Трегуб Ольги, Ксм-03А

**Лабораторна робота №6 Дисципліна: Операційні системи**

**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 (Windows 7).

3. Віртуальна машина – Virtual Box (Oracle).

4. Операційна система GNU/Linux – CentOS.

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

Tasks for preliminary preparation

**4.** On the basis of the considered material, answer the following questions:

**4.1.** What is the purpose of tar, xz, zip, bzip, gzip commands? Make a brief description of each team and highlight their main parameters. How to install them?

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.

**4.2.** Give three examples of implementing data archiving and compression using different commands.

The mentioned packing programs differ significantly in terms of compression rates and speed. When it comes to the **syntax and use** of these tools, though, the similarities are noticeable. All programs can be used without a specific graphic interface or archive manager, via the command line. Beginners can quickly become accustomed to the different parameters and commands.

The universal **syntax of bzip2** has the following form:

bzip2 Optional file(s)

For the standard compression process it’s not necessary to specify options. This is only required if you want to **change compression settings**, access the overview menu, or unpack a *.bz2* file. For example, to pack the text document *test.txt*, you just need to complete the command

bzip2 test.txt

to delete the original file and replace it with the compressed file test.txt.bz2. By placing the documents together, you can also package multiple files with a single command:

bzip2 text.txt test2.txt test3.txt

If you want to **decompress a packed document**, it’s necessary – as mentioned earlier – to set the corresponding option parameters (*-d*):

bzip2 –d test.txt

The combination of tar and a compression tool isn’t required, so you can also **combine files in an archive** that you haven’t previously packaged or don’t want to compress. For example, if you want to bundle the uncompressed test documents *test.txt* and *test2.txt* in the same archive named *archive.tar*, the following command will suffice:

tar –cf archive.tar test.txt test2.txt

To unzip this archive on Linux, replace the *–c* (create new archive) parameter with *–x* (extract files from archive). If not only a certain archive component is to be unzipped, then the file(s) can be omitted:

tar –xf archiv.tar

Alternatively, if you aim to pack a compressed archive – for example, on the basis of the gzip compression, including the extended formatting *.tar.gz* – then tar also offers corresponding options. Since the program has implemented options for compression and decompression with the bzip2, xz, compress, and gzip pack programs, this is also possible with a single command:

tar –czf archive.tar.gz test.txt test2.txt

The command to unpack *.tar.gz* differs from the equivalent for uncompressed directories only through the specification of the pack program parameter:

tar –xzf archive.tar.gz

**4.3.** What is the purpose of the commands cat, less, more, head and tail? Make a brief description of each command and highlight their main parameters. How to install them?

The more, less, and cat commands are used to display the contents of large files. The more command is a very simple command with a small set of options, for example, compared to the similar less command, but it is more than enough for your main task. The text is divided into pages, because the entire text may not fit in the terminal window, when using the more commands, you need to press the "Enter" key to go to the next page, where the percentage of the download of the file content will be shown at the end of the page. Both the less and cat commands have many filtering and sorting options for more convenient output and search of some data. The cat utility has the ability to create and combine files in addition to the task of displaying content on the screen.

The head and tail utilities help perform searches and facilitate the inference of the necessary file contents, and are used in combination with other utilities for a kind of filtering. But tail will output lines from the end of the document, and head at the beginning. The -n option is used to specify the number of lines to display.

**4.4.** Explain how the command shell works with channels, streams, and filters.

A command shell in UNIX is a command line interface in Unix-like operating systems, that is, it executes commands that are given by the user or that are read from files. Such files with shell commands are called shell scripts (scripts, programs). These scripts are not compiled, but interpreted by the shell. This means that the shell reads the script from beginning to end, line by line, looking for the commands specified there and executing them; in contrast to this approach, the compiler converts the entire program into machine-executable form—the code file can then be used in a shell script. A characteristic feature of the shell language is that many operations that are built-in in traditional programming languages are performed by calling external programs.

**4.5.** What is the purpose of the grep command?

grep is a command-line interface utility that finds strings in the input that match a given regular expression and prints them. The name of the utility is a sequence of regular expression search commands in the ed editor — g/re/p. This sequence of commands can be described by the English phrase "search **g**lobally for lines matching the **r**egular **e**xpression, and **p**rint them" — "search everywhere for lines matching the regular expression, and print them".

Modifications of grep:

~egrep (with advanced regular expression processing),

~fgrep (which interprets $\*[]^|()\ characters literally),

~rgrep (with recursive search enabled).