documents$ cat
hello
hello
how are you ?
how are you ?
goodbye
goodbye
^C
```

The first command in the example below redirects the output of the cat command to a newly created file called new.txt. This action is followed up by providing the cat command with the *new.txt* file as an argument to display the redirected text in STDOUT.

```
adavtyan@artur-lpt:~/tmp/Documents$ cat > new.txt
Hello
Bye
^C
adavtyan@artur-lpt:~/tmp/Documents$ cat new.txt
Hello
Bye
```

sourcemind

Sorting files or Input

The sort command can be used to rearrange the lines of files or input in either dictionary or numeric order. The following example creates a small file, using the head command to grab the first 3 lines of the /etc/passwd file and send the output to a file called mypasswd.

```
adavtyan@artur-lpt:~/tmp$ head -n 3 /etc/passwd > mypasswd
adavtyan@artur-lpt:~/tmp$ cat mypasswd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
adavtyan@artur-lpt:~/tmp$
```

Now we will sort the mypasswd file:

```
adavtyan@artur-lpt:~/tmp$ sort mypasswd
bin:x:2:2:bin:/bin:/usr/sbin/nologin
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
root:x:0:0:root:/root:/bin/bash
adavtyan@artur-lpt:~/tmp$
```

Three options are used to achieve:

Option	Function
-t	The -t option specifies the field delimiter. If the file or input is separated by a delimiter other than whitespace, for example a comma or colon, the -t option will allow for another field separator to be specified as an argument.
-k	The -k option specifies the field number. To specify which field to sort by, use the -k option with an argument to indicate the field number, starting with 1 for the first field.
-n	This option specifies the sort type.

Example:

```
adavtyan@artur-lpt:~/tmp$ sort -t: -n -k3 mypasswd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
adavtyan@artur-lpt:~/tmp$
```

sourcemind

Filter File Contents [grep]

The grep command can be used to filter lines in a file or the output of another command that matches a specified pattern. That pattern can be as simple as the exact text that you want to match or it can be much more advanced through the use of regular expressions.

For example, to find all the users who can log in to the system with the BASH shell, the grep command can be used to filter the lines from the /etc/passwd file for the lines containing the pattern bash:

```
adavtyan@artur-lpt:~/tmp$ grep bash /etc/passwd
root:x:0:0:root:/root:/bin/bash
adavtyan:x:1000:1000:Artur Davtyan,,,:/home/adavtyan:/bin/bash
```

In some cases, it may not be important to find the specific lines that match the pattern, but rather how many lines match the pattern. The -c option provides a count of how many lines match:

```
adavtyan@artur-lpt:~/tmp$ grep -c bash /etc/passwd
2
```

The -n option to the grep command will display original line numbers. To display all lines and their line numbers in the /etc/passwd file which contain the pattern bash:

```
adavtyan@artur-lpt:~/tmp$ grep -n bash /etc/passwd
1:root:x:0:0:root:/root:/bin/bash
41:adavtyan:x:1000:1000:Artur Davtyan,,,:/home/adavtyan:/bin/bash
```

The -v option inverts the match, outputting all lines that do not contain the pattern. To display all lines not containing nologin in the /etc/passwd file:

```
adavtyan@artur-lpt:~/tmp$ grep -v nologin /etc/passwd
root:x:0:0:root:/root:/bin/bash
sync:x:4:65534:sync:/bin:/bin/sync
speech-dispatcher:x:111:29:Speech Dispatcher,,,:/var/run/speech-dispatcher:/bin/false
whoopsie:x:112:117::/nonexistent:/bin/false
```

Filter File Contents [grep]

The -i option ignores the case (capitalization) distinctions. The following searches for the pattern the in newhome.txt, allowing each character to be uppercase or lowercase:

```
adavtyan@artur-lpt:~/tmp/Documents$ grep the newhome.txt
**Beware** of the ghost in the bedroom.
**Caution** the spirits don't like guests.
adavtyan@artur-lpt:~/tmp/Documents$ grep -i the newhome.txt
There are three bathrooms.
**Beware** of the ghost in the bedroom.
The kitchen is open for entertaining.
**Caution** the spirits don't like guests.
```

The -w option only returns lines which contain matches that form whole words. To be a word, the character string must be preceded and followed by a non-word character. Word characters include letters, digits, and the underscore character.

The following examples search for the are pattern in the newhome.txt file. The first command searches with no options, while the second command includes the -w option. Compare the outputs:

```
adavtyan@artur-lpt:~/tmp/Documents$ grep are newhome.txt
There are three bathrooms.
**Beware** of the ghost in the bedroom.
adavtyan@artur-lpt:~/tmp/Documents$ grep -w are newhome.txt
There are three bathrooms.
adavtyan@artur-lpt:~/tmp/Documents$ grep -nw are newhome.txt
1:There are three bathrooms.
adavtyan@artur-lpt:~/tmp/Documents$
```

sourcemind

Basic Regular Expressions

Regular expressions, also referred to as regex, are a collection of normal and special characters that are used to find simple or complex patterns, respectively, in files. These characters are characters that are used to perform a particular matching function in a search.

Normal characters are alphanumeric characters which match themselves. For example, an a would match an a. Special characters have special meanings when used within patterns by commands like the grep command. They behave in a more complex manner and do not match themselves.

There are both Basic Regular Expressions (available to a wide variety of Linux commands) and Extended Regular Expressions (available to more advanced Linux commands). Basic Regular Expressions include the following:

Character	**Function**
.	Any single character
[ ]	A list or range of characters to match one character If the first character within the brackets is the caret ^, it means any character not in the list
*	The previous character repeated zero or more times
^	If the first character in the pattern, the pattern must be at the beginning of the line to match, otherwise just a literal ^ character
$	If the last character in the pattern, the pattern must be at the end of the line to match, otherwise just a literal $ character

The grep command is just one of the many commands that support regular expressions. Some other commands include the more and less commands.

Note: While some of the regular expressions are unnecessarily quoted with single quotes, it is good practice to use single quotes around regular expressions to prevent the shell from trying to interpret special meaning from them.

sourcemind

The Period . Character

One of the most useful expressions is the period . character. It matches any character except for the new line character. Consider the unfiltered contents of the ~/Documents/red.txt file:

```
adavtyan@artur-lpt:~/tmp/Documents$ cat red.txt
red
reef
rot
reeed
rd
rod
roof
reed
root
reel
read
adavtyan@artur-lpt:~/tmp/Documents$ grep 'r..f' red.txt
reef
roof
```

The line does not have to be an exact match, it simply must contain the pattern, as seen here when r..t is searched for in the /etc/passwd file:

```
adavtyan@artur-lpt:~/tmp/Documents$ grep 'r..t' /etc/passwd
root:x:0:0:root:/root:/bin/bash
nm-openvpn:x:126:134:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/nologin
adavtyan@artur-lpt:~/tmp/Documents$
```

The period character can be used any number of times. To find all words that have at least four characters, the following pattern can be used:

```
adavtyan@artur-lpt:~/tmp/Documents$ grep '....' red.txt
reef
reeed
roof
reed
root
reel
read
```

The Bracket [] Character

When using the . character, any possible character could match it. In some cases, you want to specify exactly which characters you want to match, such as a lowercase alphabet character or a number character.

The square brackets [ ] match a single character from the list or range of possible characters contained within the brackets. To find all the lines in profile.txt which have a number in them, use the pattern [0123456789] or [0-9]:

```
adavtyan@artur-lpt:~/tmp/Documents$ grep '[0-9]' profile.txt
I am 37 years old.
3121991
I have 2 dogs.
123456789101112
adavtyan@artur-lpt:~/tmp/Documents$
```

Note that each possible character can be listed out [abcd] or provided as a range [a-d], as long as the range is in the correct order. For example, [d-a] wouldn't work because it isn't a valid range:

```
adavtyan@artur-lpt:~/tmp/Documents$ grep '[d-a]' profile.txt
grep: Invalid range end
adavtyan@artur-lpt:~/tmp/Documents$
```

To match a character that is not one of the listed characters, start the set with a ^ symbol. To find all the lines which contain any non-numeric characters, insert a ^ as the first character inside the brackets. This character negates the characters listed:

```
adavtyan@artur-lpt:~/tmp/Documents$ grep '[^0-9]' profile.txt
Hello my name is Joe.
I am 37 years old.
My favorite food is avocados.
I have 2 dogs.
adavtyan@artur-lpt:~/tmp/Documents$
```

sourcemind

The Asterisk * Character

The asterisk * character is used to match zero or more occurrences of a character or pattern preceding it. For example, e* would match zero or more occurrences of the letter e:

```
adavtyan@artur-lpt:~/tmp/Documents$ grep 're*d' red.txt
red
reeed
rd
reed
```

It is also possible to match zero or more occurrences of a list of characters by utilizing the square brackets. The pattern [oe]* used in the following example matches zero or more occurrences of the o character or the e character:

```
adavtyan@artur-lpt:~/tmp/Documents$ grep 'r[oe]*d' red.txt
red
reeed
rd
rod
reed
```

When used with only one other character, * isn't very helpful. Any of the following patterns would match every string or line in the file: '.*' 'e*' 'b*' 'z*' because the asterisk * character can match zero occurrences of a pattern.

```
adavtyan@artur-lpt:~/tmp/Documents$ grep 'z*' red.txt
red
reef
rot
reeed
```

To make the asterisk character useful, it is necessary to create a pattern which includes more than just the one character preceding it. For example, the results above can be refined by adding another e to make the pattern ee* effectively matching every line which contains at least one e.

```
adavtyan@artur-lpt:~/tmp/Documents$  grep 'ee*' red.txt
red
reef
reeed
reed
reel
read
```

Anchor Character

When performing a pattern match, the match could occur anywhere on the line. Anchor characters are one of the ways regular expressions can be used to narrow down search results. They specify whether the match occurs at the beginning of the line or the end of the line.

For example, the pattern root appears many times in the /etc/passwd file:

```
adavtyan@artur-lpt:~/tmp/Documents$ grep 'root' /etc/passwd
root:x:0:0:root:/root:/bin/bash
nm-openvpn:x:126:134:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/nologin
adavtyan@artur-lpt:~/tmp/Documents$
```

The caret (circumflex) ^ character is used to ensure that a pattern appears at the beginning of the line. For example, to find all lines in /etc/passwd that start with root use the pattern ^root. Note that ^ must be the first character in the pattern to be effective:

```
adavtyan@artur-lpt:~/tmp/Documents$ grep '^root' /etc/passwd
root:x:0:0:root:/root:/bin/bash
adavtyan@artur-lpt:~/tmp/Documents$
```

The second anchor character $ can be used to ensure a pattern appears at the end of the line, thereby effectively reducing the search results. To find the lines that end with an r in the alpha-first.txt file, use the pattern r$:

```
adavtyan@artur-lpt:~/tmp/Documents$ cat alpha-first.txt
A is for Animal
B is for Bear
C is for Cat
D is for Dog
E is for Elephant
F is for Flower
adavtyan@artur-lpt:~/tmp/Documents$ grep 'r$' alpha-first.txt
B is for Bear
F is for Flower
adavtyan@artur-lpt:~/tmp/Documents$
```

Again, the position of this character is important. The $ must be the last character in the pattern to be effective as an anchor.

The Backslash \ Character

In some cases, you may want to match a character that happens to be a special regular expression character. For example, consider the following:

```
adavtyan@artur-lpt:~/tmp/Documents$ cat new-home.txt
Thanks for purchasing your new home!!

**Warning** it may be haunted.

There are three bathrooms.

**Beware** of the ghost in the bedroom.

The kitchen is open for entertaining.

**Caution** the spirits don't like guests.

Good luck!!!
adavtyan@artur-lpt:~/tmp/Documents$ grep 're*' new-home.txt
Thanks for purchasing your new home!!
**Warning** it may be haunted.
There are three bathrooms.
**Beware** of the ghost in the bedroom.
The kitchen is open for entertaining.
**Caution** the spirits don't like guests.
```

In the output of the grep command above, the search for re* matched every line which contained an r followed by zero or more of the letter e. To look for an actual asterisk * character, place a backslash \ character before the asterisk * character:

```
adavtyan@artur-lpt:~/tmp/Documents$ grep 're\*' new-home.txt
**Beware** of the ghost in the bedroom.
adavtyan@artur-lpt:~/tmp/Documents$
```

Extended Regular Expressions

The use of extended regular expressions often requires a special option be provided to the command to recognize them. Historically, there is a command called egrep, which is similar to grep, but can understand extended regular expressions. Now, the egrep command is deprecated in favor of using grep with the -E option.

The following regular expressions are considered extended:

Character	Function
?	Matches previous character zero or one time, so it is an optional character
+	Matches previous character repeated one or more times
|	Alternation or like a logical "or" operator

To match “colo” followed by zero or one u character followed by an r character:

```
adavtyan@artur-lpt:~/tmp/Documents$ grep -E 'colou?r' spelling.txt
American English: Do you consider gray to be a color or a shade?
British English: Do you consider grey to be a colour or a shade?
```

To match one or more e characters:

```
adavtyan@artur-lpt:~/tmp/Documents$ grep -E 'e+' red.txt
red
reef
reeed
reed
reel
read
```

To match either gray or grey:

```
adavtyan@artur-lpt:~/tmp/Documents$ grep -E 'gray|grey' spelling.txt
American English: Do you consider gray to be a color or a shade?
British English: Do you consider grey to be a colour or a shade?
```

Memory usage check

The motherboard typically has slots where random-access memory (RAM) can be connected to the system. The 32-bit architecture systems can use up to 4 gigabytes (GB) of RAM, while 64-bit architectures are capable of addressing and using far more RAM.

In some cases, the RAM a system has might not be enough to handle all of the operating system requirements. Once invoked, programs are loaded in RAM along with any data they need to store, while instructions are sent to the processor when they execute.

To view the amount of RAM in your system, including the swap space, execute the free command. The free command has a -m option to force the output to be rounded to the nearest megabyte (MB) and a -g option to force the output to be rounded to the nearest gigabyte (GB):

```
adavtyan@artur-lpt:~/tmp/Documents$ free -m
              total        used        free      shared  buff/cache   available
Mem:          31911        4816       22301         332        4793       26307
Swap:           979           0         979
```

If you want to monitor memory usage over time with the free command, then you can execute it with the -s option (how often to update) and specify that number of seconds. For example, executing the following free command would update the output every ten seconds:

```
adavtyan@artur-lpt:~/tmp/Documents$ free -m -s 20
              total        used        free      shared  buff/cache   available
Mem:          31911        5423       20729         648        5758       25385
Swap:           979           0         979

              total        used        free      shared  buff/cache   available
Mem:          31911        5376       20776         648        5758       25431
Swap:           979           0         979
```

Processes

The kernel provides access to information about active processes through a pseudo filesystem that is visible under the /proc directory. Hardware devices are made available through special files under the /dev directory, while information about those devices can be found in another pseudo filesystem under the /sys directory.

Pseudo *filesystems* appear to be real files on disk but exist only in memory. Most pseudo file systems such as /proc are designed to appear to be a hierarchical tree off the root of the system of directories, files and subdirectories, but in reality only exist in the system's memory, and only appear to be resident on the storage device that the root file system is on.

The /proc directory not only contains information about running processes, as its name would suggest, but it also contains information about the system hardware and the current kernel configuration.

The /proc directory is read, and its information utilized by many different commands on the system, including but not limited to top, free, mount, umount and many many others. It is rarely necessary for a user to mine the /proc directory directly—it's easier to use the commands that utilize its information.

```
adavtyan@artur-lpt:~/tmp/Documents$ ls /proc
1     1060  1180  13903  147    159    17654  2157   2232  2444  2593  2706  2914   2989   3041   33     34821  36724  39    4662  53   646  743  89                 dma          locks         sys
10    1065  1182  13905  1472   16     18     2165   2242  2459  26    2708  29247  2990   3056   3306   34848  36886  4     4675  54   65   744  9                  driver       mdstat        sysrq-trigger
1008  1070  1192  13925  148    160    180    2170   2246  2476  2611  273   2926   2991   30627  332    34849  36933  40    4691  541  650  745  90                 execdomains  meminfo       sysvipc
1032  1071  12    13927  1489   16061  18002  2171   2252  2478  2613  274   2930   2992   308    3326   35     36941  41    47    545  654  75   91                 fb           misc          thread-self
1033  1072  1200  13940  149    161    189    2177   2256  2479  2618  275   2935   2995   309    33272  35013  37020  4151  4736  546  66   751  917                filesystems  modules       timer_list
1034  1073  1202  13941  15     162    1911   2179   2262  2480  2620  2765  2943   29953  30920  33333  35090  37027  42    4771  553  662  76   934                fs           mounts        tty
1040  1074  1206  14     1504   163    1913   2180   23    2481  2625  278   2951   29972  3093   3346   35152  37111  44    48    554  667  77   935                interrupts   mtrr          uptime
1041  1075  1208  140    151    164    1915   2182   2375  2482  2627  28    29539  2999   3096   33521  35216  37123  45    485   56   668  78   982                iomem        net           version
1042  1076  1211  141    153    16416  192    2185   2376  2483  2644  2832  29613  3      3097   33538  35441  37125  4544  486   57   68   80   acpi               ioports      pagetypeinfo  version_signature
1043  11    1215  14450  154    16424  1923   2186   2377  2484  2649  2849  29639  30     310    33599  35456  37336  4545  4917  58   688  81   asound             irq          partitions    vmallocinfo
1046  1105  1216  14453  15544  1644   1926   2193   2380  2485  2651  2854  2965   3002   311    3363   35649  37406  46    493   59   69   82   buddyinfo          kallsyms     pressure      vmstat
1048  1106  1217  145    156    165    1927   22     2385  2486  2665  2860  2968   3003   31249  3366   35736  37532  4617  494   6    70   83   bus                kcore        sched_debug   zoneinfo
1049  1156  1225  14527  1560   1657   1929   2208   24    2487  2669  2867  2975   3004   3146   33670  36     37806  462   495   60   705  84   cgroups            keys         schedstat
1050  1157  1295  14536  157    17     1964   22118  2406  2489  2671  2881  2978   3005   3152   33681  36193  37809  4624  498   609  71   856  cmdline            key-users    scsi
1052  1160  13    14556  1572   17091  2      2214   2422  2492  2678  29    2979   30097  3163   337    36370  37897  4625  50    62   72   86   consoles           kmsg         self
1053  1162  1319  14557  15759  17142  20     2220   2425  2495  2683  2904  2980   302    31909  3376   36467  37937  4629  500   63   735  866  cpuinfo            kpagecgroup  slabinfo
1056  1165  1352  146    15763  172    207    2225   2428  2503  2688  2908  2982   30205  32     338    36514  37938  4631  5099  637  738  87   crypto             kpagecount   softirqs
1057  1169  139   14638  158    173    21     2230   2431  2546  2692  2909  2985   3029   32549  34     36630  38     4633  51    64   74   88   devices            kpageflags   stat
1059  1173  13901 1464   15818  17366  2154   22306  2440  2547  27    2911  2988   304    32684  3405   36632  38144  4636  52    642  742  881  diskstats          loadavg      swaps
```

The output shows a variety of named and numbered directories. There is a numbered directory for each running process on the system, where the name of the directory matches the process ID (PID) for the running process.

Process Hierarchy

When the kernel finishes loading during the boot procedure, it starts the init process and assigns it a PID of 1. This process then starts other system processes, and each process is assigned a PID in sequential order.

As either of the init processes starts up other processes, they, in turn, may start up processes, which may start up other processes, on and on. When one process starts another process, the process that performs the starting is called the parent process and the process that is started is called the child process. When viewing processes, the parent PID is labeled PPID.

When the system has been running for a long time, it may eventually reach the maximum PID value, which can be viewed and configured through the /proc/sys/kernel/pid_max file. Once the largest PID has been used, the system "rolls over" and continues seamlessly by assigning PID values that are available at the bottom of the range.

Processes can be “mapped” into a family tree of parent and child couplings. If you want to view this tree, the command pstree displays it:

```
adavtyan@artur-lpt:~/tmp/Documents$ pstree
systemd─┬─ModemManager───2*[{ModemManager}]
        ├─NetworkManager───2*[{NetworkManager}]
        ├─accounts-daemon───2*[{accounts-daemon}]
        ├─acpid
        ├─apache2───2*[apache2───26*[{apache2}]]
        ├─at-spi-bus-laun─┬─dbus-daemon
        │                 └─3*[{at-spi-bus-laun}]
        ├─at-spi2-registr───2*[{at-spi2-registr}]
        ├─avahi-daemon───avahi-daemon
        ├─bdepsecd───30*[{bdepsecd}]
        ├─bdlogd───11*[{bdlogd}]
        ├─bdmond───3*[{bdmond}]
        ├─bdregd───2*[{bdregd}]
        ├─bdsrvscand.bin───95*[{bdsrvscand.bin}]
        ├─bdupdated───4*[{bdupdated}]
```

Viewing Process Snapshot

Another way of viewing processes is with the ps command. By default, the ps command only shows the current processes running in the current shell. Ironically, even though you are trying to obtain information about processes, the ps command includes itself in the output:

```
adavtyan@artur-lpt:~/tmp/Documents$ ps
  PID TTY          TIME CMD
16424 pts/1    00:00:01 bash
35244 pts/1    00:00:00 ps
```

If you run ps with the option --forest, then, similar to the pstree command, it shows lines indicating the parent and child relationship:

```
adavtyan@artur-lpt:~/tmp/Documents$ ps --forest
  PID TTY          TIME CMD
16424 pts/1    00:00:01 bash
35425 pts/1    00:00:00  \_ ps
```

To be able to view all processes on the system execute either the ps aux command or the ps -ef command:

```
adavtyan@artur-lpt:~/tmp/Documents$ ps -ef | head -n 4
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 Uwj05 ?        00:00:05 /sbin/init splash
root           2       0  0 Uwj05 ?        00:00:00 [kthreadd]
root           3       2  0 Uwj05 ?        00:00:00 [rcu_gp]
```

An administrator may be more concerned about the processes of another user. There are several styles of options that the ps command supports, resulting in different ways to view an individual user's processes. To use the traditional UNIX option to view the processes of a specific user, use the -u option:

```
adavtyan@artur-lpt:~/tmp/Documents$ ps -u root
    PID TTY          TIME CMD
      1 ?        00:00:05 systemd
      2 ?        00:00:00 kthreadd
      3 ?        00:00:00 rcu_gp
```

Viewing Process in Real Time

Whereas the ps command provides a snapshot of the processes running at the instant the command is executed, the top command has a dynamic, screen-based interface that regularly updates the output of running processes.

By default, the output of the top command is sorted by the percentage % of CPU time that each process is currently using, with the higher values listed first, meaning more CPU-intensive processes are listed first:

```
top - 15:04:51 up 20:09,  1 user,  load average: 0.89, 0.56, 0.39
Tasks: 452 total,   2 running, 450 sleeping,   0 stopped,   0 zombie
%Cpu(s):  5.9 us,  1.9 sy,  0.0 ni, 91.3 id,  0.1 wa,  0.0 hi,  0.7 si,  0.0 st
MiB Mem :  31911.5 total,  20847.7 free,   5350.6 used,   5713.2 buff/cache
MiB Swap:    980.0 total,    980.0 free,      0.0 used.  25498.4 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   2881 adavtyan  20   0 4739944 587732 124444 R  24.2   1.8  24:58.24 gnome-shell
  14527 adavtyan  20   0   32.5g 226416 106496 S  22.8   0.7  18:43.43 chrome
  36772 adavtyan  20   0   36.4g 103296  79448 S  10.9   0.3   0:00.33 chrome
  29953 adavtyan  20   0   36.6g 287156 234396 S  10.6   0.9   2:03.93 chrome
   2230 root     -51   0       0      0      0 S   8.6   0.0   8:39.87 irq/147-nvidia
```

There is an extensive amount of interactive commands that can be executed from within the running top program. Use the **H** key to view a full list.

One of the advantages of the top command is that it can be left running to stay on top of processes for monitoring purposes. If a process begins to dominate, or run away with the system, then by default it will appear at the top of the list presented by the top command.

```
adavtyan@artur-lpt:~/tmp/Documents$ cat /proc/loadavg
0.58 0.57 0.42 1/1651 37000
```

- *Load Average:* *The first three numbers in this file indicate the load average over the last one, five and fifteen minute intervals.*
- *Number of processes:* *The fourth value is a fraction which shows the number of processes currently executing code on the CPU 1 and the total number of processes 1651.*
- *Last PID :* *The fifth value is the last PID value that executed code on the CPU.*

Log Files

As the kernel and various processes run on the system, they produce output that describes how they are running. Some of this output is displayed as standard output and error in the terminal window where the process was executed, though some of this data is not sent to the screen. Instead, it is written to various files. This information is called log data or log messages.

Log files are useful for many reasons; they help troubleshoot problems and determine whether or not unauthorized access has been attempted.

Some processes can log their own data to these files, other processes rely on a separate process (a daemon) to handle these log data files.

In yet more recent distributions, those based on systemd, the logging daemon is named journald, and the logs are designed to allow for mainly text output, but also binary. The standard method for viewing journald-based logs is to use the journalctl command.

You can view the contents of various log files using two different methods. First, as with most other files, you can use the cat command, or the less command to allow for searching, scrolling and other options.

The second method is to use the journalctl command on systemd-based systems, mainly because the /var/log/journal file now often contains binary information and using the cat or less commands may produce confusing screen behavior from control codes and binary items in the log files.

Although most log files contain text as their contents, which can be viewed safely with many tools, other files such as the /var/log/btmp and /var/log/wtmp files contain binary. By using the file command, users can check the file content type before they view it to make sure that it is safe to view. The following file command classifies /var/log/wtmp as data, which usually means the file is binary:

```
adavtyan@artur-lpt:~/tmp/Documents$ file /var/log/wtmp
/var/log/wtmp: dBase III DBT, version number 0, next free block index 2
```

sourcemind

Log Files

Regardless of what the daemon process being used, the log files themselves are almost always placed into the /var/log directory structure. Although some of the file names may vary, here are some of the more common files to be found in this directory:

File	Contents
boot.log	Messages generated as services are started during the startup of the system.
cron	Messages generated by the crond daemon for jobs to be executed on a recurring basis.
dmesg	Messages generated by the kernel during system boot up.
maillog	Messages produced by the mail daemon for e-mail messages sent or received.
messages	Messages from the kernel and other processes that don't belong elsewhere. Sometimes named syslog instead of messages after the daemon that writes this file.
secure	Messages from processes that required authorization or authentication (such as the login process).
journal	Messages from the default configuration of the systemd-journald.service; can be configured in the /etc/journald.conf file amongst other places.
Xorg.0.log	Messages from the X Windows (GUI) server.

sourcemind

Thank you for your attention !

Q&A