Note: *In this document we will use some commands/operators that is explained in our [first tutorial](). Anything new will be explained.*

Example 1: *Creating a random password generator*

In this example we will create a random password generator by using the some commands we know and some new ones. We will use them in a *pipe chain* fashion, where output of one command is fed to the next command as input.

First things first, we need something *random* for a random password generator right? For that, we have a special "file" in our *Unix* based machines, which is `/dev/urandom` . Whenever you try to read from this file it will give you random bits, forever. Now, if you try to print this file with `cat` command your terminal will try to print these random bits by interpreting them with Unicode/UTF-8 encoding (depending on terminal or OS this may change), and you'll see that most of them are gibberish but there are some characters that we can directly use. We're going to exploit this fact to create our random generator.

We will use `tr` command to get rid of the unwanted characters and only get alphanumeric ones:

```
# This will print random gibberish to your terminal screen
$ cat /dev/urandom

# This will print only alphanumeric characters
$ cat /dev/urandom | tr -dc 'A-Za-z0-9'
```
[1]

Note: *Press `Control+C` or `Command+C` to cancel execution*

Explanation:

-d: Delete. `tr` will delete any character that will satisfy the condition

-c: Complement. `tr` will take the *inverse* of given condition

'A-Za-z0-9': Every character from A to Z, a to z and all numbers

When they're used together `tr` will delete any character from its input that is not in the given set and print the result.

Now, you should see an endless stream of characters again, but this time they're only letters and numbers. So how do we get, say, 16 characters from this? Note that this infinite stream of characters are in fact one very long line. If we could somehow turn it into infinite number of lines of 16 characters, we could just take the first of those lines with `head` command and finish the job. Fortunately, we have `fold` command that does exactly that:

```
# This will print only alphanumeric characters, but only
# 16 of them in a line (-w, specifies width of each line)

$ cat /dev/urandom | tr -dc 'A-Za-z0-9' | fold -w 16
```

This will give you infinite number of lines with 16 random characters on each. Now all we need to do is to get the first line and we're done.

```
# This will print 16 character random string consisting of alphanumeric characters
$ cat /dev/urandom | tr -dc 'A-Za-z0-9' | fold -w 16 | head -n 1
```

And we're done. Everytime you run this command, you will get a different random alphanumeric string/password.

[1] Mac users add `LC_CTYPE=c` in front of `tr`

Example 2: *Monitoring progress of a command.*

In this example we will simply archive and compress a large file using `tar` command. But first we need a *large file.* For this, `dd` command comes to our help. `dd` is *"convert and copy tool"* for files. Similar to `cat`, but `dd` is more intelligent, for example we can specify how many bytes to read where to start reading etc. So let's create our file:

```
# Create a file of ~5Gb containing only null bits (i.e. nothing)
$ dd if=/dev/zero of=bigfile.zero bs=100M count=50
```
[2]

Explanation:

if: Input file (i.e. which file to read from)
/dev/zero: This is a special "file" in your computer (like `/dev/urandom` from previous example) when you try to read from it, it will give you "0" bits.
of: output file name
bs: Block size. How much data is transferred from "if" to "of" in a single iteration
count: How many iterations is done. (In our example 50 iterations each of which is 100Mb so the result is $50 \times 100 = 500\ Mb \approx 5\ Gb$)

Now we have our big file called `bigfile.zero` You can confirm it with `ls` command. We would normally compress and archive it with tar command like this:

```
# Create an archive called compressed.tar.gz with bigfile.zero in it
$ tar -czf compressed.tar.gz bifile.zero
```

Explanation:

-c: Tells tar to create an archive
-z: Compresses the archive with gzip
-f comp… : Name of the newly created archive

The problem is tar won't give much information, so when we're dealing with big files we users usually get anxious. Is it stuck? Will it end in a year? In a minute? How fast it is? To overcome these questions we will first install a monitoring tool called `pv`, using your system's package manager.

For Ubuntu: `$ sudo apt install pv`
For Mac: `$ brew install pv` [3]

Most common use of `pv` is to give it an input from *stdin* and take the same result from *stdout*, and `pv` will give you some monitoring information about it. So to use it properly we have to create a pipe chain, where first commands output goes as input to the second command, second one's output goes to third as input etc. Let's first divide our compression command into two commands:

```
$ tar -cf - bifile.zero | gzip -9 > compressed.tar.gz
```

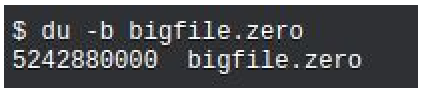
Explanation:

-f - : Instead of a filename we give a dash. This means write the output to *stdout*, so that we can pipe it.
gzip: Instead of *-z* option we have used gzip command directly to compress -9 is the highest compression level and we've redirected the output to a file with ">" operator

This does the exact same thing as the command above.

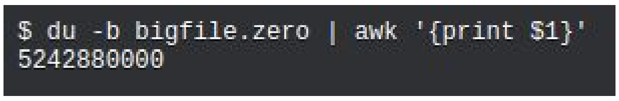
[2] Mac users use small "m": `bs=100m`
[3] Before using brew visit brew's website then execute the command written there in your terminal

Now that we divided our `tar` command into a pipe chain, we can put `pv` in the middle so that we can obtain the monitoring we need, but before we pipe them all together, `pv` also needs total size of the date flowing to give more information. We need to get the size of the file (bigfile.zero in this case) we're processing in bytes and pass it to the `pv` command. Let's get the size of the file in bytes first:



```
$ du -b bigfile.zero
5242880000  bigfile.zero
```
[4]

So far so good, now we need only the number as output and not the file name. To get rid of it, we need to use `awk`, a text manipulation command to get only the first column. Let's pipe this command into `awk` and extract the first column:

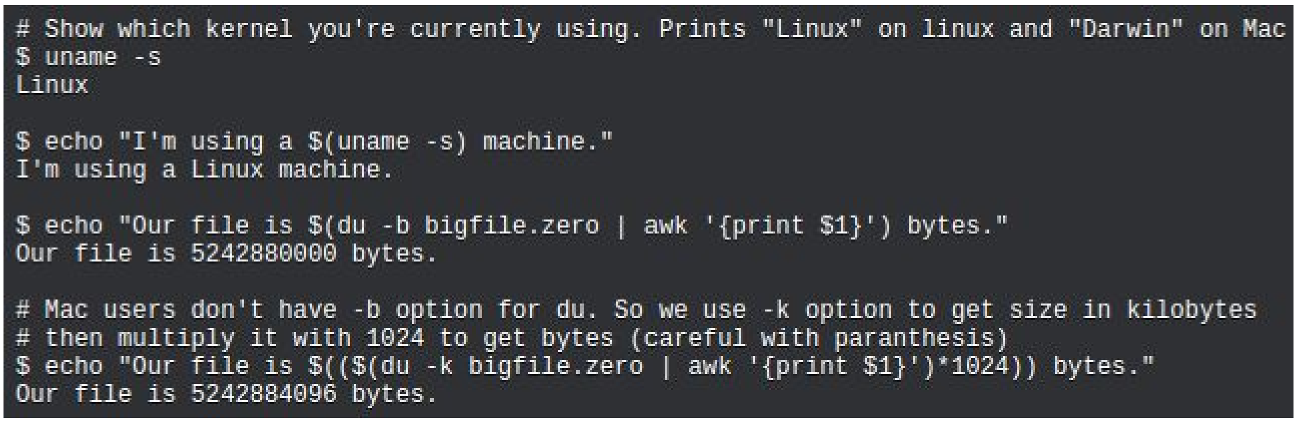


```
$ du -b bigfile.zero | awk '{print $1}'
5242880000
```

Explanation:

awk takes the output ("`5242880000    bigfile.zero`") as input, and we make it to *"print the first column (`$1`)"*

Okay, but we need to pass this result to `pv` command, how do we do that? We put this command into a *subshell*, which will execute the command and return its result. A subshell can be called with either `$(command here)` or `` `command here` `` (those are backticks). Let's see an example:

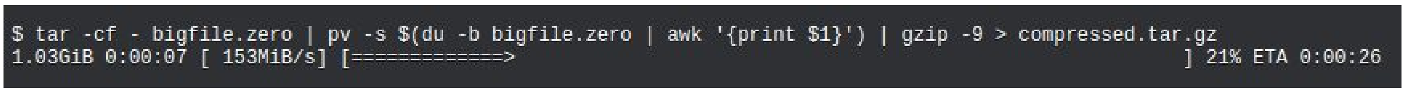


```
# Show which kernel you're currently using. Prints "Linux" on linux and "Darwin" on Mac
$ uname -s
Linux

$ echo "I'm using a $(uname -s) machine."
I'm using a Linux machine.

$ echo "Our file is $(du -b bigfile.zero | awk '{print $1}') bytes."
Our file is 5242880000 bytes.

# Mac users don't have -b option for du. So we use -k option to get size in kilobytes
# then multiply it with 1024 to get bytes (careful with paranthesis)
$ echo "Our file is $(($(du -k bigfile.zero | awk '{print $1}')*1024)) bytes."
Our file is 5242884096 bytes.
```
[5]

Alright great! Now we can take the size of a file in bytes programmatically and put it in some other commands. Now what were we doing again? Right, archive monitoring thing. Let's combine everything together and reach our goal.

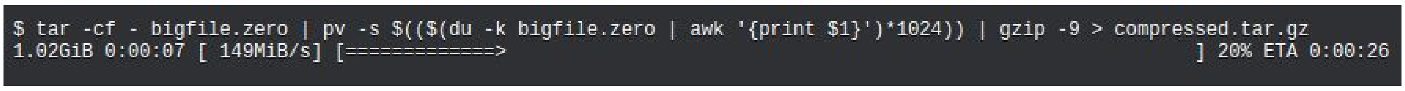
In previous step, we have converted archiving into a pipe chain of 2 commands: `tar` and `gzip`. Now lets insert `pv` in the middle of it as well and get it working.

For Linux:



```
$ tar -cf - bigfile.zero | pv -s $(du -b bigfile.zero | awk '{print $1}') | gzip -9 > compressed.tar.gz
1.03GiB 0:00:07 [ 153MiB/s] [=============>                                                  ] 21% ETA 0:00:26
```

For Mac:



```
$ tar -cf - bigfile.zero | pv -s $(($(du -k bigfile.zero | awk '{print $1}')*1024)) | gzip -9 > compressed.tar.gz
1.02GiB 0:00:07 [ 149MiB/s] [=============>                                                   ] 20% ETA 0:00:26
```

Now we can easily see useful progress information as we archive & compress our files!

[4] Mac users don't have a "-b" option for bytes, I know, keep reading.
[5] Mac users, here's your answer about command not working