**Assignment 1 Part A**

**Operating Systems Lab**

**Name: Prasad Sanjay Khalkar**

**Roll No : 33138**

**TE-09 L-09**

**Problem Statement:** Study of basic linux Commands: echo, ls, read, cat, touch, test, loops, arithmetic comparison, conditional loops, grep, sed, etc.

**Command 1:** echo – displays a line of text

**SYNOPSIS:**

echo [SHORT-OPTION]... [STRING]...

echo LONG-OPTION

**DESCRIPTION:**

Echo the STRING(s) to standard output.

-n do not output the trailing newline

-e enable interpretation of backslash escapes

-E disable interpretation of backslash escapes (default)

--help display this help and exit

--version

output version information and exit

If -e is in effect, some of the sequences that are recognized are as follows:

\\ backslash

\b backspace

\n newline

\t horizontal tab

\v vertical tab

etc..

**Command 2:** ls – list directory contents

**SYNOPSIS:**

ls [OPTION]... [FILE]...

**DESCRIPTION:**

List information about the FILEs (the current directory by default). Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

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

-a, --all

do not ignore entries starting with .

-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'; 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

-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

--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

-m fill width with a comma separated list of entries

-r, --reverse

reverse order while sorting

-R, --recursive

list subdirectories recursively

-s, --size

print the allocated size of each file, in blocks

-S sort by file size, largest first

--sort=WORD

sort by WORD instead of name: none (-U), size (-S), time (-t), version (-v), extension (-X)

--time=WORD

with -l, show time as WORD instead of default modification time: atime or access or use (-u); ctime or status (-c); also use specified time as sort key if --sort=time (newest first)

--time-style=TIME\_STYLE

time/date format with -l; see TIME\_STYLE below

-t sort by modification time, newest first

-T, --tabsize=COLS

assume tab stops at each COLS instead of 8

-u with -lt: sort by, and show, access time; with -l: show access time and sort by name; otherwise: sort by access time, newest first

-U do not sort; list entries in directory order

-v natural sort of (version) numbers within text

-w, --width=COLS

set output width to COLS. 0 means no limit

-x list entries by lines instead of by columns

-X sort alphabetically by entry extension

-Z, --context

print any security context of each file

-1 list one file per line. Avoid '\n' with -q or -b

--help display this help and exit

--version

output version information and exit

The SIZE argument is an integer and optional unit (example: 10K is 10\*1024). Units are K,M,G,T,P,E,Z,Y (powers of 1024) or KB,MB,... (powers of 1000).

Exit status:

0 if OK,

1 if minor problems (e.g., cannot access subdirectory),

2 if serious trouble (e.g., cannot access command-line argument).

**Command 3:** read – read from a file descriptor

**SYNOPSIS:**

#include <unistd.h>

ssize\_t read(int fd, void \*buf, size\_t count);

**DESCRIPTION:**

read() attempts to read up to count bytes from file descriptor fd into the buffer starting at buf.

On files that support seeking, the read operation commences at the file offset, and the file offset is incremented by the number of bytes read. If the file offset is at or past the end of

file, no bytes are read, and read() returns zero.

If count is zero, read() may detect the errors described below. In the absence of any errors, or if read() does not check for errors, a read() with a count of 0 returns zero and has no

other effects.

According to POSIX.1, if count is greater than SSIZE\_MAX, the result is implementation-defined; see NOTES for the upper limit on Linux.

**RETURN VALUE:**

On success, the number of bytes read is returned (zero indicates end of file), and the file position is advanced by this number. It is not an error if this number is smaller than the number

of bytes requested; this may happen for example because fewer bytes are actually available right now (maybe because we were close to end-of-file, or because we are reading from a pipe, or

from a terminal), or because read() was interrupted by a signal. See also NOTES.

On error, -1 is returned, and errno is set appropriately. In this case, it is left unspecified whether the file position (if any) changes.

**ERRORS:**

EAGAIN The file descriptor fd refers to a file other than a socket and has been marked nonblocking (O\_NONBLOCK), and the read would block. See open(2) for further details on the O\_NONBLOCK

flag.

EAGAIN or EWOULDBLOCK

The file descriptor fd refers to a socket and has been marked nonblocking (O\_NONBLOCK), and the read would block. POSIX.1-2001 allows either error to be returned for this case, and

does not require these constants to have the same value, so a portable application should check for both possibilities.

EBADF fd is not a valid file descriptor or is not open for reading.

EFAULT buf is outside your accessible address space.

EINTR The call was interrupted by a signal before any data was read; see signal(7).

EINVAL fd is attached to an object which is unsuitable for reading; or the file was opened with the O\_DIRECT flag, and either the address specified in buf, the value specified in count, or

the file offset is not suitably aligned.

EINVAL fd was created via a call to timerfd\_create(2) and the wrong size buffer was given to read(); see timerfd\_create(2) for further information.

EIO I/O error. This will happen for example when the process is in a background process group, tries to read from its controlling terminal, and either it is ignoring or blocking SIGTTIN

or its process group is orphaned. It may also occur when there is a low-level I/O error while reading from a disk or tape. A further possible cause of EIO on networked filesystems

is when an advisory lock had been taken out on the file descriptor and this lock has been lost. See the Lost locks section of fcntl(2) for further details.

EISDIR fd refers to a directory.

Other errors may occur, depending on the object connected to fd.

**Command 4:** cat – concatenate files and prints on standard output

**SYNOPSIS:**

cat [OPTION]... [FILE]...

**DESCRIPTION:**

Concatenate FILE(s) to standard output.

With no FILE, or when FILE is -, read standard input.

-A, --show-all

equivalent to -vET

-b, --number-nonblank

number nonempty output lines, overrides -n

-e equivalent to -vE

-E, --show-ends

display $ at end of each line

-n, --number

number all output lines

-s, --squeeze-blank

suppress repeated empty output lines

-t equivalent to -vT

-T, --show-tabs

display TAB characters as ^I

-u (ignored)

-v, --show-nonprinting

use ^ and M- notation, except for LFD and TAB

--help display this help and exit

--version

output version information and exit

**EXAMPLES:**

cat f - g

Output f's contents, then standard input, then g's contents.

cat Copy standard input to standard output.

**Command 5:** touch – change file timestamps

**SYNOPSIS:**

touch [OPTION]... FILE...

**DESCRIPTION:**

Update the access and modification times of each FILE to the current time.

A FILE argument that does not exist is created empty, unless -c or -h is supplied.

A FILE argument string of - is handled specially and causes touch to change the times of the file associated with standard output.

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

-a change only the access time

-c, --no-create

do not create any files

-d, --date=STRING

parse STRING and use it instead of current time

-f (ignored)

-h, --no-dereference

affect each symbolic link instead of any referenced file (useful only on systems that can change the timestamps of a symlink)

-m change only the modification time

-r, --reference=FILE

use this file's times instead of current time

-t STAMP

use [[CC]YY]MMDDhhmm[.ss] instead of current time

--time=WORD

change the specified time: WORD is access, atime, or use: equivalent to -a WORD is modify or mtime: equivalent to -m

--help display this help and exit

--version

output version information and exit

Note that the -d and -t options accept different time-date formats.

**DATE STRING:**

The --date=STRING is a mostly free format human readable date string such as "Sun, 29 Feb 2004 16:21:42 -0800" or "2004-02-29 16:21:42" or even "next Thursday". A date string may contain

items indicating calendar date, time of day, time zone, day of week, relative time, relative date, and numbers. An empty string indicates the beginning of the day. The date string format

is more complex than is easily documented here but is fully described in the info documentation.

**Command 6:** test – checks file types and compare values

**SYNOPSIS:**

test EXPRESSION

test

[ EXPRESSION ]

[ ]

[ OPTION

**DESCRIPTION:**

Exit with the status determined by EXPRESSION.

--help display this help and exit

--version

output version information and exit

An omitted EXPRESSION defaults to false. Otherwise, EXPRESSION is true or false and sets exit status. It is one of:

( EXPRESSION )

EXPRESSION is true

! EXPRESSION

EXPRESSION is false

EXPRESSION1 -a EXPRESSION2

both EXPRESSION1 and EXPRESSION2 are true

EXPRESSION1 -o EXPRESSION2

either EXPRESSION1 or EXPRESSION2 is true

-n STRING

the length of STRING is nonzero

STRING equivalent to -n STRING

-z STRING

the length of STRING is zero

STRING1 = STRING2

the strings are equal

STRING1 != STRING2

the strings are not equal

INTEGER1 -eq INTEGER2

INTEGER1 is equal to INTEGER2

INTEGER1 -ge INTEGER2

INTEGER1 is greater than or equal to INTEGER2

INTEGER1 -gt INTEGER2

INTEGER1 is greater than INTEGER2

INTEGER1 -le INTEGER2

INTEGER1 is less than or equal to INTEGER2

INTEGER1 -lt INTEGER2

INTEGER1 is less than INTEGER2

INTEGER1 -ne INTEGER2

INTEGER1 is not equal to INTEGER2

FILE1 -ef FILE2

FILE1 and FILE2 have the same device and inode numbers

FILE1 -nt FILE2

FILE1 is newer (modification date) than FILE2

FILE1 -ot FILE2

FILE1 is older than FILE2

-b FILE

FILE exists and is block special

-c FILE

FILE exists and is character special

-d FILE

FILE exists and is a directory

-e FILE

FILE exists

-f FILE

FILE exists and is a regular file

-g FILE

FILE exists and is set-group-ID

-G FILE

FILE exists and is owned by the effective group ID

-h FILE

FILE exists and is a symbolic link (same as -L)

-k FILE

FILE exists and has its sticky bit set

-L FILE

FILE exists and is a symbolic link (same as -h)

-O FILE

FILE exists and is owned by the effective user ID

-p FILE

FILE exists and is a named pipe

-r FILE

FILE exists and read permission is granted

-s FILE

FILE exists and has a size greater than zero

-S FILE

FILE exists and is a socket

-t FD file descriptor FD is opened on a terminal

-u FILE

FILE exists and its set-user-ID bit is set

-w FILE

FILE exists and write permission is granted

-x FILE

FILE exists and execute (or search) permission is granted

Except for -h and -L, all FILE-related tests dereference symbolic links. Beware that parentheses need to be escaped (e.g., by backslashes) for shells. INTEGER may also be -l STRING, which

evaluates to the length of STRING.

NOTE: Binary -a and -o are inherently ambiguous. Use 'test EXPR1 && test EXPR2' or 'test EXPR1 || test EXPR2' instead.

**Command 7:** loops

**NAME:**

loop, loop-control - loop devices

**SYNOPSIS:**

#include <linux/loop.h>

**DESCRIPTION:**

The loop device is a block device that maps its data blocks not to a physical device such as a hard disk or optical disk drive, but to the blocks of a regular file in a filesystem or to an‐

other block device. This can be useful for example to provide a block device for a filesystem image stored in a file, so that it can be mounted with the mount(8) command. You could do

$ dd if=/dev/zero of=file.img bs=1MiB count=10

$ sudo losetup /dev/loop4 file.img

$ sudo mkfs -t ext4 /dev/loop4

$ sudo mkdir /myloopdev

$ sudo mount /dev/loop4 /myloopdev

See losetup(8) for another example.

A transfer function can be specified for each loop device for encryption and decryption purposes.

The following ioctl(2) operations are provided by the loop block device:

LOOP\_SET\_FD

Associate the loop device with the open file whose file descriptor is passed as the (third) ioctl(2) argument.

LOOP\_CLR\_FD

Disassociate the loop device from any file descriptor.

LOOP\_SET\_STATUS

Set the status of the loop device using the (third) ioctl(2) argument. This argument is a pointer to loop\_info structure, defined in <linux/loop.h> as:

struct loop\_info {

int lo\_number; /\* ioctl r/o \*/

dev\_t lo\_device; /\* ioctl r/o \*/

unsigned long lo\_inode; /\* ioctl r/o \*/

dev\_t lo\_rdevice; /\* ioctl r/o \*/

int lo\_offset;

int lo\_encrypt\_type;

int lo\_encrypt\_key\_size; /\* ioctl w/o \*/

int lo\_flags; /\* ioctl r/o \*/

char lo\_name[LO\_NAME\_SIZE];

unsigned char lo\_encrypt\_key[LO\_KEY\_SIZE];

/\* ioctl w/o \*/

unsigned long lo\_init[2];

char reserved[4];

};

The encryption type (lo\_encrypt\_type) should be one of LO\_CRYPT\_NONE, LO\_CRYPT\_XOR, LO\_CRYPT\_DES, LO\_CRYPT\_FISH2, LO\_CRYPT\_BLOW, LO\_CRYPT\_CAST128, LO\_CRYPT\_IDEA, LO\_CRYPT\_DUMMY,

LO\_CRYPT\_SKIPJACK, or (since Linux 2.6.0) LO\_CRYPT\_CRYPTOAPI.

The lo\_flags field is a bit mask that can include zero or more of the following:

LO\_FLAGS\_READ\_ONLY

The loopback device is read-only.

LO\_FLAGS\_AUTOCLEAR (since Linux 2.6.25)

The loopback device will autodestruct on last close.

LO\_FLAGS\_PARTSCAN (since Linux 3.2)

Allow automatic partition scanning.

LOOP\_GET\_STATUS

Get the status of the loop device. The (third) ioctl(2) argument must be a pointer to a struct loop\_info.

LOOP\_CHANGE\_FD (since Linux 2.6.5)

Switch the backing store of the loop device to the new file identified file descriptor specified in the (third) ioctl(2) argument, which is an integer. This operation is possible

only if the loop device is read-only and the new backing store is the same size and type as the old backing store.

LOOP\_SET\_CAPACITY (since Linux 2.6.30)

Resize a live loop device. One can change the size of the underlying backing store and then use this operation so that the loop driver learns about the new size. This operation

takes no argument.

LOOP\_SET\_DIRECT\_IO (since Linux 4.10)

Set DIRECT I/O mode on the loop device, so that it can be used to open backing file. The (third) ioctl(2) argument is an unsigned long value. A non-zero represents direct I/O mode.

LOOP\_SET\_BLOCK\_SIZE (since Linux 4.14)

Set the block size of the loop device. The (third) ioctl(2) argument is an unsigned long value. This value must be a power of two in the range [512,pagesize]; otherwise, an EINVAL

error results.

Since Linux 2.6, there are two new ioctl(2) operations:

LOOP\_SET\_STATUS64, LOOP\_GET\_STATUS64

These are similar to LOOP\_SET\_STATUS and LOOP\_GET\_STATUS described above but use the loop\_info64 structure, which has some additional fields and a larger range for some other fields:

struct loop\_info64 {

uint64\_t lo\_device; /\* ioctl r/o \*/

uint64\_t lo\_inode; /\* ioctl r/o \*/

uint64\_t lo\_rdevice; /\* ioctl r/o \*/

uint64\_t lo\_offset;

uint64\_t lo\_sizelimit;/\* bytes, 0 == max available \*/

uint32\_t lo\_number; /\* ioctl r/o \*/

uint32\_t lo\_encrypt\_type;

uint32\_t lo\_encrypt\_key\_size; /\* ioctl w/o \*/

uint32\_t lo\_flags; /\* ioctl r/o \*/

uint8\_t lo\_file\_name[LO\_NAME\_SIZE];

uint8\_t lo\_crypt\_name[LO\_NAME\_SIZE];

uint8\_t lo\_encrypt\_key[LO\_KEY\_SIZE]; /\* ioctl w/o \*/

uint64\_t lo\_init[2];

}; LO\_FLAGS\_READ\_ONLY

The loopback device is read-only.

LO\_FLAGS\_AUTOCLEAR (since Linux 2.6.25)

The loopback device will autodestruct on last close.

LO\_FLAGS\_PARTSCAN (since Linux 3.2)

Allow automatic partition scanning.

LOOP\_GET\_STATUS

Get the status of the loop device. The (third) ioctl(2) argument must be a pointer to a struct loop\_info.

LOOP\_CHANGE\_FD (since Linux 2.6.5)

Switch the backing store of the loop device to the new file identified file descriptor specified in the (third) ioctl(2) argument, which is an integer. This operation is possible

only if the loop device is read-only and the new backing store is the same size and type as the old backing store.

LOOP\_SET\_CAPACITY (since Linux 2.6.30)

Resize a live loop device. One can change the size of the underlying backing store and then use this operation so that the loop driver learns about the new size. This operation

takes no argument.

LOOP\_SET\_DIRECT\_IO (since Linux 4.10)

Set DIRECT I/O mode on the loop device, so that it can be used to open backing file. The (third) ioctl(2) argument is an unsigned long value. A non-zero represents direct I/O mode.

LOOP\_SET\_BLOCK\_SIZE (since Linux 4.14)

Set the block size of the loop device. The (third) ioctl(2) argument is an unsigned long value. This value must be a power of two in the range [512,pagesize]; otherwise, an EINVAL

error results.

Since Linux 2.6, there are two new ioctl(2) operations:

LOOP\_SET\_STATUS64, LOOP\_GET\_STATUS64

These are similar to LOOP\_SET\_STATUS and LOOP\_GET\_STATUS described above but use the loop\_info64 structure, which has some additional fields and a larger range for some other fields:

struct loop\_info64 {

uint64\_t lo\_device; /\* ioctl r/o \*/

uint64\_t lo\_inode; /\* ioctl r/o \*/

uint64\_t lo\_rdevice; /\* ioctl r/o \*/

uint64\_t lo\_offset;

uint64\_t lo\_sizelimit;/\* bytes, 0 == max available \*/

uint32\_t lo\_number; /\* ioctl r/o \*/

uint32\_t lo\_encrypt\_type;

uint32\_t lo\_encrypt\_key\_size; /\* ioctl w/o \*/

uint32\_t lo\_flags; /\* ioctl r/o \*/

uint8\_t lo\_file\_name[LO\_NAME\_SIZE];

uint8\_t lo\_crypt\_name[LO\_NAME\_SIZE];

uint8\_t lo\_encrypt\_key[LO\_KEY\_SIZE]; /\* ioctl w/o \*/

uint64\_t lo\_init[2];

};

/dev/loop-control

Since Linux 3.1, the kernel provides the /dev/loop-control device, which permits an application to dynamically find a free device, and to add and remove loop devices from the system. To

perform these operations, one first opens /dev/loop-control and then employs one of the following ioctl(2) operations:

LOOP\_CTL\_GET\_FREE

Allocate or find a free loop device for use. On success, the device number is returned as the result of the call. This operation takes no argument.

LOOP\_CTL\_ADD

Add the new loop device whose device number is specified as a long integer in the third ioctl(2) argument. On success, the device index is returned as the result of the call. If the

device is already allocated, the call fails with the error EEXIST.

LOOP\_CTL\_REMOVE

Remove the loop device whose device number is specified as a long integer in the third ioctl(2) argument. On success, the device number is returned as the result of the call. If the

device is in use, the call fails with the error EBUSY.

FILES

/dev/loop\*

The loop block special device files.

**EXAMPLE:**

The program below uses the /dev/loop-control device to find a free loop device, opens the loop device, opens a file to be used as the underlying storage for the device, and then associates

the loop device with the backing store. The following shell session demonstrates the use of the program:

$ dd if=/dev/zero of=file.img bs=1MiB count=10

10+0 records in

10+0 records out

10485760 bytes (10 MB) copied, 0.00609385 s, 1.7 GB/s

$ sudo ./mnt\_loop file.img

loopname = /dev/loop5

**Program source:**

#include <fcntl.h>

#include <linux/loop.h>

#include <sys/ioctl.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#define errExit(msg) do { perror(msg); exit(EXIT\_FAILURE); \

} while (0)

int

main(int argc, char \*argv[])

{

int loopctlfd, loopfd, backingfile;

long devnr;

char loopname[4096];

if (argc != 2) {

fprintf(stderr, "Usage: %s backing-file\n", argv[0]);

exit(EXIT\_FAILURE);

}

loopctlfd = open("/dev/loop-control", O\_RDWR);

if (loopctlfd == -1)

errExit("open: /dev/loop-control");

devnr = ioctl(loopctlfd, LOOP\_CTL\_GET\_FREE);

if (devnr == -1)

errExit("ioctl-LOOP\_CTL\_GET\_FREE");

sprintf(loopname, "/dev/loop%ld", devnr);

printf("loopname = %s\n", loopname);

loopfd = open(loopname, O\_RDWR);

if (loopfd == -1)

errExit("open: loopname");

backingfile = open(argv[1], O\_RDWR);

if (backingfile == -1)

errExit("open: backing-file");

if (ioctl(loopfd, LOOP\_SET\_FD, backingfile) == -1)

errExit("ioctl-LOOP\_SET\_FD");

exit(EXIT\_SUCCESS);

}

**Command 8: Arithmetic Comparison:**

a) **Relational Operators:**

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| **-eq** | Checks if the value of two operands are equal or not; if yes, then the condition becomes true. | [ $a -eq $b ] is not true. |
| **-ne** | Checks if the value of two operands are equal or not; if values are not equal, then the condition becomes true. | [ $a -ne $b ] is true. |
| **-gt** | Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true. | [ $a -gt $b ] is not true. |
| **-lt** | Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true. | [ $a -lt $b ] is true. |
| **-ge** | Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then the condition becomes true. | [ $a -ge $b ] is not true. |
| **-le** | Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true. | [ $a -le $b ] is true. |

b) **Boolean Operators:**

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| **!** | This is logical negation. This inverts a true condition into false and vice versa. | [ ! false ] is true. |
| **-o** | This is logical **OR**. If one of the operands is true, then the condition becomes true. | [ $a -lt 20 -o $b -gt 100 ] is true. |
| **-a** | This is logical **AND**. If both the operands are true, then the condition becomes true otherwise false. | [ $a -lt 20 -a $b -gt 100 ] is false. |

c) **String Comparison Operators:**

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| **=** | Checks if the value of two operands are equal or not; if yes, then the condition becomes true. | [ $a = $b ] is not true. |
| **!=** | Checks if the value of two operands are equal or not; if values are not equal then the condition becomes true. | [ $a != $b ] is true. |
| **-z** | Checks if the given string operand size is zero; if it is zero length, then it returns true. | [ -z $a ] is not true. |
| **-n** | Checks if the given string operand size is non-zero; if it is nonzero length, then it returns true. | [ -n $a ] is not false. |
| **str** | Checks if **str** is not the empty string; if it is empty, then it returns false. | [ $a ] is not false. |

**Command 9:** **Conditional Statements**

Conditional Statements: There are total 5 conditional statements which can be used in bash programming

1. if statement
2. if-else statement
3. if..elif..else..fi statement (Else If ladder)
4. if..then..else..if..then..fi..fi..(Nested if)
5. switch statement

a) **If statement:**

if [ expression ]

then

statement

fi

b) **if-else statement:**

if [ expression ]

then

statement1

else

statement2

fi

c) **if..elif..else..fi statement:**

if [ expression1 ]

then

statement1

statement2

.

.

elif [ expression2 ]

then

statement3

statement4

.

.

else

statement5

fi

d) **Nested if :**

if [ expression1 ]

then

statement1

statement2

.

else

if [ expression2 ]

then

statement3

.

fi

fi

e) **switch statement:**

case in

Pattern 1) Statement 1;;

Pattern n) Statement n;;

esac

**Command 10: grep**

**NAME:**

grep, egrep, fgrep, rgrep - print lines that match patterns

**SYNOPSIS:**

grep [OPTION...] PATTERNS [FILE...]

grep [OPTION...] -e PATTERNS ... [FILE...]

grep [OPTION...] -f PATTERN\_FILE ... [FILE...]

**DESCRIPTION:**

grep searches for PATTERNS in each FILE. PATTERNS is one or more patterns separated by newline characters, and grep prints each line that matches a pattern. Typically PATTERNS should be

quoted when grep is used in a shell command.

A FILE of “-” stands for standard input. If no FILE is given, recursive searches examine the working directory, and nonrecursive searches read standard input.

In addition, the variant programs egrep, fgrep and rgrep are the same as grep -E, grep -F, and grep -r, respectively. These variants are deprecated, but are provided for backward

compatibility.

**OPTIONS:**

Generic Program Information

--help Output a usage message and exit.

-V, --version

Output the version number of grep and exit.

Pattern Syntax

-E, --extended-regexp

Interpret PATTERNS as extended regular expressions (EREs, see below).

-F, --fixed-strings

Interpret PATTERNS as fixed strings, not regular expressions.

-G, --basic-regexp

Interpret PATTERNS as basic regular expressions (BREs, see below). This is the default.

-P, --perl-regexp

Interpret PATTERNS as Perl-compatible regular expressions (PCREs). This option is experimental when combined with the -z (--null-data) option, and grep -P may warn of unimplemented

features.

Matching Control

-e PATTERNS, --regexp=PATTERNS

Use PATTERNS as the patterns. If this option is used multiple times or is combined with the -f (--file) option, search for all patterns given. This option can be used to protect a

pattern beginning with “-”.

-f FILE, --file=FILE

Obtain patterns from FILE, one per line. If this option is used multiple times or is combined with the -e (--regexp) option, search for all patterns given. The empty file contains

zero patterns, and therefore matches nothing.

-i, --ignore-case

Ignore case distinctions in patterns and input data, so that characters that differ only in case match each other.

--no-ignore-case

Do not ignore case distinctions in patterns and input data. This is the default. This option is useful for passing to shell scripts that already use -i, to cancel its effects

because the two options override each other.

-v, --invert-match

Invert the sense of matching, to select non-matching lines.

-w, --word-regexp

Select only those lines containing matches that form whole words. The test is that the matching substring must either be at the beginning of the line, or preceded by a non-word

constituent character. Similarly, it must be either at the end of the line or followed by a non-word constituent character. Word-constituent characters are letters, digits, and the

underscore. This option has no effect if -x is also specified.

-x, --line-regexp

Select only those matches that exactly match the whole line. For a regular expression pattern, this is like parenthesizing the pattern and then surrounding it with ^ and $.

-y Obsolete synonym for -i.

General Output Control

-c, --count

Suppress normal output; instead print a count of matching lines for each input file. With the -v, --invert-match option (see below), count non-matching lines.

--color[=WHEN], --colour[=WHEN]

Surround the matched (non-empty) strings, matching lines, context lines, file names, line numbers, byte offsets, and separators (for fields and groups of context lines) with escape

sequences to display them in color on the terminal. The colors are defined by the environment variable GREP\_COLORS. The deprecated environment variable GREP\_COLOR is still

supported, but its setting does not have priority. WHEN is never, always, or auto.

-L, --files-without-match

Suppress normal output; instead print the name of each input file from which no output would normally have been printed. The scanning will stop on the first match.

-l, --files-with-matches

Suppress normal output; instead print the name of each input file from which output would normally have been printed. The scanning will stop on the first match.

-m NUM, --max-count=NUM

Stop reading a file after NUM matching lines. If the input is standard input from a regular file, and NUM matching lines are output, grep ensures that the standard input is

positioned to just after the last matching line before exiting, regardless of the presence of trailing context lines. This enables a calling process to resume a search. When grep

stops after NUM matching lines, it outputs any trailing context lines. When the -c or --count option is also used, grep does not output a count greater than NUM. When the -v or

--invert-match option is also used, grep stops after outputting NUM non-matching lines.

-o, --only-matching

Print only the matched (non-empty) parts of a matching line, with each such part on a separate output line.

-q, --quiet, --silent

Quiet; do not write anything to standard output. Exit immediately with zero status if any match is found, even if an error was detected. Also see the -s or --no-messages option.

-s, --no-messages

Suppress error messages about nonexistent or unreadable files.

Output Line Prefix Control

-b, --byte-offset

Print the 0-based byte offset within the input file before each line of output. If -o (--only-matching) is specified, print the offset of the matching part itself.

-H, --with-filename

Print the file name for each match. This is the default when there is more than one file to search.

-h, --no-filename

Suppress the prefixing of file names on output. This is the default when there is only one file (or only standard input) to search.

--label=LABEL

Display input actually coming from standard input as input coming from file LABEL. This can be useful for commands that transform a file's contents before searching, e.g., gzip -cd

foo.gz | grep --label=foo -H 'some pattern'. See also the -H option.

-n, --line-number

Prefix each line of output with the 1-based line number within its input file.

-T, --initial-tab

Make sure that the first character of actual line content lies on a tab stop, so that the alignment of tabs looks normal. This is useful with options that prefix their output to the

actual content: -H,-n, and -b. In order to improve the probability that lines from a single file will all start at the same column, this also causes the line number and byte offset

(if present) to be printed in a minimum size field width.

-u, --unix-byte-offsets

Report Unix-style byte offsets. This switch causes grep to report byte offsets as if the file were a Unix-style text file, i.e., with CR characters stripped off. This will produce

results identical to running grep on a Unix machine. This option has no effect unless -b option is also used; it has no effect on platforms other than MS-DOS and MS-Windows.

-Z, --null

Output a zero byte (the ASCII NUL character) instead of the character that normally follows a file name. For example, grep -lZ outputs a zero byte after each file name instead of the

usual newline. This option makes the output unambiguous, even in the presence of file names containing unusual characters like newlines. This option can be used with commands like

find -print0, perl -0, sort -z, and xargs -0 to process arbitrary file names, even those that contain newline characters.

Context Line Control

-A NUM, --after-context=NUM

Print NUM lines of trailing context after matching lines. Places a line containing a group separator (--) between contiguous groups of matches. With the -o or --only-matching

option, this has no effect and a warning is given.

-B NUM, --before-context=NUM

Print NUM lines of leading context before matching lines. Places a line containing a group separator (--) between contiguous groups of matches. With the -o or --only-matching

option, this has no effect and a warning is given.

-C NUM, -NUM, --context=NUM

Print NUM lines of output context. Places a line containing a group separator (--) between contiguous groups of matches. With the -o or --only-matching option, this has no effect

and a warning is given.

File and Directory Selection

-a, --text

Process a binary file as if it were text; this is equivalent to the --binary-files=text option.

--binary-files=TYPE

If a file's data or metadata indicate that the file contains binary data, assume that the file is of type TYPE. Non-text bytes indicate binary data; these are either output bytes

that are improperly encoded for the current locale, or null input bytes when the -z option is not given.

By default, TYPE is binary, and grep suppresses output after null input binary data is discovered, and suppresses output lines that contain improperly encoded data. When some output

is suppressed, grep follows any output with a one-line message saying that a binary file matches.

If TYPE is without-match, when grep discovers null input binary data it assumes that the rest of the file does not match; this is equivalent to the -I option.

If TYPE is text, grep processes a binary file as if it were text; this is equivalent to the -a option.

When type is binary, grep may treat non-text bytes as line terminators even without the -z option. This means choosing binary versus text can affect whether a pattern matches a file.

For example, when type is binary the pattern q$ might match q immediately followed by a null byte, even though this is not matched when type is text. Conversely, when type is binary

the pattern . (period) might not match a null byte.

Warning: The -a option might output binary garbage, which can have nasty side effects if the output is a terminal and if the terminal driver interprets some of it as commands. On the

other hand, when reading files whose text encodings are unknown, it can be helpful to use -a or to set LC\_ALL='C' in the environment, in order to find more matches even if the matches

are unsafe for direct display.

-D ACTION, --devices=ACTION

If an input file is a device, FIFO or socket, use ACTION to process it. By default, ACTION is read, which means that devices are read just as if they were ordinary files. If ACTION

is skip, devices are silently skipped.

-d ACTION, --directories=ACTION

If an input file is a directory, use ACTION to process it. By default, ACTION is read, i.e., read directories just as if they were ordinary files. If ACTION is skip, silently skip

directories. If ACTION is recurse, read all files under each directory, recursively, following symbolic links only if they are on the command line. This is equivalent to the -r

option.

--exclude=GLOB

Skip any command-line file with a name suffix that matches the pattern GLOB, using wildcard matching; a name suffix is either the whole name, or a trailing part that starts with a

non-slash character immediately after a slash (/) in the name. When searching recursively, skip any subfile whose base name matches GLOB; the base name is the part after the last

slash. A pattern can use \*, ?, and [...] as wildcards, and \ to quote a wildcard or backslash character literally.

--exclude-from=FILE

Skip files whose base name matches any of the file-name globs read from FILE (using wildcard matching as described under --exclude).

--exclude-dir=GLOB

Skip any command-line directory with a name suffix that matches the pattern GLOB. When searching recursively, skip any subdirectory whose base name matches GLOB. Ignore any

redundant trailing slashes in GLOB.

-I Process a binary file as if it did not contain matching data; this is equivalent to the --binary-files=without-match option.

--include=GLOB

Search only files whose base name matches GLOB (using wildcard matching as described under --exclude).

-r, --recursive

Read all files under each directory, recursively, following symbolic links only if they are on the command line. Note that if no file operand is given, grep searches the working

directory. This is equivalent to the -d recurse option.

-R, --dereference-recursive

Read all files under each directory, recursively. Follow all symbolic links, unlike -r.

Other Options

--line-buffered

Use line buffering on output. This can cause a performance penalty.

-U, --binary

Treat the file(s) as binary. By default, under MS-DOS and MS-Windows, grep guesses whether a file is text or binary as described for the --binary-files option. If grep decides the

file is a text file, it strips the CR characters from the original file contents (to make regular expressions with ^ and $ work correctly). Specifying -U overrules this guesswork,

causing all files to be read and passed to the matching mechanism verbatim; if the file is a text file with CR/LF pairs at the end of each line, this will cause some regular

expressions to fail. This option has no effect on platforms other than MS-DOS and MS-Windows.

-z, --null-data

Treat input and output data as sequences of lines, each terminated by a zero byte (the ASCII NUL character) instead of a newline. Like the -Z or --null option, this option can be

used with commands like sort -z to process arbitrary file names.

**Command 11: sed**

**NAME:**

sed - stream editor for filtering and transforming text

**SYNOPSIS:**

sed [OPTION]... {script-only-if-no-other-script} [input-file]...

**DESCRIPTION:**

Sed is a stream editor. A stream editor is used to perform basic text transformations on an input stream (a file or input from a pipeline). While in some ways similar to an editor which

permits scripted edits (such as ed), sed works by making only one pass over the input(s), and is consequently more efficient. But it is sed's ability to filter text in a pipeline which particularly distinguishes it from other types of editors.

-n, --quiet, --silent

suppress automatic printing of pattern space

--debug

annotate program execution

-e script, --expression=script

add the script to the commands to be executed

-f script-file, --file=script-file

add the contents of script-file to the commands to be executed

--follow-symlinks

follow symlinks when processing in place

-i[SUFFIX], --in-place[=SUFFIX]

edit files in place (makes backup if SUFFIX supplied)

-l N, --line-length=N

specify the desired line-wrap length for the `l' command

--posix

disable all GNU extensions.

-E, -r, --regexp-extended

use extended regular expressions in the script (for portability use POSIX -E).

-s, --separate

consider files as separate rather than as a single, continuous long stream.

--sandbox

operate in sandbox mode (disable e/r/w commands).

-u, --unbuffered

load minimal amounts of data from the input files and flush the output buffers more often

-z, --null-data

separate lines by NUL characters

--help

display this help and exit

--version

output version information and exit

If no -e, --expression, -f, or --file option is given, then the first non-option argument is taken as the sed script to interpret. All remaining arguments are names of input files; if no

input files are specified, then the standard input is read.

GNU sed home page: <https://www.gnu.org/software/sed/>. General help using GNU software: <https://www.gnu.org/gethelp/>. E-mail bug reports to: <bug-sed@gnu.org>.

COMMAND SYNOPSIS

This is just a brief synopsis of sed commands to serve as a reminder to those who already know sed; other documentation (such as the texinfo document) must be consulted for fuller descrip‐

tions.

Zero-address ``commands''

: label

Label for b and t commands.

#comment

The comment extends until the next newline (or the end of a -e script fragment).

} The closing bracket of a { } block.

Zero- or One- address commands

= Print the current line number.

a \

text Append text, which has each embedded newline preceded by a backslash.

i \

text Insert text, which has each embedded newline preceded by a backslash.

q [exit-code]

Immediately quit the sed script without processing any more input, except that if auto-print is not disabled the current pattern space will be printed. The exit code argument is a

GNU extension.

Q [exit-code]

Immediately quit the sed script without processing any more input. This is a GNU extension.

r filename

Append text read from filename.

R filename

Append a line read from filename. Each invocation of the command reads a line from the file. This is a GNU extension.

Commands which accept address ranges

{ Begin a block of commands (end with a }).

b label

Branch to label; if label is omitted, branch to end of script.

c \

text Replace the selected lines with text, which has each embedded newline preceded by a backslash.

d Delete pattern space. Start next cycle.

D If pattern space contains no newline, start a normal new cycle as if the d command was issued. Otherwise, delete text in the pattern space up to the first newline, and restart cycle

with the resultant pattern space, without reading a new line of input.

h H Copy/append pattern space to hold space.

g G Copy/append hold space to pattern space.

l List out the current line in a ``visually unambiguous'' form.

l width

List out the current line in a ``visually unambiguous'' form, breaking it at width characters. This is a GNU extension.

n N Read/append the next line of input into the pattern space.

p Print the current pattern space.

P Print up to the first embedded newline of the current pattern space.

s/regexp/replacement/

Attempt to match regexp against the pattern space. If successful, replace that portion matched with replacement. The replacement may contain the special character & to refer to that

portion of the pattern space which matched, and the special escapes \1 through \9 to refer to the corresponding matching sub-expressions in the regexp.

t label

If a s/// has done a successful substitution since the last input line was read and since the last t or T command, then branch to label; if label is omitted, branch to end of script.

T label

If no s/// has done a successful substitution since the last input line was read and since the last t or T command, then branch to label; if label is omitted, branch to end of script.

This is a GNU extension.

w filename

Write the current pattern space to filename.

W filename

Write the first line of the current pattern space to filename. This is a GNU extension.

x Exchange the contents of the hold and pattern spaces.

y/source/dest/

Transliterate the characters in the pattern space which appear in source to the corresponding character in dest.

Addresses

Sed commands can be given with no addresses, in which case the command will be executed for all input lines; with one address, in which case the command will only be executed for input lines

which match that address; or with two addresses, in which case the command will be executed for all input lines which match the inclusive range of lines starting from the first address and

continuing to the second address. Three things to note about address ranges: the syntax is addr1,addr2 (i.e., the addresses are separated by a comma); the line which addr1 matched will al‐

ways be accepted, even if addr2 selects an earlier line; and if addr2 is a regexp, it will not be tested against the line that addr1 matched.

After the address (or address-range), and before the command, a ! may be inserted, which specifies that the command shall only be executed if the address (or address-range) does not match.

The following address types are supported:

number Match only the specified line number (which increments cumulatively across files, unless the -s option is specified on the command line).

first~step

Match every step'th line starting with line first. For example, ``sed -n 1~2p'' will print all the odd-numbered lines in the input stream, and the address 2~5 will match every fifth

line, starting with the second. first can be zero; in this case, sed operates as if it were equal to step. (This is an extension.)

$ Match the last line.

/regexp/

Match lines matching the regular expression regexp. Matching is performed on the current pattern space, which can be modified with commands such as ``s///''.

\cregexpc

Match lines matching the regular expression regexp. The c may be any character.

GNU sed also supports some special 2-address forms:

0,addr2

Start out in "matched first address" state, until addr2 is found. This is similar to 1,addr2, except that if addr2 matches the very first line of input the 0,addr2 form will be at

the end of its range, whereas the 1,addr2 form will still be at the beginning of its range. This works only when addr2 is a regular expression.

addr1,+N

Will match addr1 and the N lines following addr1.

addr1,~N

Will match addr1 and the lines following addr1 until the next line whose input line number is a multiple of N.

REGULAR EXPRESSIONS

POSIX.2 BREs should be supported, but they aren't completely because of performance problems. The \n sequence in a regular expression matches the newline character, and similarly for \a,

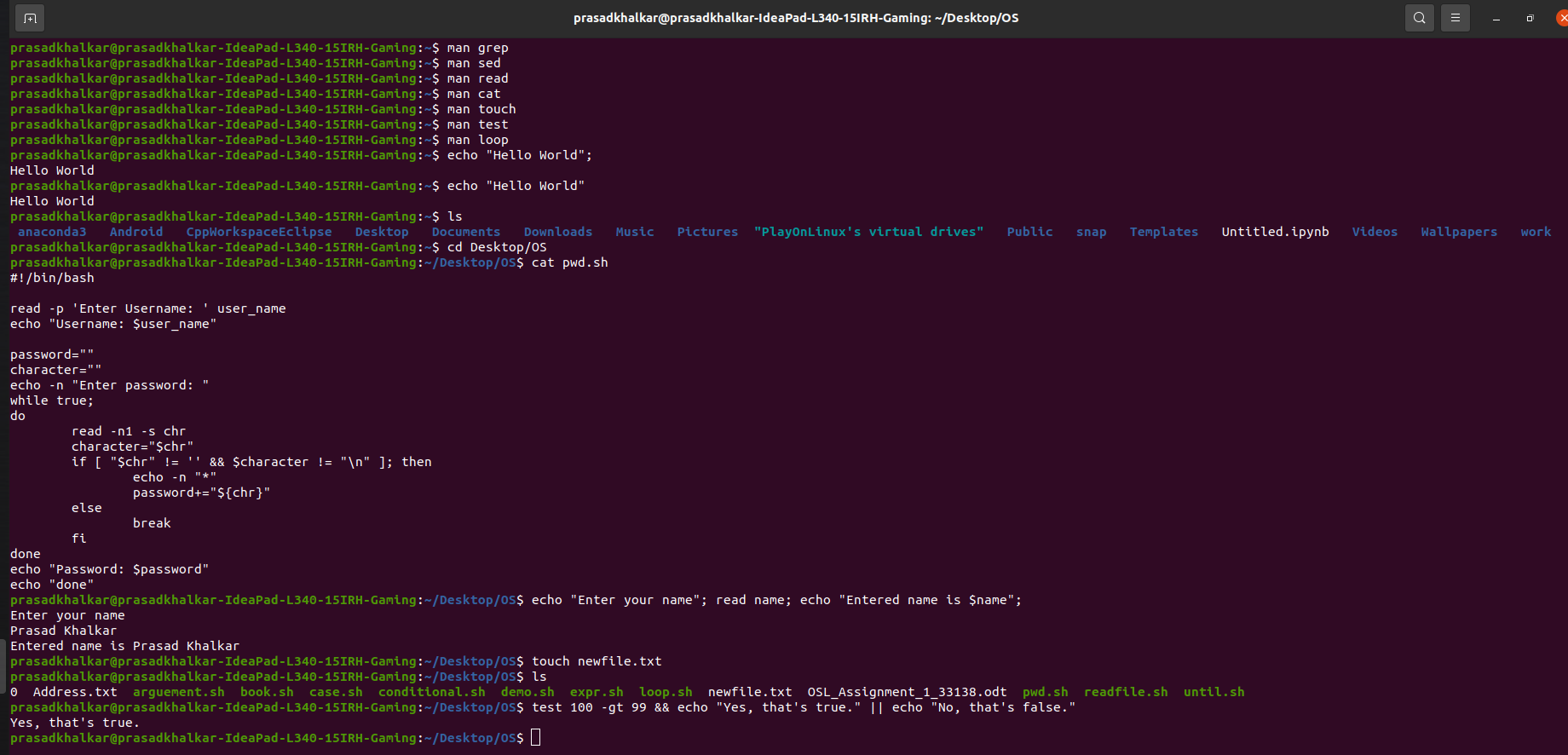
\t, and other sequences. The -E option switches to using extended regular expressions instead; it has been supported for years by GNU sed, and is now included in POSIX.

BUGS

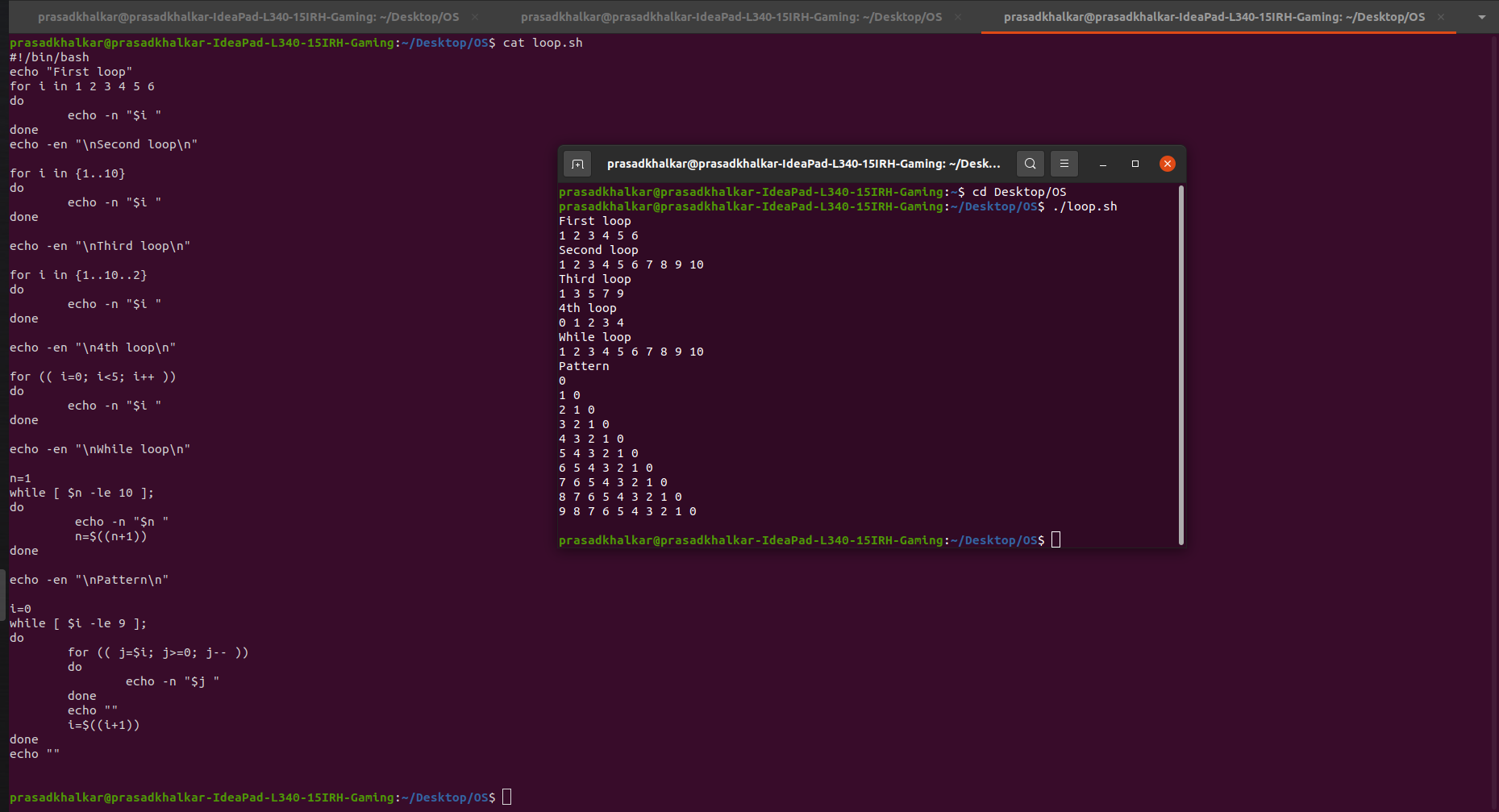
E-mail bug reports to bug-sed@gnu.org. Also, please include the output of ``sed --version'' in the body of your report if at all possible.

**Output from terminal:**

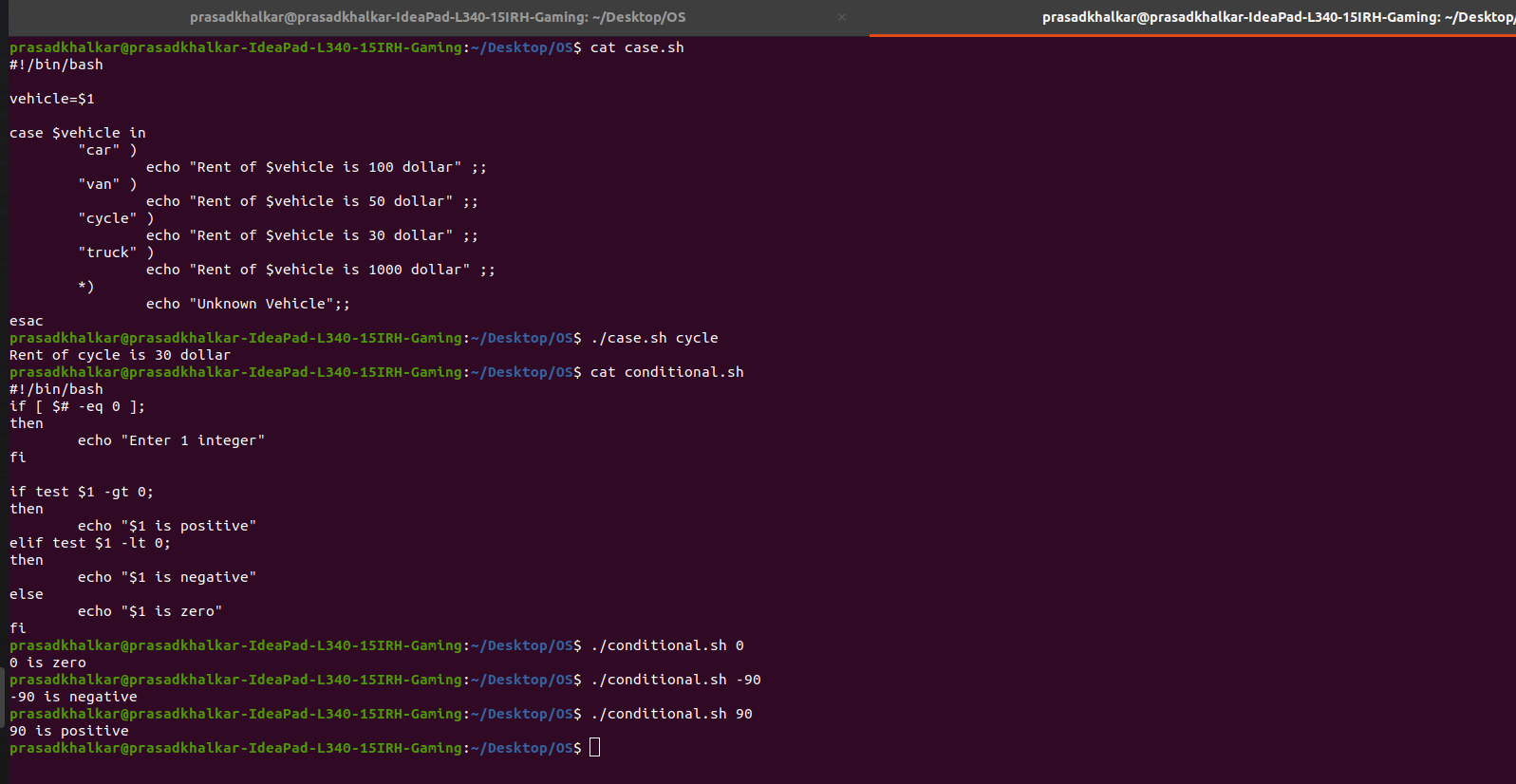
echo, ls, read, cat, touch, test:



Loops:



Conditional and arithmetic comparison:



Grep and sed:

