**“Київський фаховий коледж зв’язку”**

**Циклова комісія комп’ютерної та програмної інженерії**

**ЗВІТ ПО ВИКОНАННЮ**

**ЛАБОРАТОРНОЇ РОБОТИ №6**

**з дисципліни: «Операційні системи»**

**Тема: “Команди Linux для архівування та стиснення даних. Робота з текстом”**

**Виконавли студенти**

**групи КСМ-23Б**

**VelikieBomji: Бережний Т.І,**

**Лобода Д.Р. Чухманенко М.В.**

**Перевірила викладач**

**Сушанова В.С.**

**Київ 2024**

**Тема: “Команди Linux для архівування та стиснення даних. Робота з текстом”**

**Мета роботи:**

1. Отримання практичних навиків роботи з командною оболонкою Bash.
2. Знайомство з базовими командами для архівування та стиснення даних.
3. Знайомство з базовими діями при роботі з текстом у терміналі.

**Матеріальне забезпечення занять:**

1. ЕОМ типу IBM PC.

2. ОС сімейства Windows та віртуальна машина Virtual Box (Oracle).

3. ОС GNU/Linux (будь-який дистрибутив).

4. Сайт мережевої академії Cisco netacad.com та його онлайн курси по Linux

**Короткі теоретичні відомості:**

**Introduction**

There are many reasons why you would want to compress files and directories on a computer. Some of the more straightforward benefits are conserving disk space and using less bandwidth for network communication.

In this guide, we will discuss some of the different methods of compressing data and talk a bit about some of the trade-offs of various methods. We will also touch on some associated operations, like archiving, which make our compression tools much more flexible.

We will be demoing these tools on an Ubuntu 12.04 VPS instance, but they will operate pretty much exactly the same on any modern Linux distribution.

**Compression and Archival Basics**

Before we jump in to the actual tools we'll be using, we should define our terms and discuss some of the different characteristics of compression and archiving techniques.

Compression is a way of reducing the size of a file on disk using different algorithms and mathematical calculations. Files are formatted in certain ways that make their general structure somewhat predictable, even if their content varies. Furthermore, content itself is often repeated. Both of these areas represent opportunities to employ compression techniques.

Lossy and Lossless Compression

When discussing compression in regards to computers and file types, the same terms can mean a few different things depending on context. Let's take for example an MP3 music file. An MP3 is a compressed sound file that is used to create a smaller file from a larger source music file.

This type of compression is fundamentally different than what we will be talking about in this guide. This is because an MP3 is created by analysing the waveform of the audio file and basically figuring out which data it can throw away permanently while still retaining the spirit or general sound of the original.

This is called a lossy compression method because it really does lose the information from the original file that does not make it into the MP3. You can not convert the MP3 back into the same source file later on.

The compression may be unnoticeable to the user, but it does not contain all of the relevant information of the original. The higher the compression ratio, the more the compression will start to affect significant portions of the audio.

Another example of this is JPEG images. The more they are compressed, the more significant data is lost and the more the compression will be visible. A JPEG compression utility will try to find fields of color that are close enough to one another and replace the entire field with a single color. The greater the compression ratio that is used, the larger the range of colors is that will be blanketed in this manner.

Alternatively, a lossless compression method creates a file smaller than the original that can be used to reconstruct the original file. Lossless compression is the type that we will cover in this guide. This type of compression does not use approximations to compress data, and instead uses certain algorithms to recognize repeated portions of a file. It removes these and replaces them with a placeholder. It continues on and replaces later occurrences of the pattern with references to the same placeholder.

This allows the computer to store the information on less disk space. Think of this process as creating a list of variables that define blocks of data, and then using those variables later on to fill in the program. These are actually the two stages that every lossless compression technique uses: map highly repeated values to something which is smaller that can be easily referenced and then change the occurrences of each of those values with the reference.

Furthermore, modern lossless compression techniques are said to be adaptive. This means that they do not analyse the entire input file at the beginning and create the "dictionary" of reference substitutions from that. Instead, they analyze the file as they go and rewrite the dictionary based on what data is actually repeated. The dictionary progressively gets more efficient as the process continues.

Archival Background

The idea of "archiving" data generally means backing it up and saving it to a secure location, often in a compressed format. An "archive" on a Linux server in general has a slightly different meaning. Usually it refers to a tar file.

Historically, data from servers was often backed up onto tape archives, which are magnetic tape devices that can be used to store sequential data. This is still the preferred backup method for some industries. In order to do this efficiently, the tar program was created so that you can address and manipulate many files in a filesystem, with intact permissions and metadata, as one file. You can then extract a file or the entire filesystem from the archive.

Basically, a tar file is a file format that creates a convenient way to distribute, store, back up, and manipulate groups of related files. We will be talking about archives in this guide too because archives are often compressed during the archival process in order to store data in a more efficient manner.

**Comparing the Different Compression Tools**

Linux has a number of different compression tools available. They each make sacrifices in certain areas and each have their specific strengths. We will bias ourselves towards compression schemes that work with tar because they will be much more flexible than other methods.

gzip Compression

The gzip tool is typically categorized as the "classic" method of compressing data on a Linux machine. It has been around since 1992, is still in development, and still has many things going for it.

The gzip tool uses a compression algorithm know as "DEFLATE", an algorithm also used in other popular technologies like the PNG image format, the HTTP web protocol, and the SSH secure shell protocol.

One of its main advantages is speed. It can both compress and decompress data at a much higher rate than some competing technologies, especially when comparing each utility's most compact compression formats. It is also very resource efficient in terms of memory usage during compression and decompression and does not seem to require more memory when optimizing for best compression.

Another consideration is compatibility. Since gzip is such an old tool, almost all Linux systems, regardless of age, will have the tool available to handle the data.

Its biggest disadvantage is that it compresses data less thoroughly than some other options. If you are doing a lot of quick compressions and decompressions, this might be a good format for you, but if you plan to compress once and store the file, then other options might have advantages.

Typically, gzip files are stored with a .gz extension. You can compress files with gzip by typing a command like this:

*gzip sourcefile*

This will compress the file and change the name to sourcefile.gz on your system.

If you would like to recursively compress an entire directory, you can pass the -r flag like this:

*gzip -r directory1*

This will move down through a directory and compress each file individually. This is usually not preferred, and a better result can be achieved by archiving the directory and compressing the resulting file as a whole, which we'll show how to do shortly.

To find out more information about the compressed file, you can use the -l flag, which will give you some stats:

*gzip -l test.gz*

compressed uncompressed ratio uncompressed\_name

5133 14073 63.7% test

If you need to pipe the results to another utility, you can tell gzip to send the compressed file to standard out by using the -c flag. We will simply pipe it directly into a file again in this example:

*gzip -c test > test.gz*

You can adjust the compression optimization by passing a numbered flag between 1 and 9. The -1 flag (and its alias --fast) represent the fastest, but least thorough compression. The -9 flag (and its alias --best) represents the slowest and most thorough compression. The default option is -6, which is a good middle ground.

*gzip -9 compressme*

To decompress a file, you simply pass the -d flag to gzip (there are also aliases like gunzip, but they do the same thing):

*gzip -d test.gz*

bzip2 Compression

Another common compression format and tool is bzip2. While somewhat more modern than gzip, being first introduced in 1996, bzip2 is very heavily implemented as the traditional alternative to gzip.

While gzip relies on the "DEFLATE" algorithm, bzip2 is an implementation of an algorithm called the "Burrows-Wheeler algorithm". This difference in methodology results in a set of strengths and weaknesses that is quite different from gzip.

The most important trade off for most users is greater compression at the cost of longer compression time. The bzip2 tools can create significantly more compact files than gzip, but take much longer to achieve those results due to a more complex algorithm.

Luckily, the decompression time does not suffer as much as the compression time, so it may be advantageous to distribute files using the bzip2 format since you will only suffer the time penalty during compression and be able to distribute smaller files that can be decompressed in a reasonable amount of time. The decompression time is still much greater than gzip, but does not have as big of an impact as the compression operation.

Another thing to keep in mind is that the memory requirements are greater than gzip. This won't have an affect on most machines, but on small embedded devices, this may affect your choice. You can optionally pass a -s flag, which will cut the memory requirements roughly in half, but will also lead to a lower compression ratio.

Files compressed with this mechanism are generally given a .bz2 file extension.

To create a bzip2 compressed file, you can simple type something like:

*bzip2 afile*

This will compress the file and give it the name "afile.bz2".

As mentioned above, you can pass the -s flag to denote that the utility should operate in a reduced memory mode. This won't compress as well, but it will not require as many resources.

*bzip2 -s afile*

While bzip2 implements numbered flags, they mean something somewhat different than they do with gzip. Here, they represent the block size that the utility manages to implement its compression, so this is more a measurement of memory usage vs compression size, rather than time vs compression size. The default behavior is the -9 flag, which means high memory usage (relatively) but greater compression.

*bzip2 -1 file*

To decompress a bzip compressed file, you can pass the -d flag:

*bzip2 -d file.bz2*

This will give back an uncompressed file called "file".

xz Compression

A relative newcomer in the space is the xz compression mechanism. This compression tool was first released in 2009, and has gained a steady following ever since.

The xz compression utilities leverage a compression algorithm known as LZMA2. This algorithm has a greater compression ratio than the previous two examples, making it a great format when you need to store data on limited disk space. It creates smaller files.

This again comes at a cost, in most of the same areas that bzip2 suffers. While the compressed files that xz produces are smaller than the other utilities, it takes significantly longer to do the compression. For instance, with the heavy compression flags on a fairly large file, gzip may require around half a minute, bzip2 might be somewhere around a minute, and xz may take around four or five minutes.

The xz compression tools also take a hit in the memory requirements, sometimes up to an order of magnitude over the other methods. If you are on a system with abundant memory, this might not be a problem, but this is a consideration to keep in mind.

While the compression time might be much longer than is preferable, the decompression time is actually relatively good. While it never approaches gzip in terms of decompression speed, it is usually significantly faster at decompressing than bzip2. The memory usage for decompressing isn't quite as astronomical either (but is still rather high comparatively).

This set of advantages and disadvantages make this a great format for distributing files like software. You will have to eat the compression time penalty up front, but the consumers of your file will benefit quite a lot. They will have a compact file which will decompress quickly.

Another hidden disadvantage of this format is that it might not be supported on some older systems due to its age. If you are going for maximum compatibility, you may be forced to look elsewhere.

Files created in this format generally take an extension of .xz.

To compress a file, simply call the utility without any arguments:

*xz file*

This will process the file and produce a file called "file.xz".

To list statistics about the compression of the file, you can pass the -l flag on a compressed file:

*xz -l test.xz*

Strms Blocks Compressed Uncompressed Ratio Check Filename

1 1 5,016 B 13.7 KiB 0.356 CRC64 test.xz

If you need to send the compressed output to standard out, you can signal that to the utility with the -c flag. Here we can again direct it straight back into a file:

*xz -c test > test.xz*

For the numbered flags, xz uses the lower numbers to indicate faster compression. In fact, it has a -0 flag for the fastest preset. The -6 flag is the default and is a good middle ground for most use cases. If you really need to push the compression for larger files, you can use the higher flags, which may take a very long time but may show some gains.

If you need even more compression and don't care at about time, memory requirements, etc., you can use the -e flag, which uses an alternate "extreme" compression variant. This can also modify its performance with numeric flags:

*xz -e -9 large\_file*

This will take a long time and in the end, may not show very significant gains, but if you need that functionality, the option is available.

To decompress files, you pass the -d flag again:

*xz -d large\_file.xz*

This will decompress the data into a file called "large\_file".

**Using Tar Archiving with Compression**

While the individual compression methods are useful in their own right, most often you will see them paired with tar to compress archives of files. This allows us to preserve directory structures, permissions, etc. of the files we wrap up.

The tar command is actually very upfront about this relationship. It includes command line flags that can be used to automatically call an associated compression tool after the archival process is complete, all in one step.

Using tar with gzip

To create a tar archive that is then compressed with the gzip utility, you can pass the -z flag, which indicates that you wish to use gzip compression on top of the archive. Actually, tar flags don't actually require the leading "-" like most tools. A common idiom for accomplishing zipped archives is this:

*tar czvf compressed.tar.gz directory1*

This will create an archive (-c) from a directory called "directory1". It will create verbose output, compress the resulting archive with gzip, and output to a file called "compressed.tar.gz" (a tar file that has been gzipped).

Once the file is created, we can peek inside by using the -t flag instead of the creation flag:

*tar tzvf compressed.tar.gz*

drwxr-xr-x demouser/demouser 0 2014-03-19 18:31 directory1/

-rw-r--r-- demouser/demouser 5458 2014-03-19 18:31 directory1/httpd.conf.orig

-rw-r--r-- demouser/demouser 2295 2014-03-19 18:31 directory1/nginx.conf.orig

-rw-r--r-- demouser/demouser 5458 2014-03-19 18:21 directory1/httpd.conf

To later decompress the file and expand the archive, you can use the -x flag:

*tar xzvf compressed.tar.gz*

This will recreate the directory structure in the current directory.

Using tar with bzip2

To use archiving with bzip2, you can replace the -z flag, which is gzip-specific, with the -j flag.

This means that the zipped archive creation command gets modified to this:

*tar cjvf bzipcompressed.tar.bz2 directory2*

Again, you can look at the files contained in the archive by passing the -t flag:

*tar tjvf bzipcompressed.tar.bz2*

drwxr-xr-x demouser/demouser 0 2014-03-19 18:31 directory2/

-rw-r--r-- demouser/demouser 5458 2014-03-19 18:31 directory2/httpd.conf.orig

-rw-r--r-- demouser/demouser 2295 2014-03-19 18:31 directory2/nginx.conf.orig

-rw-r--r-- demouser/demouser 5458 2014-03-19 18:21 directory2/httpd.conf

You can extract the files and directory structure into the current directory by typing:

tar xjvf bzipcompressed.tar.bz2

Using tar with xz

Any remotely recent versions of tar have added similar functionality for xz compression. These follow the exact same format using the -J flag.

*tar cJvf xzcompressed.tar.xz directory3*

To display info, use the same mechanism:

*tar tJvf xzcompressed.tar.xz*

drwxr-xr-x demouser/demouser 0 2014-03-19 18:31 directory3/

-rw-r--r-- demouser/demouser 5458 2014-03-19 18:31 directory3/httpd.conf.orig

-rw-r--r-- demouser/demouser 2295 2014-03-19 18:31 directory3/nginx.conf.orig

-rw-r--r-- demouser/demouser 5458 2014-03-19 18:21 directory3/httpd.conf

Follow the same patterns to extract:

*tar xJvf xzcompressed.tar.xz*

This will give you your full directory structure back intact.

**Conclusion**

Hopefully, you now have enough information to make an informed decision as to which compression method to favor in different circumstances. All of the compression schemes that we discussed in this post have very attractive strengths depending on the specific requirements of your situation.

It is important to be aware of the performance drawbacks and compatibility issues that may be inherent with each solution. How much weight you give these concerns depends entirely on the machines you are operating on and what kind of clients you must support. Most modern machines should not have to pay too much attention to these details, but they can cause issues if you blindly implement a compression type when interacting with older machines..

**Завдання для попередньої підготовки:**

1. \*Прочитайте короткі теоретичні відомості до лабораторної роботи та зробіть невеликий словник базових англійських термінів з питань призначення команд та їх параметрів.
2. Вивчіть матеріали онлайн-курсу академії Cisco “NDG Linux Essentials”:

* Chapter 09 - Archiving and Compression
* Chapter 10 - Working With Text

1. Пройдіть тестування у курсі NDG Linux Essentials за такими темами:

* Chapter 09 Exam
* Midterm Exam (Modules 1 - 9) ***буде окреме завдання в гугл-класі***
* Chapter 10 Exam

1. На базі розглянутого матеріалу дайте відповіді на наступні питання:
   1. \*Яке призначення команд *tar*, *xz*, *zip*, *bzip*, *gzip*? Зробіть короткий опис кожної команди та виділіть їх основні параметри. Яким чином їх можна встановити.
   2. \*\*Наведіть три приклади реалізації архівування та стискання даних різними командами.
   3. \*Яке призначення команд cat, less, more, head and tail? Зробіть короткий опис кожної команди та виділіть їх основні параметри. Яким чином їх можна встановити
   4. \*\*Поясніть принципи роботи командної оболонки з каналами, потоками та фільтрами
   5. \*Яке призначення команди grep?
2. Підготувати в електронному вигляді початковий варіант звіту:

* Титульний аркуш, тема та мета роботи
* Словник термінів
* Відповіді на п.4.1 та п.4.5 з завдань для попередньої підготовки

**Хід роботи:**

* 1. Початкова робота в CLI-режимі в Linux ОС сімейства Linux:
  2. Запустіть операційну систему Linux Ubuntu. Виконайте вхід в систему та запустіть термінал ***(якщо виконуєте ЛР у 401 ауд.)***.
  3. Запустіть віртуальну машину Ubuntu\_PC ***(якщо виконуєте завдання ЛР через академію netacad)***
  4. Запустіть свою операційну систему сімейства Linux ***(якщо працюєте на власному ПК та її встановили)*** та запустіть термінал.
  5. Опрацюйте всі приклади команд, що представлені у лабораторних роботах курсу ***NDG Linux Essentials - Lab 9: Archiving and Compression*** та ***Lab 10: Working With Text.*** Створіть таблицю для опису цих команд

| Назва команди | Її призначення та функціональність |
| --- | --- |
| mkdir mybackups | Створення нової директорії **mybackups** у домашньому каталозі користувача |
| tar -cvf mybackups/udev.tar /etc/udev | Команда **tar** використовується для об’єднання кількох файлів в один файл. В даному випадку вміст директорії **/etc/udev** буде збережено в архів **udev.tar** у директорії **mybackups**. Параметр **-c** повідомляє команді tar створити файл tar. Параметр **-v** означає "verbose", який наказує команді tar продемонструвати, що вона робить. Параметр **-f** використовується для вказівки назви файлу tar. |
| tar -cf | перегляд вмісту архіву |
| tar -xf | витягти вміст архіву |
| bzip2 | створення архіву tar зі стисненням bzip |
| tar -czf archive.tar.gz k | * c — створити новий архів. * z — стиснути архів за допомогою gzip. * f — вказати ім'я файлу архіву (в даному випадку archive.tar.gz).   Ця команда створить архів archive.tar.gz, який міститиме файл k. |
| tar -xzf archive.tar.gz k | розпаковка gzip для tar   * x — розпакувати архів. * z — розпакувати стиснений файл за допомогою gzip. * f — вказати ім'я файлу архіву, який потрібно розпакувати. |
| tar -xjf | розархівування стисненого за допомогою bzip файлу |
| gzip | стиснення файлу |
| gzip -d | розпаковка файлу |
| echo “Greetings” > mymessage | знищення змісту файлу та запис Greetings у файл mymessage |
| >> | доповнює файл |
| find | здійснює пошуук файлів за різними парметрами |
| find /etc -name hosts 2> err.txt | виводить повідомлення про помилки у файл err.txt |
| find /etc -name hosts > std.out 2> std.err | перенаправляє sdout та stderr в 2 різні файли |
| wc | команда для аналізу файлів |
| cut | вилучення секцій або блоків символів зі строк текстового файлу |
| grep | пошук в текстовому файлі |
| more, less | ці команди дозволяють передивлятися данні по сторінці |
| head | відображає перші 10 строк файлу |
| tail | відображає останні 10 строк файлу |
|  |  |

**Примітка:** **Скріншоти** виконання команд в терміналі можна **не представляти**, достатньо **коротко описати команди в таблиці**.

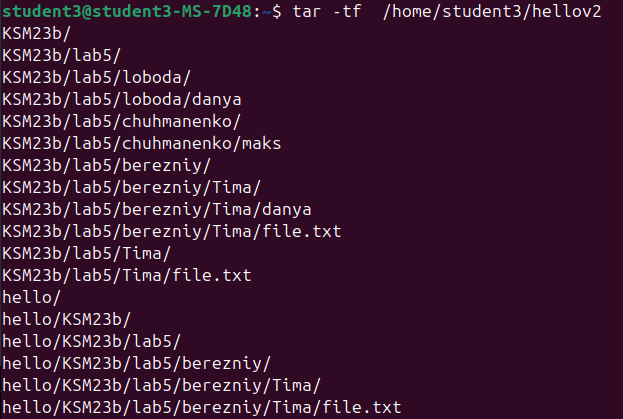
* 1. Ознайомтесь з командою tar та за її допомогою виконати у терміналі наступні дії:
* створити файл з розширенням .tar;



* створити файл з розширенням .tar, що складається з декількох файлів і каталогів одночасно;



* перегляду вмісту файлу;



* витягти вміст файлу tar;



* створити архівний файл tar, стиснений за допомогою bzip;



* витягти вміст файлу tar bzip;



* створити архівний tar файл, стисненого за допомогою gzip;





* витягти вміст файлу tar gzip.





* 1. \*Як буде відбуватись перенаправлення потоків виведення в bash для наступних дій з командами (позначено як cmd) та файлами (позначено як file):

| **Команда** | **Що виконує команда?** |
| --- | --- |
| cmd 1> file | Перенаправляє стандартний вивід команди cmd у файл |
| cmd > file | Аналогічно до попереднього |
| cmd 2> file | Перенаправляє стандартні помилки команди cmd у файл |
| cmd >> file | Додає стандартний вивід команди cmd у кінець файлу |
| cmd &> file | Перенаправляє обидва потоки стандартний вивід та стандартні помилки у файл . Якщо файл існує, він перезапишеться. |
| cmd > file 2>&1 | Спочатку перенаправляє стандартний вивід у файл , а потім перенаправляє стандартні помилки туди ж. Обидва потоки опиняться у файлі |
| cmd >> file 2>&1 | Додає обидва потоки у кінець файлу |
| cmd 2>&1 > /dev/null | Спочатку перенаправляє стандартні помилки у стандартний вивід, але потім стандартний вивід перенаправляється в /dev/null. У результаті стандартний вивід зникає, а стандартні помилки залишаються в консолі. |
| cmd 2> /dev/null | Перенаправляє стандартні помилки у /dev/null, "приховуючи" їх. |
| cmd1 | cmd2 | Використовує конвеєр (pipe), який перенаправляє стандартний вивід першої команди cmd1 на стандартний вхід другої команди cmd2. |
| cmd1 2>&1 | cmd2 | Об'єднує стандартний вивід і стандартні помилки cmd1, а потім передає їх на стандартний вхід cmd2. |

* 1. \*\*Розгляньте наведені нижче приклади та поясніть, що виконують дані команди та який тип перенаправлення потоків вони використовують:

| **Команда**  **(контейнер команд)** | **Що виконує команда?** | **Який потік перенаправлення?** |
| --- | --- | --- |
| $echo "It is a new story." > story | Команда echo виводить рядок "It is a new story." у стандартний вивід (stdout), який перенаправляється у файл story. | стандартний вивід (stdout) перенаправляється у файл story, файл перезаписується. |
| $ date > date.txt | Команда date виводить поточну дату та час у стандартний вивід (stdout), який перенаправляється у файл date.txt. | стандартний вивід (stdout) перенаправляється у файл date.txt, файл перезаписується. |
| $ cat file1 file2 file3 > bigfile | Команда cat об’єднує вміст файлів file1, file2 і file3 і виводить його у стандартний вивід (stdout), який перенаправляється у файл bigfile. | стандартний вивід (stdout) перенаправляється у файл bigfile, файл перезаписується. |
| $ls -l >> directory | Команда ls -l виводить список файлів у поточному каталозі у стандартний вивід (stdout), який додається у кінець файлу directory. | стандартний вивід (stdout) додається до файлу directory. |
| $ sort < file1\_unsorted > file2\_sorted | Команда sort читає вхідні дані з файлу file1\_unsorted через стандартний вхід (stdin), сортує його та перенаправляє результат у файл file2\_sorted. | стандартний вхід (stdin) з файлу file1\_unsorted, стандартний вивід (stdout) у файл file2\_sorted, файл перезаписується. |
| $ find -name '\*.txt' > file.txt 2> /dev/null | Команда find -name '\*.txt' шукає всі файли з розширенням .txt і перенаправляє стандартний вивід (stdout) у файл file.txt. Стандартні помилки (stderr) перенаправляються у /dev/null, щоб приховати їх. | стандартний вивід (stdout) у файл file.txt, стандартні помилки (stderr) у /dev/null. |
| $ cat file1\_unsorted | sort > file2\_sorted | Команда cat file1\_unsorted виводить вміст файлу file1\_unsorted, який передається через конвеєр | у команду sort. Результат сортування перенаправляється у файл file2\_sorted. | стандартний вивід (stdout) перенаправляється в команду sort, стандартний вивід (stdout) sort перенаправляється у файл file2\_sorted, файл перезаписується. |
| $ cat myfile | grep student | wc -l | Команда cat myfile виводить вміст файлу myfile, який передається у команду grep student. grep фільтрує рядки, що містять слово "student", і передає їх у команду wc -l, яка рахує кількість рядків. | стандартний вивід (stdout) перенаправляється через конвеєр у grep, а потім у wc -l. |

**Контрольні запитання:**

1. Надайте порівняльну характеристику процесам стискання та архівування.
2. Які програми, окрім наведених в роботі, можуть використовуватись для стискання та архівування файлів та каталогів в ОС Linux? Наведіть приклади та їх короткий опис.
3. \*Порівняйте алгоритми стискання, що використовуються в командах (програмах), використовуваних в Linux. Які з алгоритмів можна вважати найшвидшим та найефективнішим?
4. \*Опишіть програмні засоби для стискання та архівування, що можуть бути використані у вашому мобільному телефоні.
5. \*Опишіть та порівняйте програмні засоби для стискання та (де)архівування даних у ОС сімейства Windows.
6. \*\*Поясніть яким чином стиснення та архівування даних може бути використано для резервування даних. В яких ще задачах системного адміністрування воно може бути використано.
7. \*\*Яке призначення директорії файлу /dev/null?

**Оформлення звіту:**

1. Титульний аркуш
2. Тема та мета роботи
3. Завдання попередньої підготовки
4. Основні позиції ходу роботи
5. Відповіді на контрольні запитання
6. Висновки за результатами роботи **(обов’язково!!!)**

**Система оцінювання лабораторної роботи:**

Виконано завдання базового рівня складності - **3 бали**

Виконано завдання базового та середнього рівня складності - **4 бали**

Виконано завдання всіх рівнів складності (в тому числі й підвищеного) - **5 балів**

Завдання середнього рівня складності позначені в завданнях (\*)

Завдання підвищеного рівня складності позначені в завданнях (\*\*)

**Примітка**: за виконання робіт в командах та оформлення звітів з використанням системи контролю версій (git) та англійської мови може бути нараховано **додатковий 1 бал**.