LINUX
Practical Guide

Expand your Linux knowledge, skills and technical expertise.

+ Visualization of the regex engine, explanatory
Illustrations, examples and a complete backup script

Aiad Faris

Linux Practical Guide

Leipzig, Germany

Email: printf "bGludXgtZ3VpZGVAcHJvdG9uLm1lCg==" | base64 -d

Table of Contents

Preface

Why a new Linux book?

It’s about more returns from learning, about efficiency in the code, about saving time when searching.

With a variety of customized commands to solve problems or complete everyday tasks. In connection with all of the above, affiliation is also discussed. New products considered by 2023.

Even if the focus is on a partial aspect, a command or a combination on of commands, this is taken into account and referred to other commands or resources.

I hope you don’t mind finding something you already know explained. On the one hand, it cannot be avoided, but on the other hand, it can help to deepen what you have already learned.

Unique Selling Points

- Visualized, pauseable animations that mimics the regex engine.
- A Complete backup script

About the Book

With this book steps are taken towards mastering Linux! It will provide you with valuable knowledge and skills. Given the growing popularity of Linux and its numerous applications across various industries, this book equips you with the tools to confidently navigate the Linux ecosystem. Embrace the journey of learning and soon you will unlock new opportunities and expand your technical expertise.

The topics in this book are arranged alphabetically to make them easy to find.

This book took into account that the level of knowledge varies and the readership is different learning types. The compromises that had to be made are determined by consideration and perspective.

The goal of this book is to deepen understanding through visualization and illustrations, explanatory examples rather than memorizing the examples. It should be an impetus to use Linux intelligently and close the likely skills gap.

Occasionally, details are discussed in great detail in order to trigger a case-specific rethink.

For example, we learn different methods for renaming multiple files at the same time under Linux. We will See a variety of command line examples that cover many different scenarios or are slightly similar can be adapted to certain situations.

If certain syntax seems incomprehensible, analyze the code and question all the details. If you still get stuck, leave it for a while and then return. You'll learn better by figuring it out yourself rather than reading about it.

Beginners can try out the examples with success stories and advanced readers may use the CLI and regex, as I tend to do when needed, to find information quickly enough that they can use it without being distracted from the work at hand.

we're assuming you have the tools used in the book installed. A graphical package manager can be used to install unknown packages. It's useful to easily see what the new package is for, how big it is, and what dependencies it has.

Where the context is distribution specific, then it applies to Debian and Debian based distributions

This book uses GNU Bash version 5.0.3 and GNU SED version 4.7.

This book is v0.2.

For a possible update of the book or to register errata, take a look at the repository of the book on https://github.com/aiadFaris/linux-practical

Notes

The admonition types and syntax highlighting for the source block are enabled and tested several times, still the EPUB reader may not display the admonitions. It may displayed also the colored syntax in monochrome. We depend on EPUB readers more than anyone of us would like.

It may be helpful to adjust the color scheme and prevent the EPUB reader from overwriting the book color in settings/preferences. But even if we succeed, that's no guarantee that it will be the same next time.

The visualized animation makes the book partly interactive. The animation is intended to make it easier to understand how the regex engine works and ultimately consciously select the most efficient regular expressions. This animation is only possible in the EPUB or HTML version.

The animation in the SVG version can be paused.

Dedication

To friends,

To each and every one of you, thank you for being a part of my journey.

With heartfelt gratitude,

Aiad Faris, 2023

About the Author

> Aiad Faris is a freelance software developer, autodidact and trainer. His experience primarily focused on Linux.

Pages

17

45

are skipped

GNU Awk

By combining and varying the commands used here, I hope to help make learning easier and consolidate the knowledge gained by users.

AWK and PCRE (Perl Compatible Regular Expressions)

First of all, awk does not support look-ahead or look-behind, since it uses POSIX Extended Regular Expression (ERE).

An awk program on the Command Line consists of `pattern { action }`.

Otherwise we put awk program in a file and run it with the schematic syntax `awk -f program-file input-file1 input-file2`.

Built-in Variables That Control awk

For more details read the manual `man awk` and the manual https://www.gnu.org/software/gawk/manual/gawk.html

RT; After the end of the record has been determined, gawk sets the variable RT to the text in the input that matched RS.

FIELDWIDTHS; A space-separated list of columns that tells gawk how to split input with fixed columnar boundaries. Example

FPAT; A regular expression (as a string) that tells gawk to create the fields based on text that matches the regular expression.

Here we really want to define the fields by what they are, and not by what they are not. Example

FS; The input "field separator". The value is a single-character string or a multicharacter regular expression that matches the separations between fields in an input record. If the value is the null string (""), then each character in the record becomes a separate field. (This behavior is a gawk extension).

You can set the value of FS on the command line using the -F option:

```
awk -F, 'program' input-files
```

Otherwise gawk is using FIELDWIDTHS or FPAT for field splitting. assigning a value to FS e.g. 'FS = FS' causes gawk to return to the normal, FS-based field splitting.

OFMT; A string that controls conversion of numbers to strings (see Conversion of Strings and Numbers) for printing with the print statement. Example for OFMT

Earlier versions of awk used OFMT to specify the format for converting numbers to strings in general expressions; this is now done by CONVFMT. Example for CONVFMT

OFS; The "output field separator" (see Output Separators). It is output between the fields printed by a print statement. Its default value is " ", a string consisting of a single space.

ORS; The "output record separator". It is output at the end of every print statement. Its default value is "\n", the newline character. (See Output Separators.)

RS; The input "record separator". Its default value is a string containing a single newline character, which means that an input record consists of a single line of text. It can also be the null string, in which case records are separated by runs of blank lines. If it is a regexp, records are separated by matches of the regexp in the input text. (See How Input Is Split into Records.)

The ability for RS to be a regular expression is a gawk extension. In most other awk implementations, or if gawk is in compatibility mode (see Command-Line Options), just the first character of RS's value is used.

Built-in Variables That Convey Information

Listing 28. ARGC, ARGV

```
$ awk 'BEGIN {
         for (i = 0; i < ARGC; i++)
             print ARGV[i]
      }' Employee date
awk
Employee
data
```

Check awk variables with `cat awkvars.out`

Built-in Variables That Convey Information:

FNR; The current record number in the current file. awk increments FNR each time it reads a new record (see How Input Is Split into Records). awk resets FNR to zero each time it starts a new input file.

NF; The number of fields in the current input record. NF is set each time a new record is read, when a new field is created, or when $0 changes (see Examining Fields).

Unlike most of the variables described in this subsection, assigning a value to NF has the potential to affect awk's internal workings. In particular, assignments to NF can be used to create fields in or remove fields from the current record. See Changing the Contents of a Field.

Listing 29. Example for FIELDWIDTHS:

```
$ echo "bbcccddddeeeee"|awk -v FIELDWIDTHS="2 3 4 5" '{for(i=1;i<=NF;  i++)print $i}'
 bb
 ccc
 dddd
 eeeee
# It is helpful, when it is difficult to find a FS of the record, but the record has
fixed length fields.
```

Example for FPAT:

Quoting the example in the manual, section "Defining Fields. Content"

As example is file with comma-separated values, where is each fields are separated by commas. If one of the fields contains an embedded comma, is handled by enclosing the content in double quotes.So, we might have data like this:

```
Tom, Harry,"12 John Doe Street, NE",Town,State,12345-6789,USA
```

The FPAT variable offers a solution by describing each field with regular expression (Don't rely on commas).

The part `([^,]+)` match content between commas and the other part match content between double quotes.

```
$ echo "Tom, Harry,\"12 John Doe Street, NE\",Town,State,12345-6789,USA" | awk '
BEGIN {
    FPAT = "([^,]+)|(\"[^\"]+\")"
}

{
    print "NF = ", NF
    for (i = 1; i <= NF; i++) {
        printf("$%d = >%s<\n", i, $i)
    }
}'
NF =  7
$1 = >Tom<
$2 = > Harry<
$3 = >"12 John Doe Street, NE"<
$4 = >Town<
$5 = >State<
$6 = >12345-6789<
$7 = >USA<
```

Example for OFMT:

OFMT is used numbers are printed, e.g.:

```
$ <<< 0.6665643 awk '{ print 0+$0 }' OFMT='%.2g'
  0.67

# With this is $0 converted to number, confirming:
$ echo "scale=4;$o+9.11" | bc
  9.78
```

CONVFMT is used when numbers are explicitly converted to strings, e.g.:

```
# To convert $0 into a sting we concatenate it with the empty string.
$ <<< 0.6665643 awk '{ print "" 0+$0 }' CONVFMT='%.2g'
  0.67
```

Delete Duplicate Lines With AWK

```
## Delete duplicate lines in a file without sorting. Independent of location of
duplicated line
$ awk '!arr[$0]++' dups.txt

# To persist the result in the same input file
$ awk '!arr[$0]++' dups.txt | sponge dups.txt
```

Explain by examples

Listing 30. Run only awk file

```
#awk-file
$ cat scriptawk
awk 'BEGIN {
   week["Monday"] = "sunny";
   week["Friday"] = "cloudy"
   print week["Monday"] "\n" week["Friday"]
}'

$ ls -l scriptawk
  -rwxr-xr-x 1 user user 119 Feb 26 13:35 scriptawk

# Running awk file without input file
$ ./scriptawk
  sunny
  cloudy

# incorrect
$ scriptawk
  bash: scriptawk: command not found

# Not possible. It expects an argument. Must be aborted.
$ awk scriptawk
^C
```

The keyboard combination to cancel is “^C” and means Ctrl+C

Pages

50

68

are skipped

BASH

Read:

- the manual page $ `man bash`S
- "Global Environment Variables" A free book: https://riptutorial.com/ebook/bash

Run a script explicitly in bash

```
$ bash file.sh
# If bash is already defined as current shell
$ ./file.sh
```

Listing 65. Conditional Execution In Bash, Examples:

```
# To provide feedback when the command used does not have a verbose option:
$ touch some-file && echo "some-file is created."
# That was like using the following commands
$ touch some-file; if (( $? == 0 )); then echo "some-file is created."; fi
# Or
$ if touch some-file; then echo "some-file is created."; fi

# Execute only when we get to a specific folder
$ cd tests/ && \rm *txt

# Create a folder, change the current directory to it, create a file and write into it
$ mkdir smFldr && { cd $_ ; fortune > some-file ; }
~/smFldr$ ls
  some-file

~/smFldr$ cd .. && rmdir smFldr
  rmdir: failed to remove 'smFldr': Directory not empty

$ rmdir smFldr > /dev/null 2>&1 || { echo "smFldr is not empty, we&#8217;ll use \"rm
-rf\"" ; \rm -rf ./smFldr && echo "Now it's deleted" ; }
smFldr is not empty, we&#8217;ll use "rm -rf"
Now it&#8217;s deleted
```

shopt & autocd: turn on the autocd option on bash

Bash Parameter Expansion:

The output of "echo $-" shows your Builtin Set Flags. man bash then look for SHELL BUILTIN COMMANDS and then look for the set subsection. You will find the meanings of all those flags:

```
h: Remember the location of commands as they are looked up for execution. This is
enabled by default.
i: interactive
m: Monitor mode.  Job control is enabled
B: The shell performs brace expansion (see Brace Expansion above). This is on by
default
H: Enable !  style history substitution.  This option is on by default when the shell
is interactive.
```

"$-" is current option flags set by the shell itself, on invocation, or using the set builtin command:

Listing 66. app.sh

```
$ echo $- ①
  bhimBHs
$ set -a && echo $-  ②
  abhimBHs
```

① Library import

② URL mapping

Is the current shell interactive?

```
$ [[ $- == *i* ]] && echo "This shell is interactive." || echo "This shell is not
interactive"
# OR
case "$-" in
 *i*)   echo This shell is interactive ;;
 *) echo This shell is not interactive ;;
esac
```

Use Case in .bashrc: navigating the bash_history:

```
if [[ $- == *i* ]]
 then
  ## arrow up
  bind '"\e[A": history-search-backward'
  ## arrow down
  bind '"\e[B": history-search-forward'
fi

# #As alternative ~/.inputrc maybe used with similar content.
```

Bash: Shell Environment Variables:

Environment and shell variables are always present in your shell sessions and can be very useful. They are an interesting way for a parent process to set the configuration details for its child process and provide a way to set options outside of files.

The list consists of "SHELL,TERM,USER, BASHOPTS, BASH_VERSION" and others.

- To see the full list of environment variables, issue the command `env` or `printenv`. You can see the list of shell variables by running the “set” command.

```
# Ex: to display our current shell:
$ printenv SHELL
  /bin/bash

# we create a shell variable as name and value:
$ MY_VAR='Wonderful World!'

#This variable is available in our current session, but is not passed to child
processes. Because if it were, it wouldn't be available for set

$ set | grep MY_VAR
  MY_VAR='Wonderful World!'
# We can verify that it is not an environment variable
$ printenv | grep MY_VAR
# but a shell variable:
$ echo $MY_VAR
  Wonderful World!

# Now we have a shell variable. It should not be passed to child processes. We can
start a new bash shell from our current one to demonstrate:
$ bash
$ echo $MY_VAR
# isn't available
# We go back to our original shell by typing exit:
$ exit
  exit

# Now let us turn our shell variable into an environment variable. We can do this by
exporting the variable:
$ export MY_VAR
# We can verify this by checking our environment list again:
$ printenv | grep MY_VAR
  MY_VAR=Wonderful World!

# As a further use case, there are examples that can be placed in .bashrc
odt ()
{ arg1=$1;
  lowriter -o $arg1 & disown }
```

Pages

72

88

are skipped

Bitwise Operators

Figure 1. Representation of 8 bits values:

```
Ex.:
13 = 00001101
```

```
.Explanatory Breakdown:
 13 = 00001       1       0       1
          1*(8) + 1*(4) + 0*(2) + 1*(1)  = 13
```

Operators		Description
~	NOT	Inverts all bits.
&	AND	Returns 1 only if both bits are equal to 1
|	OR	Returns 1 if at least one of the two bits is equal to 1
^	XOR	Returns 1 if only the two bits are different to 1.(Returns 0 if both are 0 or both are 1)
~|	NOR	Inverts all bits of the OR operation result.
~&	NAND	Inverts all bits of the AND operation result
~^ or ^~	XNOR	Inverts all bits of the XOR operation result
<<	shift left	Takes a binary number and literally shifts all the bits to the left, and adds a 0 in the least significant bit positions.
>>	shift right	Takes a binary number and shifts all the bits to the right by a specified amount, and adds a 0 in the most significant bit positions.
If an operator occurs in combination with e.g. assignment operator, then it is noted as "&=" or "~=" and is called "Bitwise AND Assignment" and "Bitwise NOT Assignment" accordingly. For instance "a ^= b" means "a = a^b". Example: 0111 ^ 0100 ⇒ 0011. But Note "!=" stands for "inequality"		

Bit Operators - Explanatory Breakdown

Table 1. AND OR XOR

	AND		OR		XOR	
Operands	**0**	**1**	**0**	**1**	**0**	**1**
0	0	0	0	1	0	1
1	0	1	1	1	1	0

AND		
A	**B**	**A & B**
0	0	0
0	1	0
1	0	0
1	1	1

OR		
A	**B**	**A | B**
0	0	0
0	1	1
1	0	1
1	1	1

XOR		
A	**B**	**A ^ B**
0	0	0
0	1	1
1	0	1
1	1	0

To express XOR, the circled plus symbol can be used instead of “^”.

NOR		
A	**B**	**A ~ B**
0	0	1
0	1	0
1	0	0
1	1	0

NAND		
A	**B**	**A ~& B**
0	0	1
0	1	1
1	0	1
1	1	0

XNOR		
A	**B**	**A ~^ B**
0	0	1
0	1	0
1	0	0
1	1	1

NOT	
A	**~A**
0	1
1	0

When we add 1 to 9 in base 2 (1 is the largest single number we have) we get a number with a new "1" to its left: 10. When adding two numbers, this results in carrying sums larger than 9 into the next column and we get a number with a new "1" to its left: 10.

Table 2. Addition in base 10

	10^2	**10^1**	**10^0**
	1	8	6
+			
			4
=	1	9	0

Addition in base 2 (Decimal: 186 + 2)

When adding two numbers, this results in carrying sums larger than 1 into the next column and we get a number with a new "1" to its left: 10.

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
	1	0	1	1	1	0	1	0	(186)
+									
	0	0	0	0	0	0	1	0	(2)
=	1	0	1	1	1	**1**	0	0	**188**

Table 3. Subtraction in base 2 (Decimal: 186 - 2)

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
	1	0	1	1	1	0	1	0	(186)
-									
	0	0	0	0	0	0	1	0	(2)
=	1	0	1	1	1	**0**	0	0	**184**

Subtraction with Two's Complement

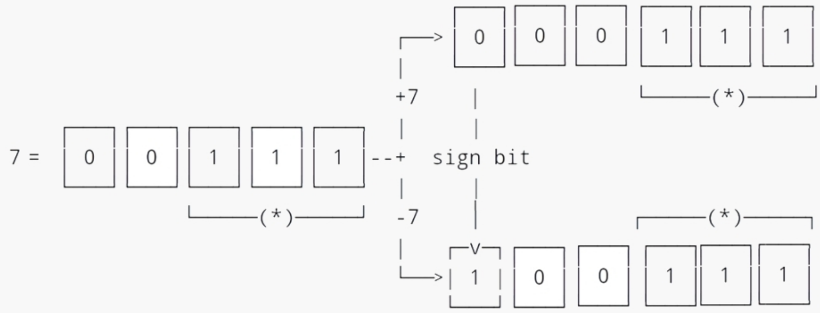
1. In take our number above (2) 00000010
2. We negate it 11111101
3. And add 1 11111110

Apply now

```
  10111010     (186)
  11111110   + (-2)
------------------------
  10111000
```

The first bit on the left specifies the sign, which is 0 for positive numbers and 1 for negative numbers If the number is negative, its absolute value to its opposite, with all bits inverted and to which we added 1.

Representation of Negative Binary Numbers



(*) magnitudes are unchanged

Sign Bit

Example:

```
         ⎛ 0111    (4 Bits)   │   ⎛ 1111    (4 Bits)
    +7   ⎜ 00111   (5 Bits)   │ -7⎜ 10111   (5 Bits)
         ⎝ 000111  (6 Bits)   │   ⎝ 100111  (6 Bits)
```

Work with flexible bit length:

In order to be able to flexibly decide in which bit length we work, we create a function for it. The first argument is the bit length and the second is the input number we are working with.

```
$ btr () { printf 'obase=2; 2^%d+%d\n' "$1" "$2" | bc | sed -E "s/.*(.{$1})$/\1/"; }
$ btr 4 7                        |   $ btr 4 -7
  0111                           |     1001
$ btr 5 7                        |   $ btr 5 -7
  00111                          |     11001
$ btr 6 7                        |   $ btr 6 -7
  000111                         |     111001
```

Explanation

```
btr () { printf 'obase=2; 2^%d+%d\n' "$1" "$2" | bc | sed -E "s/.*(.{$1})$/\1/"; }
```

```
$ btr 8 0
  00000000
```

```
$ btr 8 32
  00100000
```

Listing 78. Explanatory Breakdown

```
     2^8
     ^^^
     vvv
     2^d
```

```
   2^%d+%d\n'
      ^  ^
      |  |
      v  v
$ btr 8  0
  00000000
```

```
$ btr     8
      vvvv^vvvv
      0000 0000
```

```
$ btr 8 32
        ^^
        ||
        vvvvvv
  00    100000
```

Real World Example
Toggle case of a string using Bitwise Operators

Listing 79. With XOR Operator

```
a  01100001       97
       ^
   00100000       32
```

```
   ________________
A  01000001       65

$ echo $((2#01100001^2#00100000))
  65
$ a=01100001; x=00100000; echo $((2#$a^2#$x))
  65
```

Listing 80. Subtraction

```
$ echo $((2#01100001-2#00100000))
  65
```

Pages

97

117

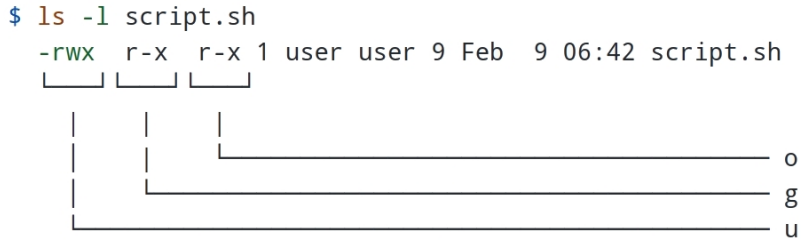
are skipped

respectively. A **who** consisting of the symbol **a** shall be equivalent to **ugo**.

Permissions in symbolic_mode

```
$ ls -l file.txt
-rw-r--r-- 1 user user 0 Feb  9 06:12 file.txt

$ echo 'echo hey' > script.sh
$ chmod a+x script.sh
```




```
                                                  
o: other, the octal value of rwx is (4+2+1)       
g: group, the octal value of r-x is (4+1)         a =(u,g,o)
u: user (owner), the octal value of r-x is (4+1)  
                                                  
```

For file

```
# User, hostname and path: "user@lnb:/tmp/tmp.tEDNhKUIcH" will be truncated in the
following lines

$ printf '#!/bin/bash\necho greeting' > file.sh
$ ls -l file.sh
  -rw-r--r-- 1 user user 25 Feb 10 17:34 file.sh

$ chmod u+x file.sh
$ ls -l file.sh
  -rwxr--r-- 1 user user 25 Feb 10 17:34 file.sh
$ ./file.sh
  greeting
  -rwxr--r-- 1 user user 25 Feb 10 17:34 file.sh

$ chmod a+x file.sh
$ ls -l file.sh
  -rwxr-xr-x 1 user user 25 Feb 10 17:34 file.sh
$ chmod a+x file.sh
$ ls -l file.sh
  -rwxr-xr-x 1 user user 25 Feb 10 17:34 file.sh

$ chmod 744 file.sh
```

```
$ ls -l file.sh
 -rwxr--r-- 1 user user 25 Feb 10 17:34 file.sh

$ chmod 755 file.sh
$ ls -l file.sh
  -rwxr-xr-x 1 user user 25 Feb 10 17:34 file.sh
```

```
# Breaking it down:
-rwxr-xr-x 1 user user 25 Feb 10 17:34 file.sh
^^^^
||||
|||└──────────────────────────────────────────── x (1)
||└──────────────────────────────────────────── w (2)
|└──────────────────────────────────────────── r (4)
└──────────────────────────────────────────── - (represents file)
```

Figure 2. Breaking it down:

```
x Execute: Can run a script. The octal value of *x* is 1.
w Write:   editing, moving and deleting of files. The octal value of *w* is 2
r Read:    reading a file. The octal value of *r* is 4.
- precedes always the usual file

- between r and x means no permissions have been granted.
```

For directories

```
$ mkdir Dir1
```

```
$ mkdir Dir1
  $ ls -ld Dir1
  drwxr-xr-x 2 user user 4096 Feb  9 06:12 Dir1
  ^^^^
  ||||
  |||└──────────────────────────────────────────── x (1)
  ||└──────────────────────────────────────────── w (2)
  |└──────────────────────────────────────────── r (4)
  └──────────────────────────────────────────── d (represents folder)
```

```
x Execute: Can "cd" into the directory or use it in a pathname. The octal value of *x*
is 1.
w Write: creating or deleting files within the directory. The octal value of *w* is 2.
r Read: displaying the file/sub-directories and their meta data. The octal value of
*r* is 4.
d stands for directory
```

For directories chmod preserves set-user-ID and set-group-ID bits unless you explicitly specify

Pages

120

151

are skipped

Encryption

GnuPG

Encryption and Decryption a file with Ciphers

Listing 125. Consider

```
$ openssl enc -aes-256-cbc -salt -in secret.md -out secret.md.enc
enter aes-256-cbc encryption password:
Verifying - enter aes-256-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
```

The warning above relates to the lack of algorithm pbkdf2 and iterations for the password. Maybe it helps to know which version is installed.

Listing 126. Check the version of the installed openssl

```
$ openssl version
  OpenSSL 1.1.1n  15 Mar 2022
```

Listing 127. Version 1.1.1 and higher allows secure encryption

```
# Make sure your password has very high entropy !
$ openssl enc -aes-256-cbc -md sha512 -pbkdf2 -iter 100000 -salt -in secret.md -out
secret.md.enc
enter aes-256-cbc encryption password:
Verifying - enter aes-256-cbc encryption password:
$
```

Explanation

-aes-256-cbc

AES is a variant of Rijndael, with a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits. CBC: Cipher Block Chaining Mode (CBC Mode) is a mode in which block ciphers can be operated. Before a plaintext block is encrypted, it is first XORed (exclusive OR) with the cipher text block generated in the previous step. A 256-bit secret key means that an attacker has to do 2^255 "operations" to discover the key by brute force guessing.

-md sha512

SHA 512 is a modified version of MD5.

-pbkdf2

Use PBKDF2 (Abbr. for: Password-Based Key Derivation Function 2) algorithm. As CPU speeds increase more iteration can be done. For PBKDF2-HMAC-SHA512 the iterations are less than for

Pages

153

154

are skipped

choice.

- Enter how long the key should be valid. Press Enter to specify the default selection, indicating that the key doesn't expire. Unless you require an expiration date, we recommend accepting this default.
- Verify that your selections are correct.
- Enter your user ID information.
- Type a secure password.

```
$ gpg -c .bashrc
# 2x password
$ ls *gpg
  .bash_aliases.gpg  .bashrc.gpg
$ tar cvf bashrc_bash_aliases.tar .bashrc.gpg .bash_aliases.gpg
```

Encryption of folder

Listing 131. Encrypt a directory. No pass required:

```
$ gpgtar --encrypt -o homebkpGpg -r noppetil@outlook.com homebkp
```

Listing 132. Encryption with openssl

```
$ openssl genrsa -help
  Usage: genrsa [options]
  Valid options are:
   -help               Display this summary
   -3                  Use 3 for the E value
   -F4                 Use F4 (0x10001) for the E value
   -f4                 Use F4 (0x10001) for the E value
   -out outfile        Output the key to specified file
   -rand val           Load the file(s) into the random number generator
   -writerand outfile  Write random data to the specified file
   -passout val        Output file pass phrase source
   -*                  Encrypt the output with any supported cipher
   -engine val         Use engine, possibly a hardware device
   -primes +int        Specify number of primes
```

```
$ openssl genrsa --out my.ssl.key.pem 4096
  Generating RSA private key, 4096 bit long modulus (2 primes)
  ..............[... truncated ...].........++++
  e is 65537 (0x010001)
```

Caesar encryption

Replace each letter with another letter 3 positions later down the alphabet. XYZ in plain text become ABC in cipher.

Listing 133. Encryption:

```
$ echo "CAESAR" | tr '[A-Z]' '[D-ZA-C]'
  FDHVDU
```

Listing 134. Decryption:

```
$ echo "FDHVDU" | tr '[A-Z]' '[X-ZA-W]'
  CAESAR
```

```
$ age-keygen -o key.pub
Public key: age175ufkx4f257jn77sgu38fnfqwt6upzuz244pysu2..x23n
```

Listing 135. Run age command to encrypt a file with public key:

```
$ age -r age175ufkx4f257jn77sgu38fnfqwt6upzuz244pysu2..x23n data.txt > data.txt.age
```

```
$ rm data.txt; age -d -i key.pub data.txt.age > data_decrypted.txt
```

Listing 136. Use 7-Zip with AES-256 encryption

```
$ 7z a -pdI..2 -mhe -t7z sec.txt.age.7z sec.txt.age
```

Listing 137. Decrypt

```
$ 7z x sec.txt.age.7z -pdI..2
```

Encipher with ImageMagick

With ImageMagick only the pixels are encrypted. The encrypted image will still be recognized as an image andwill even be displayed on your website. However, the content appears to be gibberish and has nothing to do with the original content.

Encipher an Image

Use the "-encipher" option to encrypt your image so that it becomes unrecognizable. The option requires a file that contains your passphrase.

Pages

157

192

are skipped

ls Command

LS man page

ls - list directory contents sorted

SYNOPSIS: ls [OPTION]... [FILE]...

```
Mandatory arguments to long options are mandatory for short options too.
```

Option	Description
-a, --all	do not ignore entries starting with "." (hidden files/folders)
-A, --almost-all	do not list implied . and ..
--author	with **-l**, print the author of each file
-b, --escape	print C-style escapes for nongraphic characters

--block-size=SIZE	with **-l**, scale sizes by SIZE when printing them; e.g., '--block-size=M';
	
	`$ ls -l`
	`  total 8`
	`  drwxr-xr-x 2 user user 4096 Mar 18 01:31 dir`
	`  -rw-r--r-- 1 user user   49 Mar 17 23:52 funk1.txt`
	
	`$ ls -l --block-size=K`
	`  total 8K`
	`  drwxr-xr-x 2 user user 4K Mar 18 01:31 dir`
	`  -rw-r--r-- 1 user user 1K Mar 17 23:52 funk1.txt`
	
	`$ ls -l --block-size=M`
	`  total 1M`
	`  drwxr-xr-x 2 user user 1M Mar 18 01:31 dir`
	`  -rw-r--r-- 1 user user 1M Mar 17 23:52 funk1.txt`
	
	see SIZE format below

-B, --ignore -backups	do not list implied entries ending with ~

-c	with **-lt**: sort by, and show, ctime (time of last modification of file status information); with **-l**: show ctime and sort by name; otherwise: sort by ctime, newest first
	
	`$ ls -cl`
	`  total 8`
	`  drwxr-xr-x 2 user user 4096 Mar 18 01:31 dir`
	`  -rw-r--r-- 1 user user  219 Mar 18 01:51 funk1.txt`
	`$ ls -clt`
	`  total 8`
	`  -rw-r--r-- 1 user user  219 Mar 18 01:51 funk1.txt`
	`  drwxr-xr-x 2 user user 4096 Mar 18 01:31 dir`

-C	list entries by columns
--color[=WHEN]	colorize the output; WHEN can be 'always' (default if omitted), 'auto', or 'never'; more info below
-d, --directory	list directories themselves, not their contents
-D, --dired	generate output designed for Emacs' dired mode
-f	do not sort, enable -aU, disable -ls --color

-F, --classify	append indicator (one of */⇒@|) to entries
	
	`$ mkdir dir`
	`$ touch dir/file`
	`$ ln -s dir/file`
	`$ mkfifo fifo`
	`$ ls -F`
	`  dir/  fifo|  file@`
	
	`$ ls -l fi*`
	`prw-r--r-- 1 user user  0 Apr  4 05:47 fifo`

--file-type	likewise, except do not append '*'
--format=WORD	across -x, commas -m, horizontal -x, long -l, single-column -1, verbose -l, vertical -C
--full-time	like -l --time-style=full-iso
-g	like -l, but do not list owner
--group -directories-first	group directories before files; can be augmented with a --sort option, but any use of --sort=none (-U) disables grouping
-G, --no-group	in a long listing, don't print group names
-h, --human -readable	with -l and -s, print sizes like 1K 234M 2G etc.
--si	likewise, but use powers of 1000 not 1024

-H, --dereference -command-line	follow symbolic links listed on the command line
--dereference -command-line -symlink-to-dir	follow each command line symbolic link that points to a directory
--hide=PATTERN	do not list implied entries matching shell PATTERN (overridden by -a or -A)
--hyperlink[=WHEN]	hyperlink file names; WHEN can be 'always' (default if omitted), 'auto', or 'never'
--indicator -style=WORD	append indicator with style WORD to entry names: none (default), slash (-p), file-type (--file-type), classify (-F)
-i, --inode	print the index number of each file
-I, --ignore=PATTERN	do not list implied entries matching shell PATTERN
-k, --kibibytes	default to 1024-byte blocks for disk usage; used only with -s and per directory totals
-l	use a long listing format

-L, --dereference	when showing file information for a symbolic link, show information for the file the link references rather than for the link itself
	
	First find symlinks in a directory tree
	
	`$ ln -s linux/audit`
	
	`$ ls -l audit`
	`lrwxrwxrwx 1 user user 11 Apr 4 05:35 audit -> linux/audit`
	
	`$ ls -lR ./aud* | grep ^l`
	`lrwxrwxrwx 1 user user 11 Apr 4 05:35 ./audit -> linux/audit`
	
	`$ ls -lLd audit`
	`drwxr-xr-x 2 user user 4096 Apr 4 05:34 audit`

-m	fill width with a comma separated list of entries

-n, --numeric-uid -gid	like -l, but list numeric user and group IDs
	
	`$ ls -ld dir*`
	`drwxr-xr-x 2 user user 4096 Apr 4 05:46 dir`
	`drwxr-xr-x 2 user user 4096 Apr 4 05:53 dir2`
	
	`$ ls -nd dir*`
	`drwxr-xr-x 2 1000 1001 4096 Apr 4 05:46 dir`
	`drwxr-xr-x 2 1000 1001 4096 Apr 4 05:53 dir2`

Pages

196

226

are skipped

Q

QR qrencode

```
See also the manual ``man qrencode`` for more details
```

```
$ qrencode -s 6 -l H -o "text.png" "This type of QR holds plain text that text is
shown when they scan the QR code."
$ qrencode -s 6 -l H --foregound="3599FE" --background="FFFFFF" -o "blue.png" "This QR
code will be blue and white."
```

```
Connect to a Wi-Fi Network from a QR Code
T: type;"WEP, WPA, or WPA2."S: The Service Set ID (SSID)", P: The password.
```

```
## Syntax: qrencode -s 6 -l H -o "wifi.png" "WIFI:T:WPA;S:<SSID>;P:<PSWD>;;
$ qrencode -s 6 -l H -o "wifi.png" "WIFI:T:WPA2;S:AI.FA!;P:CwHaTeVeR;;
```

Listing 183. Redirection into qrencode

```
$ qrencode -s 6 -l H -o "contacts.png" < contacts.txt
$ qrencode -o UNSW-bob.png -t png < data.txt
```

Reading from QR code with Python as file o on CLI

```
$ python qrdecode.py
```

```
$ cat qrdecode.py
from qrtools import QR
my_QR = QR(filename = "/home/user/Desktop/UNSW-mara.png")
my_QR.decode()
print (my_QR.data)
```

Pages

228

242

are skipped

To use SSH with CA certificates and two-factor authentication see the related SSH chapter

Explanatory Representation:

Pull (by host-IP and by default port 22):

```
~$ rsync -av -e   ssh    tractatio@192.168.2.40 : it/       it/
        └─┬────┘ └─┬──┘└──────────┬──────────┘ └────┘   └────┘
          |        |     username    host - IP    remote   local
          |        |                              source   destination
          |        |    └──────────────────────────────┘
          |        |           remote URI
          |        └──────────────────────────────────── protocol
          └──────────────────────────────────────────────── options
```

Push (by host-IP and by default port 22):

```
~$ rsync -av -e    ssh      it/   tractatio@192.168.2.40 : it/
        └─┬────┘ └─┬──┘└──┬──┘└──────────┬──────────┘ └────┘
          |        |     local   username    host-IP      remote
          |        |     source                           destination
          |        |              └──────────────────────────────┘
          |        |                    remote URI
          |        └──────────────────────────────────────────── protocol
          └──────────────────────────────────────────────────────── options
```

Push (by hostname in config file and by default port 22):

```
~$ rsync -av -e    ssh      it/            pc            : it/
        └─┬────┘ └─┬──┘└──┬──┘└──────┬──────┘ └────┘
          |        |     local         hostname          remote
          |        |     source                          destination
          |        |              └──────────────────────────┘
          |        |                    remote URI
          |        └──────────────────────────────────────── protocol
          └──────────────────────────────────────────────────── options
```

See how to use .ssh/config

Use Cases With Other Options

We push from source Tab11 to the Notebook "lnb" to update some files. Thereby we use new flags **-P** (or --progress) and **-u** (or --update) to skip files that are newer on the receiver.

```
$ rsync -aPvu -e ssh it/ lnb:./it/
```

Sometimes we need to delete some of our files, e.g. to gain free disk space, to keep the data up-to-date, or for other reasons. To do this, we use the "**--delete**" option:

```
$ rsync -aPvu --delete -e ssh it/ user@192.168.2.45:it/
```

Excluding data from syncing

There are a number of file types that need to be excluded for various reasons. Using a list file with

Pages

244

246

are skipped

The config file "$HOME.ssh/config"

Synchronization via SSH with certificates and 2FA uses Google Authenticator

An essential excerpt from the configuration file config. It has been successfully tested with rsync as shown in the example above:

```
Host *
 StrictHostKeyChecking no
 UserKnownHostsFile=/dev/null
 LogLevel QUIET
 ForwardAgent yes
 ControlMaster auto
 ControlPath ~/.ssh/control-%h-%p-%r
 ControlPersist = no
 IdentityFile ~/.ssh/ssh-user-rsa-key
 #PubKeyAuthentication yes  +
Host lnb
 HostName 192.168.70.197
 User user
 #Port 7002 +
Host pc
...#truncated. Use the same directives with adjusted values for all other host
entries.
```

Data names with meta characters

Our data names should not contain any meta character. If so, we need to remove these characters.

See how to do this with SED

Rsync on Android (Termux)

Syncing by pulling. Destination is Tab11 (Android with Termux) pull from Source PC with Debian 10.

On Android one has to work with limitations such as changing file attributes or following symlinks as it all happens without root privileges.
To avoid error messages, exclude these changes (even implicitly) from your command. Android (Termux) should be used as the destination and the synchronization should only be done via pull. Turning Android into an SSH server (by installing corresponding app) doesn't add any value.

Example :

```
~/storage/downloads $ rsync -aPpv -e ssh pc:./Downloads/tab.ebooks/ .
```

Even if that doesn't apply to what you're trying to do, you can conclude that, in practice, it's worth delving deeper into rsync.

Rsync with Termux

The Termux shortcuts briefly listed:
^j Open new tab
^1 Return to the first tab if exists
^2 Return to the second tab if exists

```
~ $ cdit
.../0/it $ cd ..
.../emulated/0 $ ls it/rsync.it.filter
```

On Android one has to work with limitations such as changing file attributes or following symlinks as it all happens without root privileges.
To avoid error messages, exclude these changes (even implicitly) from your command. Android (Termux) should be used as the target and the synchronization should only be done via pull. Turning Android into an SSH server (by installing an app) doesn't add any value.
To use SSH with CA certificates and two-factor authentication see the related SSH chapter

Example on termux:

```
.../emulated/0 $ rsync -aPpvu --delete --exclude-from=it/rsync.it.filter -e ssh
lnb:it/  it/
```

Make sure you get the correct IP (adjacent to "(\d"{1,3}.){3}(\d"{1,3})"%wlp4s0) of the remote host

Pages

249

339

are skipped

Visual Demonstration of Regex Engine Match Movements

For an HTML or EPUB HTML or EPUB version of the book/excerpt

The example immediately following provides a visual demonstration of the regex engine's matching moves in both the SVG and GIF versions, rather than a textual description.

SVG- differs from GIF demonstration in its clarity and stopability, which helps to perceive all appropriate movements. This gives an idea of what steps are being performed and helps figure out the logic behind the scenes.

Links to these and also to download the entire book will be provided upon purchase of the book. There are a total of 6 SVG demonstrations.

If you just read the excerpt, you can download the GIF demonstrations from the repository https://github.com/aiadFaris/linux-practical.

Outsourcing SVG to attached HTML pages becomes necessary because, unlike GIFs, having more than one SVG file embedded in the same HTML/EPUB seems to confuse the browser/EPUB reader regardless of the size of the book.

Therefore, the other 5 SVG demonstrations are embedded in HTML pages that allow backtrack links to and from the full book.

For an PDF version of the book/excerpt

Since embedding visual demonstrations in PDF is fundamentally not possible, both versions of the visual demonstration are out sourced. The above clause regarding complete book and sample reading also applies here.

Visualizing the match of an unspecified regular expression. See attached **own.html**, which generally visualizes the regex debugger's matching steps as a pauseable animation in the SVG version.

An image of greedy quantifier SVG demonstration just as an example to give an idea:

Greedy Quantifier

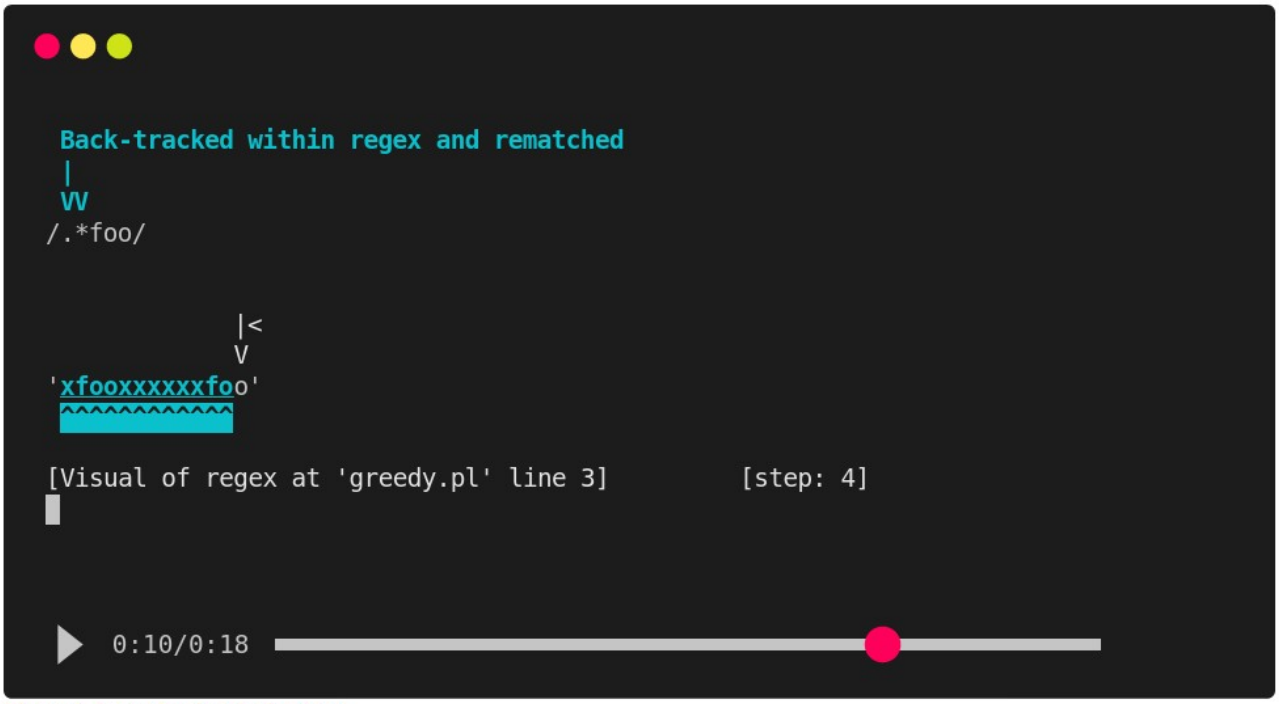


Figure 1. Regex Engine, Perl flavor:

Pages

342

358

are skipped

Regex engine when using Greedy Quantifier

```
$_ = 'xfooxxxxxxfoo';
if (/.*foo/) {
  print "It matched!\n";
}
```

Pattern

.*foo

Test String

xfooxxxxxxfoo

```
Match 1:  0-13  xfooxxxxxxfoo
```

Visualizing the match of a regular expression. See attached **greedy.html**, which visualizes the regex debugger's matching steps for greedy quantifier as a pauseable animation in the SVG version.

Non-greedy (lazy or reluctant) Quantifier

```
$_ = 'xfooxxxxxxfoo';
if (/.*?foo/) {
  print "It matched!\n";
}
```

Pattern

.*?foo

Test String

xfooxxxxxxfoo

```
Match 1:  0-4   xfoo
Match 2:  4-13  xxxxxxfoo
```

Visualizing the match of a regular expression. See attached **reluctant.html**, which visualizes the regex debugger's matching steps for reluctant quantifier as a pauseable animation in the SVG version.

Here's a visual representation of what both patterns matched:

```
  ----VidleXgrabbyVtardyX----
      \____/l      \____/l      l = lazy
      \_________g_______/       g = greedy
#Or
     [VidleX]grabby[VtardyX]
      \_VidleXgrabbyVidleX_/    g = greedy
```

Example based partly on answer made by polygenelubricants

```
$ echo -e '----VidleXgrabbyVtardyX----' | rg 'V.*'  -r '[${0}]'
  ----[VidleXgrabbyVtardyX----]

$ echo -e '----VidleXgrabbyVtardyX----' | rg 'V.*X'  -r '[${0}]'
  ----[VidleXgrabbyVtardyX]----

$ echo -e '----VidleXgrabbyVtardyX----' | rg 'V.*?X'  -r '[${0}]'
  ----[VidleX]grabby[VtardyX]----

$ echo -e '----VidleXgrabbyVtardyX----' | rg 'V.+?X'  -r '[${0}]'
  ----[VidleX]grabby[VtardyX]----
```

Visualizing the match of a regular expression. See attached **greedyVSlazy.html**, which visualizes the regex debugger's matching steps for the greedy vs lazy quantifier as a pauseable animation in the SVG version.

Continuation Non-greedy Quantifiers

- With zero, one and two question marks

```
$ echo 'feet' | rg -o 'f.*' -r '[${0}]'
  [feet]

$ echo 'feet' | rg 'f.?e' -r '[${0}]'
  [fee]t

$ echo 'feet' | rg 'f.??e' -r '[${0}]'
  [fe]et
```

```
$ echo '12345' | rg -o '\d{3,5}'
  12345
$ echo '12345' | rg -o '\d{3,5}?'
```

```
123
```

Possessive quantifier '.*+'

It is just like the greedy quantifier, but it doesn’t backtrack.

Pattern

.*+foo

Test String

xfooxxxxxxfoo

Match: **It fails in 28 steps**

Pages

364

367

are skipped

```
$ printf 'reassure\ndodo' | grep -xE '([a-z].)\1'
  dodo

$ printf 'anan\ndodo' | grep -E '([a-z].)\1'
  anan
  dodo
```

Named Capturing Group

(?<name> subexpression)

or

(?'name' subexpression)

Captures the matched subexpression into a named group: (?P<NAME>group) or (?<name>)

Named capture groups conveniently use names instead of numbers for back-referencing. The naming done by prefixing (?P<NAME>group).

NAME must be an alphanumeric sequence starting with a letter. You can reference the contents of the group with the named backreference (?P=name)

```
$ echo 'anan,dodo' | rg -P '(?<name>[^,]+)\k<name>' -r '[${0}]'
[anan],[dodo]
```

```
$ echo '192.168.172.41' | rg
'(?P<fr>[^\.]{3}\.)(?P<sn>[^\.]{3}\.)([^\.]{3}\.)([^\.]{1,3}$)' -r
'[$fr]/$sn/($3){$4}'
  [192.]/168./(172.){41}
```

'(f)(?P<s>e)(e)(?P<u>d)' matches 'feed'

1. 1st Capturing Group (f) f matches the character f
2. Named Capture Group s (?P<s>e) s matches the character e
3. 3rd Capturing Group (e) e matches the character e
4. Named Capture Group u (?P<u>d) u matches the character d

```
$ echo 'feed' | rg '(f)(?P<s>e)(e)(?P<u>d)' -r '/$1/ ($s) [$u] <$3>'
  /f/ (e) [d] <e>

$ printf "$(date +%F)\n2022-10-01" | rg -P '(?<year>[0-9]{4})-(?<month>[0-9]{2})-
(?<day>[0-9]{2})' -r '$day.$month.$year'
  04.07.2023
  01.10.2022
```

```
$ echo '1,2,3,3,5' | rg -P '^(?:[^,]+,){2}(?<col3>[^,]+),\k<col3>,' -r '[${0}]'
  [1,2,3,3,]5


$  echo '1,2,4,3,3,5' | grep -P '^(?:[^,]+,){3}(?<c3rd>[^,]+),\k<c3rd>,'
1,2,4,3,3,5

$ echo '1,2,4,3,3,6,6,5' | rg -P '^(?:[^,]+,){3}(?<c3>[^,]+),\k<c3>,(?<c6>[^,]+),\k<c6>,' -r
'[${0}]'
[1,2,4,3,3,6,6,]5
```

At index 0: the full match.

Non-capturing Group (:?)

(?:...)

The construct "?:" is used to discard backreferencing.

The syntax is (?:subexpression) or (?:pattern) to define a non-capturing group.

The non-capturing group construct is typically used when a quantifier is applied to a group, but the substrings captured by the group are not of interest.

The regular expression (?:mm)(_bbb), like the regular expression (mm)(-yyyy), always returns mm-yyyy as an overall match. However, the first regex has one capturing group that returns -yyyy as a match, while the second regex hastwo capturing groups that return mm and -yyyy as their respective matches. To highlight this, we need to use the group(0) for returning the overall match and group(1) for the partial match of the second group.

The main advantage of non-capturing groups is that you can add them to a regex without changing the numbering of the capturing groups in the regex as is the case with backreferencing. The regex engine doesn't have to keep track of the text matched by non-capturing groups.

Example 1:

```
$ printf "mm-yyyy" | rg "(?:mm)(-yyyy)" -r '[$0]'
  [mm-yyyy]

$ printf "mm-yyyy" | rg "(?:mm)(-yyyy)" -r '[$1]'
  [-yyy]
```

The same applies to the following examples:

```
$ echo '1st 2nd 3rd third fourth' | rg '([0-9]+)(?:st|nd|rd)' -r '[$0]'
  [1st] [2nd] [3rd] third fourth

$ echo '1st 2nd 3rd third fourth' | rg '([0-9]+)(?:st|nd|rd)' -r '[$1]'
  [1] [2] [3] third fourth
```

Pages

370

372

are skipped

Positive Lookbehind

(?<=component)match

> Lookbehind is to check what's before your regex match, while lookahead does the opposite, so check what's after your match. And the presence or absence of an component before or after the match component is relevant to a match.

Where "match" is the word to be compared and "component" is the element or token to be checked that precedes "match item". The entire lookbehind expression is a bracketed group.

The construct always consists immediately in this order: "(", "?", "<", "=", "component to compare", ")" and "the actual component to match".

With the regex

```
'(?<=C)o'
```

We only want to find an "o", and only if it is preceded by a "C".

This expression will match "o" in Coach, Coast, Coats and Codex but it will not match clock, chock, crypto and chromo.

Now lets see how a regex engine works in case of a positive lookbehind.

Test string:
'That is the book'

Regex:

```
/(?<=a)t/
```

Visualizing the match of an regular expression. See attached **pos_lookbehind.html**, which visualizes the regex debugger's matching steps for Positive Lookbehind as a pauseable animation in the SVG version.

positive lookbehind "?⇐"

```
$ printf '<div><h2>News</h2></div>' | grep -Po '(?<=<div>).*'
  <h2>News</h2></div>
```

```
          ┌──────match──────┐
<div><h2>News</h2></div>
```

Compare Positive Lookbehind with Positive lookahead:

```
$ printf '<div><h2>News</h2></div>' | grep -Po '.*(?=<\/div>)'
  <div><h2>News</h2>
```

```
        match
┌──────────────────┐
<div><h2>News</h2> </div>
```

Positive lookbehind "?⇐"

```
$ printf 'BTC+0.60 ETH-0.23 USDT-0.02 USDC+0.02'  | rg -P '(?<=BTC)[+-]\d+' -r
'[${0}]'
  BTC[+0].60 ETH-0.23 USDT-0.02 USDC+0.02
```

Insertig the tag p in the body area

```
$ printf "<html><body></body></html>" | rg -P '(?<=<body>\W)' -r '${0}p
class=center>paragraph.</p><'
  <html><body><p class=center>paragraph.</p></body></html>
```

Negative lookbehind assertion

The syntax of a negative lookbehind is

'(?<!element)match'

With negative lookahead, the regex engine looks for a specific element -immediately before the main match-, which can be one or more characters or a group, for a match on the element. If that particular element doesn't exist, the regex declares the match as a match,otherwise it simply rejects that match.

For example / (?<!c)d / will match "d" in "ad" and "bd" but it will not match "cd". Or we can say it will not match d in "cd", other wise it will match every "d" which doesn't have an "c" before it.

```
$ printf 'ad bd cd' | rg -P '(?<!c)d' -r '[$0]'
  a[d] b[d] cd
```

Another Example:

Lets say you want to match all cryptocurrencies but BTC. The regex will be '(?<!BTC)[-]0.\d'

```
$ printf 'BTC+0.60 ETH-0.23 USDT-0.02 USDC+0.02' |  rg -Po '(?<!BTC)[+-]0.\d+' -r
'[${0}]'
  [-0.23]
  [-0.02]
```

Atomic group

(?>pattern)

The atomic grouping (?>pattern) has the property "Do not capture" so that the position of the text that matches it is not captured.

In the example /e(?:fg|f)g/, if the "fg" alternative in the group matches, it will never go back and try the "f" alternative.

Comparing with "efghefgg" you can see that it still matches "efgg" at the end of the string and not "efg" at the beginning. If you don't use an atomic group, it can go beyond "fg" and try the "f" alternative and end up matching "efg" at the beginning.

```
$ printf "efgg\nefg"  | rg 'e(?:fg|f)g' -r '[${0}]'
  [efgg]
  [efg]
```

```
$ printf "efgg\nefg"  | rg 'e(?:f|fg)g' -r '[${0}]'
  [efg]g
  [efg]
```

```
$ printf "wy\nway\nay" | grep -P '(?>w|wa)y' -r '[${0}]'
  [wy]
```

Atomic group animated example

```
$_ = 'eedeeedeeeef';
if (/((?>.*)|e*)[df]/) {
  print "It matched!\n";
}
```

Visualizing the match of an regular expression. See attached **atomgrp.html**, which visualizes the regex debugger's matching steps for the Atomic Grouping as a pauseable animation in the SVG version.

Specific topics related to Regex

Since this is a book about Linux, let's first introduce the specific topics related to regular expressions:

Shell Regular Expressions

shell regular expressions

SED

sed is a venerable Stream EDitor. sed is a line-oriented text processing tool. It reads text from an input stream or file, line by line, into an internal buffer called the pattern space. It supports regular expressions.

The basic syntax of the command is:

sed [options] [pattern] [filepath]

Operations Pattern matching using "/" and substitution using s///.

-Search

-Substitution

's/SEARCH STRING/REPLACEMENT/FLAGS'
The "s" stands for substitution and "/" is delimiter which can be another Punctuation from regular expression character class [:punct:]. The SEARCH STRING is most notably REGEXP.

Flags

d delete the pattern space **N** to add the next line to the pattern space **q** to quit **g** global **I** ignore case

Options -e add the script to the commands to be executed -f read from a file

Character sets used as predefined escape sequences

- **\b** The metacharacter \b is an anchor like the caret and the dollar sign. It matches as zero-length at a position that is called a "word boundary".
- **\w** Stands for "word character", usually [A-Za-z0-9_].It can be used to form words.
- **\s** matches all whitespace characters: tab, newline, vertical tab, form feed, carriage return and space
- **\B** is the negated version of \b.
- **\W** is the negated version of \w.
- **\S** matches all non-whitespace characters

Escape sequences

Tab "\t" , carriage return "\r" and newline "\n"

Changing case in replacement part:

\E denotes the end of case conversion
\l convert next character/Expression to lowercase
\u convert next character/Expression to uppercase
\L convert following characters to lowercase, unless \U or \E is used
\U convert following characters to uppercase, unless \L or \E is used

Specifying address ranges

We can specify line or range of lines to print out using the pattern below:

n: Operate only on line number n. Ex.: '3p' would print just the third line.
n,m: Print lines number n to m inclusively. Ex.: '3,5p' would print the 3rd, 4th, and 5th lines.
,m: Print lines from beginning of file to line m inclusively.
n,+m: Print m starting from n . Ex.: '7,+2p' would print lines 7 to 9. (GNU only).
n~m: print starting from line n, than every m line (GNU only). Ex.: '0~2' would print even numbered line.
**$**: Last line. Ex.:'$p' would print just the last line.
!: Negate the specified line in front of it specifying all other lines instead. Ex.: '1,2!p' would print all lines except the first and second.

The Anatomy of a SED Command

```
$ sed -i '2,4 s/see/saw/I' lyric.txt

## Schematic representation
$ sed -i '2,4 s/ see / saw  /  I' lyric.txt
       ^  ^^^ ^^ ^^^ ^ ^^^  ^  ^
       |   |  ||  |  |  |   |  |
       |   |  ||  |  |  |   |  |
       |   |  ||  |  |  |   |  └──────────────── modifier/flag
       |   |  ||  |  |  └───|──────────────────── replacement string/pattern/term
       |   |  ||  └──|──────|──────────────────── search string/pattern/term
       |   |  |└─────└──────└──────────────────── delimiter
       |   |  └──────────────────────────────── command
       |   └──────────────────────────────────── address
       └──────────────────────────────────────── option (in-place)
```

```
$ printf 'x,y,z\n0,0,7\n' | sed 's/,/ /g'
  x y z
  0 0 7
## g stands for global

## Replace with variables

$ bar=excellent; printf '%s\n' foo baz | sed "s/foo/$bar/g"
  excellent
  baz

$ echo '~/' | sed 's@~/@'"$PWD"/'@'
/home/user/
```

Pages

380

404

are skipped

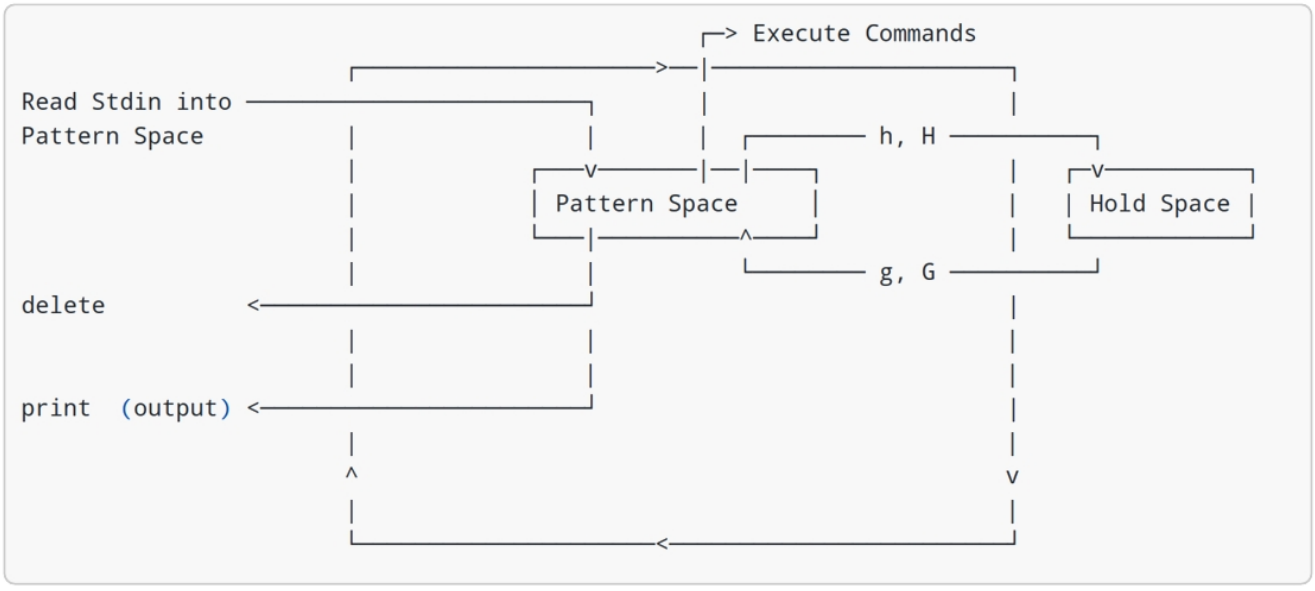
```
## With Character class
$ printf '%s\n' Freddy Fredys\nFredy | sed -nE '/Fred[d]+ys?/p'
  Freddy
```

Use of n, N, G and h commands in pattern space and hold space

D: deletes line from the pattern space until the first newline, and restarts the cycle.
G: appends line from the hold space to the pattern space, with a newline before it.
H: appends line from the pattern space to the hold space, with a newline before it.
N: appends line from the input file to the pattern space.
P: prints line from the pattern space until the first newline.

h: Replace the contents of the hold space with the contents of the pattern space. **H**: Append a newline to the contents of the hold space, and then append the contents of the pattern space to that of the hold space. **g**: Replace the contents of the pattern space with the contents of the hold space. **G**: Append a newline to the contents of the pattern space, and then append the contents of the hold space to that of the pattern space. **x**: Exchange the contents of the hold and pattern spaces.

Functionality of sed with pattern space and hold space



Reverse the line's Order with `1!G;h;$p`

1!: Don't delete the 1. Implicitly delete all other lines.
G: Append a newline to the contents of the pattern space, and then append the contents of the hold space to that of the pattern space.
h: Replace the contents of the hold space with the contents of the pattern space.
$p: Print the last line

```
## The original and          the result
$ cat lyric.txt              |$ sed -n '1!G;h;$p' lyric.txt
  We see trees of green      |  And I think to myself
  Red roses too              |  For me and you
  I see them bloom           |  I see them bloom
  For me and you             |  Red roses too
  And I think to myself      |  We see trees of green
```

```
  1# '1!G' (pattern space)   2# h (hold space)         3# $p iterating > Result
  |                          |  And I think to myself  | And I think to myself
  | Red roses too            |  For me and you         | For me and you
  | We see trees of green    |  I see them bloom       | I see them bloom
  |                          |  Red roses too          | Red roses too
  |                          |  We see trees of green  | We see trees of green
  |                          |                         |


  1# From Hold Space to      2# From Pattern Space     3# (pattern space) Print
     Pattern Space           to Hols Space > reserved
```

```
The N command is less frequently-used commands. It add a newline to the pattern space,
 then append the next line of input to the pattern space.
```

```
Append next line to pattern space and then replace newline character with plus sign.
```

```
if line contains '3', the next line gets appended to pattern space
then  substitute '4' with p appending it to '3'.
```

```
$ seq 5 | sed 'N; /3/ s/\n4/p/'
  1
  2
  3p
  5
```

Listing 221. Sed: Replace on the second line after the match.

```
$ printf "aaa\nbbb\nccc\nddd" | sed "/aaa/{n;n;s/.*/x/}"
aaa
bbb
x
```

```
## What happens in the background when we call the following command?
```

Pages

407

410

are skipped

SSH2

Using SSH with CA certificates and 2FA/MFA

Using passwords came with its downsides, they could be sniffed over the shoulder or keys logged. Another con of password authentication is that usernames and passwords have to be directly transmitted to the server being logged into, thus making this method more prone to hacking. Password could be stored in clear text in database and if hackers breaks in, they can see your password as well. Even if the password is salted and hashed, a hacker could steal all the passwords, brute force the salt and hash, and see if you used that same password for other apps and websites. A final con is that passwords have to be remembered by employees and they thus may get frustrated.

These disadvantages were eliminated with the use of public keys. However, SSH keys come with their own problems. Cons of SSH key authentication.

The first con of using SSH key authentication is that the private key needs to be stored on the device you use to log in. These devices, such as laptops and mobile phones, can be lost or stolen. And if they're not protected properly, hackers can gain access to the private key and eventually the server.

By Using of CA with ssh you can sign a key for a user, and you can login everywhere you trusts the CA as a user. You don't have to copy your SSH key everywhere again.

First you generate a certificate authority (CA) (1) and then use this to issue and cryptographically sign certificates (2) which can authenticate users to hosts (2.b), or hosts to users (2.a).

Generating signing Certificate Authority (CA)

Ideally perform this on dedicated CA machine, in a folder with restricted access and copy them nowhere.

Listing 224. Generating Certificate Authority for the host.

```
$ ssh-keygen -t rsa -b 4096 -f host_ca
  Generating public/private rsa key pair.
  Your identification has been saved in host_ca
  Your public key has been saved in host_ca.pub
```

Used options:
-b: the number of bits in the key to create.
-t: the type of the key to create.
-f: filename

This command will generate private and public key for the hosts "host_ca" and "host_ca.pub". The host_ca file is the host CA's private key and should be protected and kept exclusively in the dedicated machines.

Generating Certificate Authority for the user

It is advantageous for security to create a separate CAs for signing the user's certificates.

Pages

412

482

are skipped

Appendix: Incremental.backup.sh

```
#!/bin/bash

## This Shell script is licensed under the terms of the MIT.
## The script is version 0.2.
#-------------------------------------------------------------------------------

## DISCLAIMER
## THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT
HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER
LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
################################################################################

### CAVEAT
## Make sure to adjust the paths in the variables and the array sections before
running the script and this script is executable. You can run the script with
``./Incremental.backup.sh``. The target drive must not be mounted. The script takes
care of mounting it. If it was accidentally mounted, unmount it.
################################################################################

### Variables
mntPnt=none
DST=lnb-home
srcDvc="/dev/nvme0n1"
dvcID="c072b60f-c042-2d4c-8757-21a859021688"
fltr="rsync_lnb_HOME_filter.txt"
wdy="$(date +%a)"
SRC=$HOME
### Optional
## dstPath="${mntPnt}/${DST}"
# -------------------------


## In addition to functionality, this script also aims to fulfill the didactic aspect.
## The many comments leave nothing new unexplained.

## This is a script to create an incremental backup with rsync
## Rsync&#8217;s options will not be covered here, see rsync above.

### Abstract
## We should look at the backup process from the "point of view" of the script, not in
a
## chronological order that is familiar to people and makes it easier to understand.
## While we can defer cycles by their characteristics, such as aging, we should not
check
```

```
## the second cycle as extra step

### weekdays and weekends
## There is no distinction between weekdays and weekends. Weekdays are used for both.

### Calling this script with ./NAME.sh or 'bash NAME.sh' instead of 'sh NAME.sh' is
preferable.

## Notes and testing are separately documented.
## I wrote numbered notes like [note¹] below. Positionally, they are placed where a
reason for
## explanation is needed Or a reference to testing makes sense.
## However, they themselves are not an integral part of this script.

## To the distribution's environment
## When using the Dolphin file manager GUI to mount a disc, the mount path appears
## to follow the respective entry in the fstab file
## When excluding files, use either a relative or absolute path (but then check)
##  #/home/user/.avfs
##  # Or so when we run the script from $SRC
##  #.avfs

## Syntax
## To remove files, here the escaped "rm" command is used since it is shadowed by
alias.
## The escaped form \rm is avoided since it raise an error

# Limitations:
#1. Preserving the Extended File Attribute
## I haven't found a way to preserve the extended file attribute. I am using Debian
GNU/Linux
## Bullseye with ext4 file system. Despite what some tutorials claim, neither rsync
nor cp nor
## The tar or dar tools helped preserve the extended file attribute
## However, I kept the -"AX" and --rsync-path="/usr/bin/rsync --fake-super" options in
the script
##
## If you find files with such attributes and want to save them to a file:
## $ find ~/ -type f -iname "fl0*" -exec lsattr {} + | grep  -v -- '------------' \
## | sed -e 's/\(-\|e\)//g' > lsattr.lst


### Cycles
##-------------+---------------------------------------+
##             | First Cycle                           |
##             | --------------------------------------+
##             | current (created e.g. Sun)            |
##             | |                                     |
##             | v                                     |
## Folders     | c  Mon  Tue  Wed  Thu  Fri  Sat  Sun  |
## Day number  | 1   2    3    4    5    6    7    8   |
```

```
##-----------------------------------------------------+

## +--------------------------------------+
## |  Next Cycle                          |
## +--------------------------------------+
## |  current (synced Mon)                |
## |  |                                   |
## |  v                                   |
## |  c  Tue  Wed  Thu  Fri  Sat  Sun  Mon  |
## |  9   10   11   12   13   14   15   16  |
## +--------------------------------------+



## The order of the course: The 1. Cycle
## Initial full backup
## 1. On the 1st day
## We start the first day with an initial full backup by filling the "current" folder.
## Rsync is used for this in this script. However, since rsync is usually of
## no use here, "cp -apr" command can be used instead. unless there is a reason to use
rsync.

## From the 2nd day to the 8th day
## We create the folders named after each day with abbreviated weekday name
accordingly
## to the pattern "$(date +%a)", which can also be represented by a variable.
## These folders are named after each day with abbreviated weekday name for
## today's day.
## Let us assume that today is Monday, backups folder will be "Mon/"
## then we will have backups folder as next:
## on 3nd day the "Tue/" and so on till the 8th day that will be Sunday and the folder
created is "Sun".

## On the 9th day it's the "current" folder's turn.

## In the first cycle the folder must have been created before the synchronization.
## Unless the folder therefor in nested nested path, the creation can be implicitly
## done. This script create the folder explicitly, which maybe advantageous in some
cases.

## The order of the course: Repetition cycle.
## This is the cycle from the first iteration and repeats as long as we backup
content.
## Affected folders in these cycles are always once the "current"folder and 7 times
the daily
## weekday folder.
### The criteria that characterize the Repetition cycle:
## 1. Exists the "current" folder and is not empty
## 2. The number of daily backup folders are 0-7 beside the "current" folder.
## 3. At the beginning of the first repeat cycle, the oldest folder must not be older
than 8 days
```

```
## if the backup has not been paused
## If this is our first iteration, it starts with day 8 by syncing the "current"
folder.
## The next day we sync the daily weekday folder "$(date +%a)" this will be "Mon"
## if we started our weekday backup on a Monday.
## We're just making sure the appropriate folder exists so we can start syncing.

## Sources of error
## If the backup process stopped for a while, but new data was produced, then
## it can make it difficult to restore the content in chronological order.
## At least the modification time (on the EXT file system of Linux) differs and
## if we rely on it, we can get unexpected results.
## It is questionable whether it would not be better to start over with the backup.

## prerequisite for synchronization
## The one-time creation of the "current" folder and the creation of the daily weekday
folder is a
## prerequisite for synchronization.

# Synchronizing incrementally & hard link
## The daily folders are synchronized incrementally, i.e. the existing files and
folders in the "current"
## folder are not copied to the daily folder again, but linked via hard link.
##  using the --link-dest option to point to the previous backup so that unchanged
files are hard linked to there.
## The hard links are preserved even if the "current" folder is deleted
## The new content that is not in the "current" folder moves to the folder of the
respective day of the
## week "$(date +%a)".

# Restoring the Backups:
## As a rule, we restore the backup to the full extent in sequence. I.e. the older the
## content, the sooner it will be restored. We&#8217;ll start the restore with
"current" and then
## the contents of the weekday folder based on its age.
##  A quick rsync restore:
##  $ rsync -aruv ./dest/* ./src

# eject command
## The command: ``sudo eject -v /media/sg`` doesn't work

# Notes about work logic, syntax and semantic if you adjust the path.
## Make sure paths don't accumulate through chattered assignments, become empty
through
## unreliability, and unintentionally apply to dangerous paths.
##--------------------------------------------------------------------

# To verify that the platform we're running the script on is the correct device, I
check a device's specific
# property. Here's the main nvme disk is such a property. Note here that '\"' is used
to escape the delimiter '"'.
```

```
# Check the platform by checking the main nvme disk. the Delimiter is masked '\"'
if [ "$(sudo blkid $srcDvc | awk -F\" '{print $2}')" == $dvcID ]; then
    d=$?; echo "It is the lnb, value is: $d";
else
    echo
  "NOT the lnb, if that's what you want, then adjust the path (lnb-home) accordingly.
Now the script will exit"
    exit 1
fi

######## Array
# To be able to use more than one drive to back up contents the array can be expanded
declare -A dvc; dvc[node]=sdb2; dvc[label]=lnx; dvc[uuid]=0a524df6-5863-45d3-b6d8-
d00e23f9bd4e; \
dvc[prtUuid]=30e4d5f7-02;
#dvc[node2]=sdb1;dvc[label2]=bundUSB; dvc[uuid2]=0a6344e2-003e-4215-8915-b65c1646bbaf;
#------

# To ensure that the external hard drive we are going to use is available :
dvcLsblk=$(lsblk -e7 | awk 'FNR == 4 {print $1}' | sed 's/└──//');
if [ "$(sudo blkid "/dev/$dvcLsblk" | awk -F\" '{print $4}')" == "${dvc[uuid]}" ];
then \
   echo "label is: $(sudo blkid "/dev/$dvcLsblk" | awk -F\" '{print $2}')"
else
   echo "The device doesn't seem to be available, maybe it's the wrong one or not
connected!"
fi

# Is the filter rsync_lnb_HOME_filter present?
if [ ! -f "$SRC/$fltr" ]; then
   echo "The rsync_lnb_Home_filter is not found. If necessary, exit with ^C Ctrl+C to
fix the problem"
   # exit 1
fi

# Is the external device already mounted and if so, where is it mounted?
if mount -l | grep -q "${dvc[node]}"; then
   mntNode=$(mount -l | awk '$1 ~ /sdb2/ { print $1 }'); mntOn=$(mount -l | awk
'ENDFILE{ print $3 }')
   printf "The device with node %s, must be %s and already mounted on %s \n" "${
dvc[node]} before starting" "$mntNode" "$mntOn/"

    if [ ! -d /$mntPnt ]; then
        echo "${dvc[label]} mounted on wrong mntPnt";
          sudo umount "$(mount -l | grep "${dvc[label]}" | awk '{print $3}')" && mnt
=true; echo "umount done" || mnt=false;
        if [ "$mnt" == true ]; then
          sudo mount -U "${dvc[uuid]}" /$mntPnt;
          echo "${dvc[label]} must now mounted on /$mntPnt"
        fi
        if (($(mount -l | grep "${dvc[label]}" | awk '{print $3}' | wc -l) > 1)); then
```

```
\
          echo "bind mount occurs:"; mount -l | grep "${dvc[label]}" | awk '{print $1
" on " $3}';
        fi
        echo "now manually mounted as $(mount -l | grep "${dvc[label]}" | awk '{print
$1 " on " $3}')"
    fi
else
    sudo mount -U "${dvc[uuid]}" /$mntPnt
    echo "now manually mounted as $(mount -l | grep "${dvc[label]}" | awk '{print $1 "
!! on " $3}')"
    dvc[node]=$(mount -l | grep "${dvc[label]}" | awk '{print $1}' | cut -d "/" -f 3)
    echo "dvc[node] is ${dvc[node]}"
fi
#

# [Note¹]

# Is the storage space (still) sufficient?
# dvcSz="$(df -h "/dev/${dvc[node]}" | awk '$1 ~ /dev/ { print $4 }' | awk
'{sub(/T$/,""); print}')";
dvcSz="$(df -h "/dev/${dvc[node]}" | awk '$1 ~ /dev/ { print $4 }')";
bkpDUsg="$(sudo du -hs "$SRC" | awk '{ print $1 }')";
echo "capacity of ${dvc[node]} is now $dvcSz, this backup need $bkpDUsg "
#bkpDUsg="$(sudo du -hs "$SRC" | awk '{ print $1 }' | awk '{sub(/G$/,""); print}' )";
printf "Press ^C if you want to cancel\n"; for i in {1..20}; do sleep 1; printf "0%%
\r $i"; done

# [Note²] Testing

### Initial Cycle:: 1. Creating "rsynclog" folder:
##============================================
if [ ! -d /$mntPnt/$DST/rsynclog ]; then
    mkdir -pv /$mntPnt/$DST/rsynclog && echo "again /$mntPnt/$DST/rsynclog is created"
fi

### Initial Cycle:: 1. Creating and Filling "current" folder:
##=========================================================
if [ ! -d /$mntPnt/$DST/current ]; then
# Otherwise, if this were omitted, the shell would create it here implicitly.
    mkdir -vp /$mntPnt/$DST/current  && echo "again /$mntPnt/$DST/current is created"
  printf "The content size of the source \"$SRC\" is %s while that of the target
\"/$mntPnt/$DST/current\" is %s" \
  "$(du -sh "$SRC" | awk '{print $1}')" "$(du -sh "/$mntPnt/$DST/current" | awk
'{print $1}')"

  b="$(echo ""; printf -- '-%.0s' {1..30}; echo "")"
  echo -e "We&#8217;ll sync as follow:$b
    rsync -aPpvuHhAXt --filter="exclude $fltr" --exclude-from=$fltr --rsync-
path="/usr/bin/rsync --fake-super" --stats \
    --filter="exclude $fltr" --exclude-from=$fltr $SRC/ /$mntPnt/$DST/current/
```

```
  $b"

  printf "Press ^C if you want to cancel\n";
  for i in {1..10}; do sleep 1; printf "0%% \r $i"; done

  rsync -aPpvuHhAXt --filter="exclude $fltr" --exclude-from=$fltr --rsync-path
="/usr/bin/rsync --fake-super" --stats \
  --filter="exclude $fltr" --exclude-from=$fltr "$SRC"/ /$mntPnt/$DST/current/
fi

### Initial Cycle:: 2. Creating and Filling the today's weekday folder (every day):
##================================================================================
## and the $wdy folder doesn't yet exist, then create it:


## If the "current" folder, exists, isn't empty (arbitrarily > 5) and the "current"
folder more than 2 hours old
## and the "current" folder more than 0 days old, otherwise the $(date +%a) folder
##  will be synced right after the "current" folder is synced. If both folders are
##  synchronized at the same time, the $(date +%a) folder will not contain anything
##  other than what "current" folder has in terms of content. Ex.:
### mkdir -v dst/current2 && touch -d "2 hours ago" dst/current2

if [ -d /$mntPnt/$DST/current ] && (($(fdfind -d 2 . "/$mntPnt/$DST/current/" | wc -l)
> 5 )) \
&& (($(fdfind -td "current" --changed-before 2h | wc -l) > 0 )); then
# # We need to anticipate synchronization and set the possibility for writing log now
# # Since /$mntPnt/$DST/rsynclog is already created we can created the log file in it.
  if [ ! -f "/$mntPnt/$DST/rsynclog/$wdy.log" ]; then
    touch -- "/$mntPnt/$DST/rsynclog/$wdy.log" && echo "/$mntPnt/$DST/rsynclog/
$wdy.log is created" || echo "$wdy.log exists already"
  fi
  if [ ! -d /$mntPnt/$DST/"$wdy" ]; then
    mkdir -pv /$mntPnt/$DST/"$wdy" && echo "yes /$mntPnt/$DST/$wdy is created" || echo
"yes /$mntPnt/$DST/$wdy exists already"
  fi

  # check path
  b="$(echo ""; printf -- '-%.0s' {1..30}; echo "")"
  echo -e "We&#8217;ll sync as follow:$b
  rsync -aPpvuHhAXt --delete --filter="exclude $fltr"  --exclude-from=$fltr --rsync
-path="/usr/bin/rsync --fake-super" --stats \
  --link-dest="/$mntPnt/$DST/current"  --log-file="/$mntPnt/$DST/rsynclog/"$wdy"".log
$SRC/ /$mntPnt/$DST/$wdy/
  $b"

  printf "\n again %s \e[7m --link-dest="/$mntPnt/$DST/current" \e[0m ...  \e[7m $SRC/
\e[0m .. \e[7m  /$mntPnt/$DST/$wdy/\e[0m ...\n"

  printf "Press ^C if you want to cancel\n";
  for i in {1..20}; do sleep 1; printf "0%% \r $i"; done
```

```
  rsync -aPpvuHhAXt --delete --filter="exclude $fltr"  --exclude-from=$fltr --rsync
-path="/usr/bin/rsync --fake-super" --stats \
  --link-dest="/$mntPnt/$DST/current"  --log-file="/$mntPnt/$DST/rsynclog/$wdy".log
"$SRC"/ /$mntPnt/$DST/"$wdy"/
fi

## dry-run: rsync -apvn /$mntPnt/$DST/current/ $DST/"$wdy"/

### Next cycle:: 1. Syncing the "current" folder.
##===============================================================


##We recognize the new cycle every day by the fact that the respective today's folder
is not empty and is 10 days old. As
##a weekday folder we describe its age comparison as follows:

#---
# Is the current folder is present and at least before 7 days?
# # Actually at this time of check it must be 8 days old.            AND
# Is the current folder is not empty (arbitrarily has > 5 items).    AND
# Is there a 7 days old weekday backup folder. Actually at
# # this time of check it must be 8 days old.                        AND
# Are there more than 6 weekday backup folders?
# # Actually at this time of check they must be 7 folders.

if [[ "$(fdfind -td -d 1 --regex '^current' --changed-before 7d /$mntPnt/$DST/)" ]] &&
\
(($(fdfind -d 2 . "/$mntPnt/$DST/current/" | wc -l) > 5 )) && \
[[ "$(fdfind --regex '^[A-W]+[a-z]{2}$' --changed-before 7d /$mntPnt/$DST/)" ]] && \
(($(fdfind --regex '^[A-W]+[a-z]{2}$' /$mntPnt/$DST/ | wc -l) > 6 )); then
  echo "The current folder exists, 7 days old, not empty (with > 5), there is a 7 days
weekday old folder and 7 weekday folders".



    echo "7 weekday folders are found, "/$mntPnt/$DST/current" will be synced";
# ## First delete the respective log file of the day
    if [ -f "/$mntPnt/$DST/rsynclog/current.log" ]; then "rm" "/$mntPnt/
$DST/rsynclog/current.log"; fi
    # Create a new log file to be able to log:
    touch -- "/$mntPnt/$DST/rsynclog/current.log" && echo "/$mntPnt/
$DST/rsynclog/current.log is created"

    b="$(echo ""; printf -- '-%.0s' {1..30}; echo "")"
    echo -e "We&#8217;ll sync as follow:$b
    rsync -aPpvuHhAXt --delete --filter="exclude $fltr"  --exclude-from=$fltr --rsync
-path="/usr/bin/rsync --fake-super" --stats --log-file="/$mntPnt/$DST
/rsynclog/current".log
    $SRC/ /$mntPnt/$DST/current/
    $b"
```

```
    printf "Press ^C if you want to cancel\n";
    for i in {1..20}; do sleep 1; printf "0%% \r $i"; done

    rsync -aPpvuHhAXt --delete --filter="exclude $fltr"  --exclude-from=$fltr --rsync
-path="/usr/bin/rsync --fake-super" --stats --log-file="/$mntPnt/
$DST/rsynclog/current".log  "$SRC"/ /$mntPnt/$DST/current/
fi

# Now this is how to do:
# $ date
# Thu 11 May 2023 11:26:46 PM CEST
# The Date to required can only relay on modification time xx?
# $ fdfind -td -d 1 --regex '^current$'  --changed-before 0d .
#   current
# ? # that said how to say 7 days old?
# $ for i in *; do echo "$(stat -c %n $i), $(stat -c %y $i | cut -d' ' -f1)"; done
# Thu, 2023-05-04
# Fri, 2023-05-05
# Mon, 2023-05-08
# Sat, 2023-05-06
# Sun, 2023-05-07
# Tue, 2023-05-09
# Wed, 2023-05-10

# Since we've today Sat, 2023-05-13:01:35, the oldest weekday dir is the previous Sat
2023-05-05, thus xxx?
# current's modification time must have been the day before
# touch -d "2023-05-05 01:18" current
#---

### Next cycle:: 2. Syncing the today's weekday folder (every day):
##============================================================================
##We recognize the new cycle every day by the fact that the respective today's folder
is not empty and is 10 days old. As
##a weekday folder we describe its age comparison as follows:
if [[ "/$mntPnt/$DST/$(date --date="7 days ago" +"%a")" == "$(fdfind --regex '[A-
W]+[a-z]{2}' --changed-before 7d /$mntPnt/$DST/)" ]]; then
    rlct="$(fdfind --regex '[A-W]+[a-z]{2}' --changed-before 7d /$mntPnt/$DST/)" &&
echo "7 days old weekday folder \"${rlct:?}\" exists."
    echo "The 7 day old folder ${rlct:?} is not empty because it contains more than 3
files/folders (arbitrarily > 3)."

## First delete the respective log file of the day
    if [ -f "/$mntPnt/$DST/rsynclog/$wdy.log" ]; then "rm" "/$mntPnt/$DST
/rsynclog/$wdy.log"; fi
#     # Create a new log file to be able to log:
      touch -- "/$mntPnt/$DST/rsynclog/$wdy.log" && echo "/$mntPnt/$DST/rsynclog/
$wdy.log is created"

#     # The creation of the log file is only repeated because inexplicable errors and
```

```
subsequent errors occur at runtime:
#     # I assume the reason is elsewhere --delete
#     # Err. # rsync: [client] failed to open log-file lnb-home/rsynclog/Tue.log: No
such file or directory (2)
#       # Err. # Ignoring "log file" setting.
      if [ ! -f "/$mntPnt/$DST/rsynclog/$wdy.log" ]; then touch "/$mntPnt/
$DST/rsynclog/$wdy.log" && \
      echo "$wdy.log is created on the 2nd pass"; fi

      b="$(echo ""; printf -- '-%.0s' {1..30}; echo "")"
      echo -e "We&#8217;ll sync as follow:$b

      rsync -aPpvuHhAXt --rsync-path="/usr/bin/rsync --fake-super" --stats \
      --link-dest=/$mntPnt/$DST/current --filter="exclude $fltr"  --exclude-from=$fltr
\
      --log-file=/$mntPnt/$DST/rsynclog/$wdy.log $SRC/ $DST/$wdy/
    $b"

    printf "Press ^C to cancel\n"
    for i in {1..20}; do sleep 1; printf "0%% \r $i"; done

    rsync -aPpvuHhAXt --rsync-path="/usr/bin/rsync --fake-super" --stats \
    --link-dest="/$mntPnt/$DST/current" --filter="exclude $fltr"  --exclude-from=$fltr
\
    --log-file="/$mntPnt/$DST/rsynclog/$wdy".log "$SRC"/ $DST/"$wdy"/
fi

## [Note³]
## [Note□]

b="$(echo ""; printf -- '-%.0s' {1..30}; echo "")"
echo "Random Sample"
slsts="$SRC/it/lists";
if [ -d "$slsts" ]; then
   {
    echo -e "$b
    # Home Backup
    $b"
    echo -e "$(date +%a-%b-%d-%Y_%H-%M) :\n\n"
    echo -e "\ndiff -rq $slsts /$mntPnt/$DST/current/it/lists :\n";
    diff -rq "$slsts" /$mntPnt/$DST/current/it/lists
    echo -e "$b
    # md5deep verify md5
    $b"
    md5deep -rl "$slsts" | head -3 > /$mntPnt/$DST/rsynclog/crnt_lists.dm5
    md5sum -c /$mntPnt/$DST/rsynclog/crnt_lists.dm5
   } >> /$mntPnt/$DST/rsynclog/crnt_diff_slsts.log

   {
    echo -e "$b
    # Home Backup
```

```
    $b"
    echo -e "$(date +%a-%b-%d-%Y_%H-%M) :\n\n"
    echo -e "\ndiff -rq $slsts /$mntPnt/$DST/$wdy/it/lists :\n";
    diff -rq "$slsts" /$mntPnt/$DST/"$wdy"/it/lists
    echo -e "$b
    # md5deep verify md5
    $b"
    md5deep -rl "$slsts" | tail -3 > /$mntPnt/$DST/rsynclog/"$wdy"_lists.dm5
    md5sum -c /$mntPnt/$DST/rsynclog/"$wdy"_lists.dm5
    echo -e "\n\n"
   } >> /$mntPnt/$DST/rsynclog/"$wdy"_diff_slsts.log

  else
   printf " No data transfer took place";
fi

printf "Cat crnt_diff_slsts.log:\n\n%s\n" "$(cat /$mntPnt/
$DST/rsynclog/crnt_diff_slsts.log)"
printf "Cat "$wdy"_diff_slsts.log:\n\n%s\s" "$(cat /$mntPnt/$DST/rsynclog/"$wdy
"_diff_slsts.log)"

echo -e "\ncurrent folder has $(find /$mntPnt/$DST/current/* -type f -links +1 | wc -
l) hard links"

for i in $(fdfind -d 1 -td --regex '[A-W]+[a-z]{2}' /$mntPnt/$DST/); do \
echo "$i has $(find "${i}"/* -type f -links +1 | wc -l) hard links"; done

echo "The size of parent folder $(du -sh /$mntPnt/$DST/)"

##=== END OF THE SCRIPT ================

### - Notes::
## ========

# Check a success syncing by having an Overview with:
## $ ls -RF

# Testing check
## $ date
##   Mon 17 Apr 2023 12:01:18 PM CEST
## $ touch -d "7 days ago" "$(date +%a -d "-7 days").log"
## $ ls -l "$(date +%a -d "-7 days").log"
##   -rw-r--r-- 1 user user 0 Apr 10 11:57 Mon.log

# Explanatory representation:
##    ┌──────────────── cmd ──────────────┐      ┌────────── filename ──────────┐
## $ touch -d "7 days ago" "$(date +%a -d "-7 days").log"
## Which is the same as:
## $ touch -d "7 days ago" "$(date +%a).log"
## Or:
## $ touch -d "7 days ago" Mon.log
```

```
# find contents according to their age
## $ mkdir -v xmple{01..03}
##   mkdir: created directory 'xmple01'
##   mkdir: created directory 'xmple02'
##   mkdir: created directory 'xmple03'

## $ touch -d "7 days ago" xmple0*
## $ find . -type d -mtime +6
##   ./xmple01
##   ./xmple02
##   ./xmple03

# Remove non-empty folders with "find.* -delete" instead of "rm -rf"
## In stressful situations, I have had disastrous experiences with the
## "rm -rf" command. I only use it in modified form. A good alternative
## is the "find" command with the "-delete" option
##
## $ mkdir -v xmple{01..02}
##   mkdir: created directory 'xmple01'
##   mkdir: created directory 'xmple02'
##
## $ fortune | tee xmple0{1,2}/fl0{01..04}.txt
##   Q:      What do you call a WASP who doesn't work for his father, isn't a
##   ...
##
## $ ls -R
##   .:
##   xmple01  xmple02
##   ./xmple01:
##   fl001.txt  fl002.txt  fl003.txt  fl004.txt
##   ./xmple02:
##   fl001.txt  fl002.txt  fl003.txt  fl004.txt
##
## $ find . -name "*" -delete
## $ ls -l
##   total 0

# Check the hard links
## $ ls -l dst/current  | wc -l
##   14
## $ ls -l dst/Sat  | wc -l
##   17
## The number 17 is because 3 new files/folders were added in the dst/Sat folder
## When we moved the new contents elsewhere
## $ mv dst/Sat/fl0* deli/
## $ ls -l dst/Sat  | wc -l
##   14
## $ du -sh dst/current dst/Sat

## The original files of the "current" folder with their original size illustrate the
```

```
change of file types
## that took place in "dst/sat" folder when we compare both.
## The comparison showed that the hard links in the "dst/Sat" folder are smaller than
the original files.
##   100K    dst/current
##   8.0K    dst/Sat

## Besides the hard links have the same inode number that represents the original file
## $ ls -i Sat/
##   20319050 fl01.txt  20319126 fl02.txt  20319127 fl03.txt  20319128 fl04.txt
## Needless to say that the hard links have the same content as the original files

# (In a context of testing)
# Verifying the md5sum of source
## $ md5deep -rl ~/it/lists > /$mntPnt/$DST/slist.md5;
## user@lnb:/tmp/tmpi$ md5sum -c /$mntPnt/$DST/slist.md5
## /home/user/it/lists/03.12.md: OK
## /home/user/it/lists/tab29.08: OK

## [Note¹]
## When in the GUI of Dolphin, the KDE's file manager, the label "lnx" is clicked, the
/dev/sdb2 will also be mounted on the mount path /nono.

# [Note²]
# (According to the calender)
# (In a context of testing)
# Mockup of the daily backup folders created in a week and named by abbreviated
weekday names
## $ for i in {1..7}; do mkdir "$(date +%a -d "-$i days")" && touch -d "$i days ago"
"$(date +%a -d "-$i days")"; done
## $ ls -lt | awk '{print $6, $7, $9}' | sed -z 's/\n/, /g;s/,$/\n/'
##   , May 2 Tue, May 1 Mon, Apr 30 Sun, Apr 29 Sat, Apr 28 Fri, Apr 27 Thu, Apr 26
Wed

## $ date
##   Tue 02 May 2023 10:49:10 PM CEST
##
## $ for i in {1..7}; do set -x mkdir "$(date +%a -d "-$i days")" && touch -d "$i days
ago" "$(date +%a -d "-$i days")"; done; set +x
##   ++ date +%a -d '-1 days'
##   + touch -d '1 days ago' Mon
##   + for i in {1..7}
##   ++ date +%a -d '-2 days'
##   + set -x mkdir Sun
##   ++ date +%a -d '-2 days'
##   + touch -d '2 days ago' Sun
##   ... truncated
##   ++ date +%a -d '-7 days'
##   + touch -d '7 days ago' Tue
##   + set +x
```

```
## [Note³]
## Mocking up $rlct
##  date --date="today" +"%F"
##   2023-04-07
##
##  date --date="7 days ago" +"%F"
##    2023-03-31

# Test
#|if ...|
## if [[ -n $(find dst/ -name "current" -mtime +6) ]]; then echo y; else echo n; fi
n
## if [ $(find dst/ -name "current" -mtime +6 | wc -l) -gt 0 ]; then echo y; else echo
n; fi    n
## if [ $(fdfind -d 2 -td -i current --changed-before 7d "dst/" | wc -l) -gt 0 ]; then
echo y; else echo n; fi    n
## if [[ -n $(fdfind -d 2 -td -i current --changed-before 7d "dst/") ]]; then echo y;
else echo n; fi
## n
## if [[ -n $(fdfind -d 2 -td -i current --changed-before 7d .) ]]; then echo y; else
echo n; fi
## y

##delete
#if [ "$(find dst/ -name "current" -mtime +6 | wc -l)" -gt 0 ]; then
#  echo "current found";
#  if (( $(fdfind --regex '[A-W]+[a-z]{2}' "dst/" | wc -l) > 6 )); then
#    echo "7 weekday folders found too";
#    else
#     echo "No weekday folders found";
#  fi
#else
#   echo "No current found too";
#fi

##
## $ ls dst
##   current  Fri  Mon  Sat  Sun  Thu  Tue  Wed
## $ if ...
##   dst/current
##   current found
##   7 weekday folders found too
### $ mv -v dst/current .
###   renamed 'dst/current' -> './current'

### $ fdfind --regex '[A-W]+[a-z]{2}' "dst/" -x mv -v {} .
#     renamed 'dst/Sat' -> './Sat'
#     renamed 'dst/Mon' -> './Mon'
#     renamed 'dst/Fri' -> './Fri'
#     renamed 'dst/Sun' -> './Sun'
#     renamed 'dst/Tue' -> './Tue'
```

```
#     renamed 'dst/Thu' -> './Thu'
#     renamed 'dst/Wed' -> './Wed'


# Bash: 'if' statement always seems to evaluate first condition as true even when
false (Not confirmable )

# Bash IF statement Why aren't some nested conditions applied?
# Bash IF statement Why are the false conditions in the first nested IF evaluated as
true

# [Note]
## Why should a weekday folder lie around empty and still untreated?
#   if [ -d "$wdy" ] && files=$(ls -qAH -- "$wdy") && [ -z "$files" ]; then ..; fi #
see R.rsync.tmp3.sh if necessary.

# During the boot process there was an error that I investigate with the following
command:
# $ sudo dmesg -T --color=always --level=err,warn  | grep "none"
# [Mon May 15 16:50:54 2023] systemd-fstab-generator[420]: Mount point none is not a
valid path, ignoring.

#The System add the last entry to fstab which cause the error:
#$ cat /etc/fstab
## /etc/fstab: static file system information.
## Use 'blkid' to print the universally unique identifier for a device; this may
## ... # truncated
##/dev/sdb1                                       none        ext4   users     0 0

## END ############################################################################
```

Index