

CSc 3320: Systems Programming
Spring 2021
Homework
3: Total points 100

Submission instructions:

1. Create a Google doc for each homework assignment submission.
2. Start your responses from page 2 of the document and copy these instructions on page 1.
3. Fill in your name, campus ID and panther # in the fields provided. If this information is missing in your document TWO POINTS WILL BE DEDUCTED per submission.
4. Keep this page 1 intact on all your submissions. If this *submissions instructions* page is missing in your submission TWO POINTS WILL BE DEDUCTED per submission.
5. Each homework will typically have 2-3 PARTS, where each PART focuses on specific topic(s).
6. Start your responses to each PART on a new page.
7. If you are being asked to write code copy the code into a separate txt file and submit that as well.
8. If you are being asked to test code or run specific commands or scripts, provide the evidence of your outputs through a screenshot and copy the same into the document.
9. Upon completion, download a .PDF version of the document and submit the same.

Full Name: Rafid H. Shaon

Campus ID: rshaon1

Panther #: 002-49-7367

10 pts for the neatness factor of your presentation.

PART 1: 30pts

1. For each command tryout at least one example provided in **Chapter 3** of the Unix textbook. Feel free to use your own example. Show the screenshot for each command's output. Present your output in a tabular form with column 1 as index (1, 2, 3...), second column as the command, third as the usage, fourth as the screenshot of the output. You can just show a small snapshot for the output -- we do not need the entire screen's image.

Part II: 30pts

2. For each command tryout at least one example provided in **Chapter 4** of the Unix textbook. Feel free to use your own example. Show the screenshot for each command's output. Present your output in a tabular form with column 1 as index (1,2, 3...), second column as the command, third as the usage, fourth as the screenshot of the output. You can just show a small snapshot for the output -- we do not need the entire screen's image.

Part III: 30pts

3. For each command tryout at least one example provided in **Chapter 5** of the Unix textbook. Feel free to use your own example. Show the screenshot for each command's output. Present your output in a tabular form with column 1 as index (1,2, 3...), second column as the command, third as the usage, fourth as the screenshot of the output. You can just show a small snapshot for the output -- we do not need the entire screen's image.

Chapter 3

Example 1

Command

at

Usage

at time job/command

Used to schedule one-time tasks. The time can be specified in various formats. Unlike cron jobs these jobs are meant to be run a single time.

Screenshots

We schedule a job to run myjob.sh script and then delete the job.

```
$ # Schedule a job
$
$ at 5.45 PM
warning: commands will be executed using /bin/sh
at> sh myjob.sh
at> <EOT>
job 8 at Fri Feb 19 17:45:00 2021
$
$ # List the scheduled jobs (get their job numbers only)
$
$ atq | cut -f1
8
$
$ # Remove the job
$
$ atrm 8
$
```

Example 2

Command

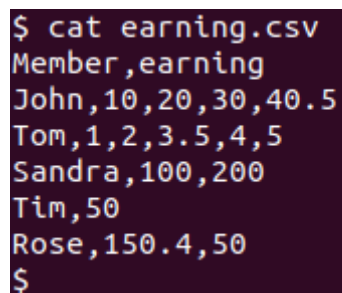
awk

Usage

```
awk '
    BEGIN { actions }
    /pattern/ { actions }
    /pattern/ { actions }
    ...
    END { actions }
' filename(s)
```

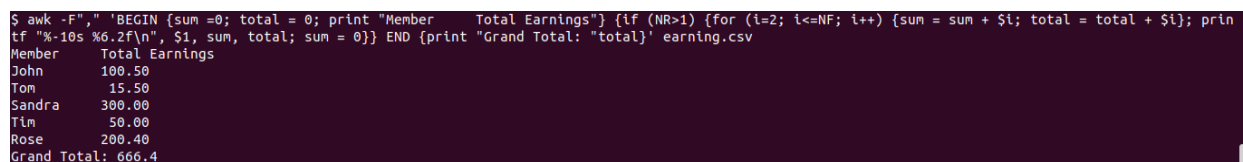
Awk is a scripting language used for manipulating data and generating reports. It has several built-in variables and has control structures, numeric and string functions.

Screenshots



```
$ cat earning.csv
Member,earning
John,10,20,30,40.5
Tom,1,2,3.5,4,5
Sandra,100,200
Tim,50
Rose,150.4,50
$
```

```
awk -F"," 'BEGIN {sum =0; total = 0; print "Member      Total
Earnings"} {if (NR>1) {for (i=2; i<=NF; i++) {sum = sum + $i;
total = total + $i}; printf "%-10s %6.2f\n", $1, sum, total; sum =
0}} END {print "Grand Total: "total}' earning.csv
```



```
$ awk -F"," 'BEGIN {sum =0; total = 0; print "Member      Total Earnings"} {if (NR>1) {for (i=2; i<=NF; i++) {sum = sum + $i; total = total + $i}; prin
tf "%-10s %6.2f\n", $1, sum, total; sum = 0}} END {print "Grand Total: "total}' earning.csv
Member      Total Earnings
John        100.50
Tom          15.50
Sandra      300.00
Tim          50.00
Rose        200.40
Grand Total: 666.4
```

Explanation

We have a csv file `earning.csv` which has information as member names and earnings made by them separated by commas.

We write an `awk` command to show total earnings by them and finally a grand total of all the earnings.

We use `BEGIN` section to initialize values and print header, process each line to calculate sum of earnings for each member and the intermediate grand total, and finally `END` section to print the grand total.

Example 3

Command

```
biff
```


Usage

```
biff [ny]
```

n = Disable mail notifications

y = Enable mail notifications

Screenshots

A terminal window with a dark background and light-colored text. It shows three lines of input: '\$ biff y', '\$ biff n', and '\$'.

```
$ biff y  
$ biff n  
$
```

Example 4

Command

cmp

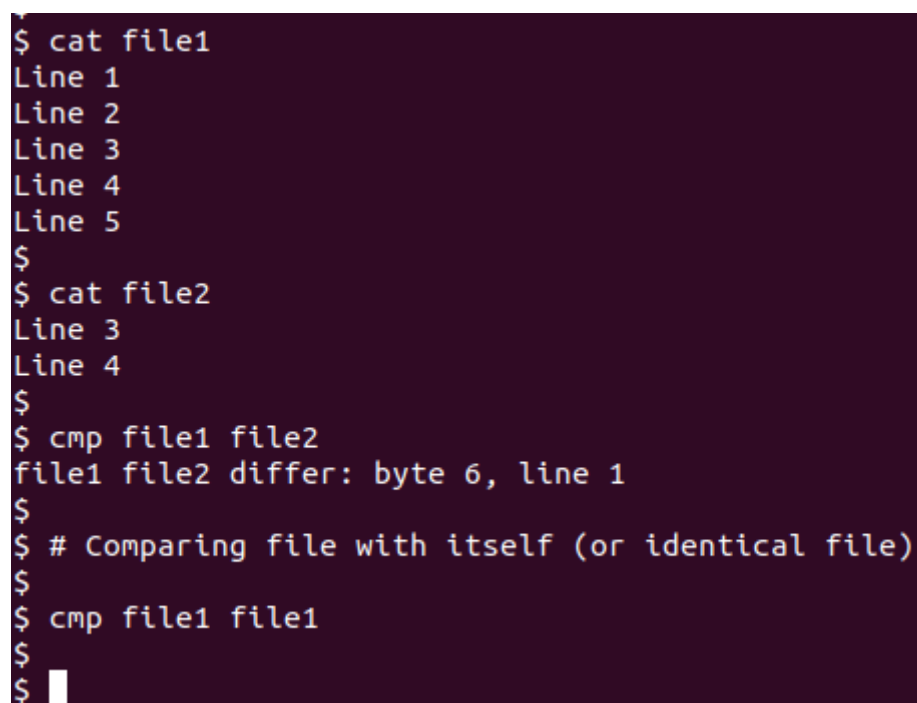
Usage

```
cmp [OPTION]... FILE1 [FILE2 [SKIP1 [SKIP2]]]
```

Compares two files byte by byte.

SKIP1, SKIP2: Number of bytes to skip at the beginning of each file. The default is 0.

Screenshots



```
$ cat file1
Line 1
Line 2
Line 3
Line 4
Line 5
$
$ cat file2
Line 3
Line 4
$
$ cmp file1 file2
file1 file2 differ: byte 6, line 1
$
$ # Comparing file with itself (or identical file)
$
$ cmp file1 file1
$
$
```

Example 5, 6

Command

compress
uncompress

Usage

compress [OPTION]... [FILE]...
uncompress [-cfv] [*file*...]

Screenshots

```
$ # A test file is created
$ cat myfile
1 one one one one one one one one one
2 two two two two two two two two two
3 three three three three three three three three three
4 four four four four four four four four four
5 five five five five five five five five five
$
$ # File stats
$ stat myfile
  File: myfile
  Size: 250          Blocks: 8          IO Block: 4096   regular file
Device: 802h/2050d  Inode: 1576064      Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/  ramesh)   Gid: ( 1000/  ramesh)
Access: 2021-02-21 02:40:05.175552885 +0530
Modify: 2021-02-21 02:39:29.687954578 +0530
Change: 2021-02-21 02:39:29.687954578 +0530
 Birth: 2021-02-21 02:39:29.687954578 +0530
$
$ # Compress the file
$ compress -v myfile
myfile:  -- replaced with myfile.Z Compression: 52.40%
$
$ stat myfile.Z
  File: myfile.Z
  Size: 119          Blocks: 8          IO Block: 4096   regular file
Device: 802h/2050d  Inode: 1575953      Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/  ramesh)   Gid: ( 1000/  ramesh)
Access: 2021-02-21 02:40:05.000000000 +0530
Modify: 2021-02-21 02:39:29.000000000 +0530
Change: 2021-02-21 02:42:07.210180889 +0530
 Birth: 2021-02-21 02:42:07.206180932 +0530
$
```

The file is 52.40% compressed and saved by adding .Z extension.

Uncompress

```
$ # The compressed file
$ ls
myfile.Z
$
$ # The contents of the compressed file
$ uncompress -c myfile.Z
1 one one one one one one one one one one
2 two two two two two two two two two two
3 three three three three three three three three three three
4 four four four four four four four four four four
5 five five five five five five five five five five
$
$ uncompress -v myfile.Z
myfile.Z:      52.4% -- replaced with myfile
$
$ # Verify the original file is created
$ ls
myfile
$
$ cat myfile
1 one one one one one one one one one one
2 two two two two two two two two two two
3 three three three three three three three three three three
4 four four four four four four four four four four
5 five five five five five five five five five five
$
```

Example 7

Command

`cpio`

copy files to and from archives. It has IN and OUT mode among other modes.

Usage

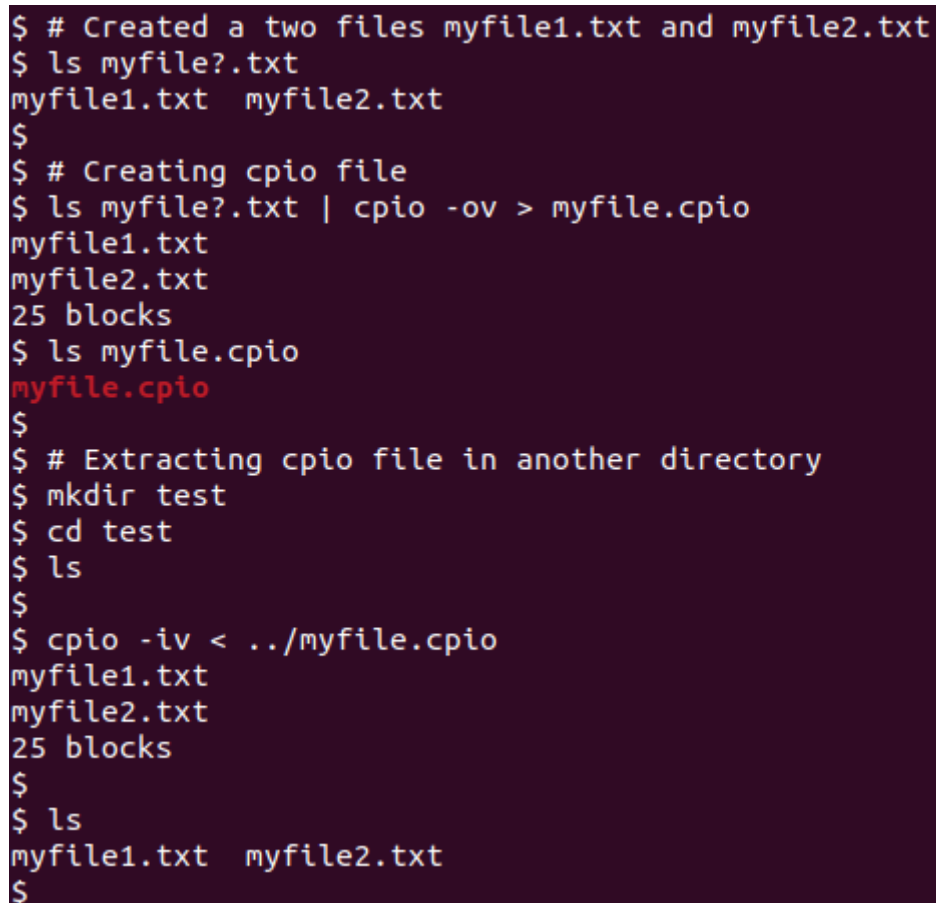
Copy-in mode extracts files from the archive:

```
cpio -i < archive
```

Copy-out mode copies files to archive:

```
cpio -o < filenames > archive
```

Screenshots



```
$ # Created a two files myfile1.txt and myfile2.txt
$ ls myfile?.txt
myfile1.txt  myfile2.txt
$
$ # Creating cpio file
$ ls myfile?.txt | cpio -ov > myfile.cpio
myfile1.txt
myfile2.txt
25 blocks
$ ls myfile.cpio
myfile.cpio
$
$ # Extracting cpio file in another directory
$ mkdir test
$ cd test
$ ls
$
$ cpio -iv < ../myfile.cpio
myfile1.txt
myfile2.txt
25 blocks
$
$ ls
myfile1.txt  myfile2.txt
$
```

Example 8, 9

Command

cron
crontab

cron runs scheduled jobs. cron daemon is responsible for running these jobs.

Usage

A cron entry is created using crontab command in the format:

m h dom mon dow command

where,

m = minute (0-59)

h = hour (0-23)

dom = day of month (1-31)

mon = month (1-12 or names)

dow = day of week (0-7 or names, 0 or 7 is Sunday)

Screenshots

Let's schedule a simple job that runs **echo** and redirects the output to a file myfile. We use * * * * * as time which means run the command every minute. We check the file every minute.

```
$ # Create a cron job
$
$ crontab -e
crontab: installing new crontab
$
$ # List a job
$
$ crontab -l
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
* * * * * echo "Hello CRON !!!" >> $HOME/test/myfile
$
```

List the contents of myfile after every minute

```
$ # Check the myfile to confirm cron is running every minute
$
$ cat myfile
Hello CRON !!!
$
$ # After a minute
$
$ cat myfile
Hello CRON !!!
Hello CRON !!!
$
$ # After a minute
$
$ cat myfile
Hello CRON !!!
Hello CRON !!!
Hello CRON !!!
$
$
$ # Delete the jobs
$
$ crontab -r
$
$ # After a minute
$
$ cat myfile
Hello CRON !!!
Hello CRON !!!
Hello CRON !!!
$
```

Example 10

Command

crypt

Encrypts a file.

Usage

Encrypt the file:

```
crypt key < file_to_be_encrypted > encrypted_file
```

Decrypt the file:

```
crypt key < encrypted_file > decrypted_file
```

Screenshots

```
$ # Create a file
$ cat > myfile
This file would be encrypted using crypt command.
The encrypted file would be named as myfile.crypt.
The key used would be 'secret'.
$ cat myfile
This file would be encrypted using crypt command.
The encrypted file would be named as myfile.crypt.
The key used would be 'secret'.
$
$ crypt secret < myfile > myfile.crypt
Unix crypt(1) emulation program using mcrypt(1).

Use crypt -h for more help.
Warning: It is insecure to specify keywords in the command line
Stdin was encrypted.
$
$ cat myfile
This file would be encrypted using crypt command.
The encrypted file would be named as myfile.crypt.
The key used would be 'secret'.
$ cat myfile.crypt
      lu4i-_H{m0k
          000g0c,a{0y[DCyjr0p0nC130Q
0
  2000'
$j"0
    0B
%"0G +7050$/ {ih0f"0000c0_ 0贈%eD0)00

zq$                                00000R0Hk00000
$
$ # Delete original file
$ rm myfile
$
```

```
$ ls myfile
ls: cannot access 'myfile': No such file or directory
$
$ # Decrypt the file
$
$ crypt secret < myfile.crypt > myfile
Unix crypt(1) emulation program using mrcrypt(1).

Use crypt -h for more help.
Warning: It is insecure to specify keywords in the command line
Stdin was encrypted.
$
$ cat myfile
This file would be encrypted using crypt command.
The encrypted file would be named as myfile.crypt.
The key used would be 'secret'.
$
$
$ crypt secret < myfile.crypt > myfile_copy
Unix crypt(1) emulation program using mrcrypt(1).

Use crypt -h for more help.
Warning: It is insecure to specify keywords in the command line
Stdin was encrypted.
$
$ cat myfile_copy
This file would be encrypted using crypt command.
The encrypted file would be named as myfile.crypt.
The key used would be 'secret'.
$
```


Example 11

Command

diff

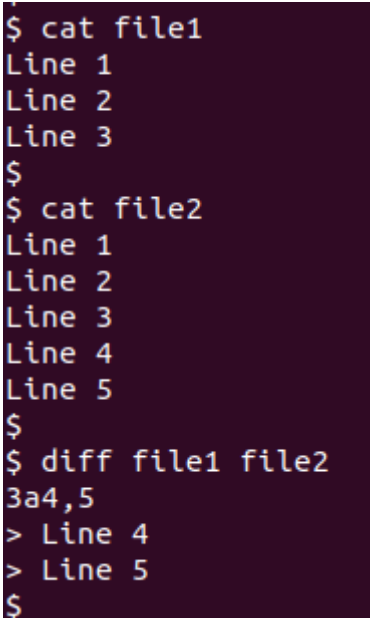
Compares two files line by line and lists the changes that should be made to the first file to make it look like the second one.

Usage

diff [OPTION]... FILES

Screenshots

> indicates lines from the second file to add to the first file



```
$ cat file1
Line 1
Line 2
Line 3
$
$ cat file2
Line 1
Line 2
Line 3
Line 4
Line 5
$
$ diff file1 file2
3a4,5
> Line 4
> Line 5
$
```

Both files are identical

```
$ cat file1
Line 1
Line 2
Line 3
Line 4
Line 5
$
$ cat file2
Line 1
Line 2
Line 3
Line 4
Line 5
$
$ diff file1 file2
$
```

< indicates lines from the first file to delete

```
$ cat file1
Line 1
Line 2
Line 3
Line 4
Line 5
$
$ cat file2
Line 1
Line 2
Line 3
$
$ diff file1 file2
4,5d3
< Line 4
< Line 5
$
```

Example 12

Command

dump

Usage

```
dump [OPTION] files_to_dump  
dump [ level ] [ f dumpFile ] [ v ] [ w ] {fileName}+
```

Takes backup of the entire file system. It also allows incremental backups.

Example

The following command performs a level 0 dump of a file system /dev/sda6 to a tape drive /dev/rmt0

```
dump 0 fv /dev/rmt0 /dev/sda6
```

Example 13

Command

egrep

Usage

egrep [options] 'PATTERN' files

Supports extended set of meta-characters in regular expressions.
It basically works same as **grep -E**

Screenshots

```
$ cat myfile
bright
right
others
$
$ # Normal grep can't handle regex metacharacter ? unless escaped or -E option is used
$
$ grep 'b?right' myfile
$
$ # But egrep can handle it. This avoids escaping.
$
$ egrep 'b?right' myfile
bright
right
$
```

Example 14

Command

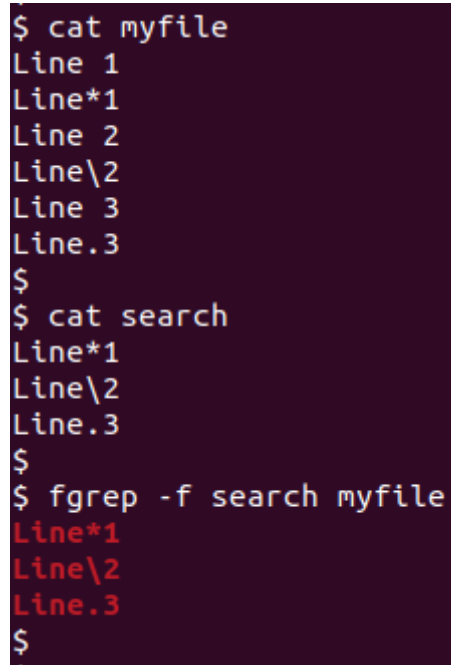
fgrep

Usage

```
fgrep [OPTIONS] PATTERN [FILE...]
```

It interprets PATTERN as a list of fixed strings instead of regular expressions, separated by newlines.

Screenshots



```
$ cat myfile
Line 1
Line*1
Line 2
Line\2
Line 3
Line.3
$
$ cat search
Line*1
Line\2
Line.3
$
$ fgrep -f search myfile
Line*1
Line\2
Line.3
$
```

Example 15

Command

find

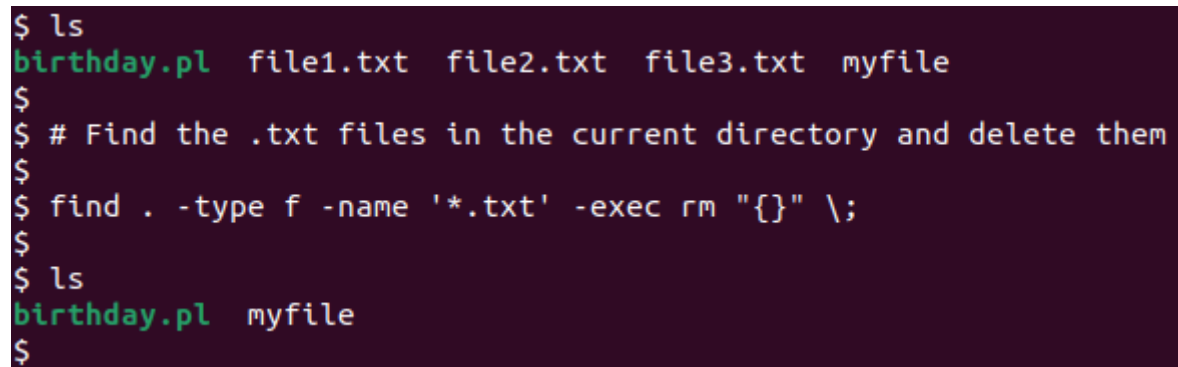
Usage

```
find [OPTION] [starting-point...] [expression]
```

Used to find files and directories and perform operations on them.

It's a very powerful command that can do lot of things like listing files based on the timestamps, recursive listing, etc.

Screenshots



```
$ ls
birthday.pl file1.txt file2.txt file3.txt myfile
$
$ # Find the .txt files in the current directory and delete them
$
$ find . -type f -name '*.txt' -exec rm "{}" \;
$
$ ls
birthday.pl myfile
$
```

Example 16

Command

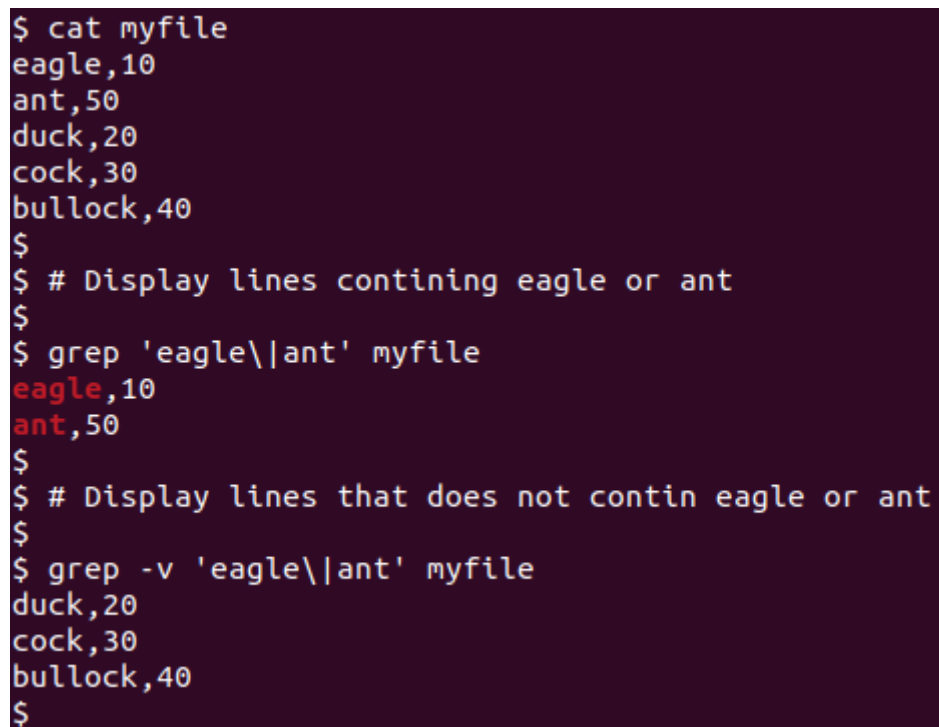
grep

Usage

```
grep [options] pattern [files]
```

It's used to search the files for pattern and display the matching lines.

Screenshots

A terminal window with a dark purple background and light green text. The terminal shows a series of commands and their outputs. First, 'cat myfile' is run, displaying a list of animals and their counts. Then, a comment is shown. Next, 'grep 'eagle\\|ant' myfile' is run, showing only the lines for 'eagle' and 'ant'. Finally, 'grep -v 'eagle\\|ant' myfile' is run, showing only the lines for 'duck', 'cock', and 'bullock'.

```
$ cat myfile
eagle,10
ant,50
duck,20
cock,30
bullock,40
$
$ # Display lines contining eagle or ant
$
$ grep 'eagle\\|ant' myfile
eagle,10
ant,50
$
$ # Display lines that does not contin eagle or ant
$
$ grep -v 'eagle\\|ant' myfile
duck,20
cock,30
bullock,40
$
```

Example 17, 18

Command

gzip
gunzip

Usage

```
gzip [Options] [filenames]  
gunzip [Option] [archive name/file name]
```

gzip compresses the files and creates an archive.
gunzip extracts files from gzip archive.

Screenshots

```
$ ls  
file1 file2 file3  
$  
$ cat file1  
This is file1  
$  
$ cat file2  
This is file2  
$  
$ cat file3  
This is file3  
$  
$ # Compress the files  
$ gzip file1 file2 file3  
$  
$ ls  
file1.gz file2.gz file3.gz  
$  
$ # Check the contents of the file without uncompressing it  
$ zcat file2.gz  
This is file2  
$  
$ # Extract file3.gz  
$ gunzip file3.gz  
$  
$ ls  
file1.gz file2.gz file3  
$  
$ cat file3  
This is file3  
$
```


Example 19

Command

ln

Usage

Three forms:

1. ln [OPTION]... [-T] TARGET LINK_NAME
2. ln [OPTION]... TARGET... DIRECTORY
3. ln [OPTION]... -t DIRECTORY TARGET...

It's used to create soft link or hard link (mirror copy) for the file.

Screenshots

```
$ ls
myfile
$
$ cat myfile
This file is for demonstrating links.
$
$ # Create a soft link
$ ln -s myfile myfile_softlink
$
$ # Create a hard link
$ ln myfile myfile_hardlink
$
$ ls
myfile  myfile_hardlink  myfile_softlink
$
$ cat myfile_softlink
This file is for demonstrating links.
$ cat myfile_hardlink
This file is for demonstrating links.
$
$ # Delete myfile. The soft link will break but the hard link will remain intact.
$ rm myfile
$
$ cat myfile_softlink
cat: myfile_softlink: No such file or directory
$
$ cat myfile_hardlink
This file is for demonstrating links.
$
$ ls
myfile_hardlink  myfile_softlink
$
```

Example 20

Command

od

Usage

od [OPTION]... [FILE]...

Converts the input file content into octal (default) or other formats depending upon the option.

Screenshots

```
$ cat myfile
eagle,10
ant,50
duck,20
cock,30
bullock,40
$
$ # Octal format
$
$ od myfile
00000000 060545 066147 026145 030061 060412 072156 032454 005060
0000020 072544 065543 031054 005060 067543 065543 031454 005060
0000040 072542 066154 061557 026153 030064 000012
0000053
$
$ # Character format
$
$ od -c myfile
00000000 e   a   g   l   e   ,   1   0   \n   a   n   t   ,   5   0   \n
0000020   d   u   c   k   ,   2   0   \n   c   o   c   k   ,   3   0   \n
0000040   b   u   l   l   o   c   k   ,   4   0   \n
0000053
$
```

Example 21

Command

perl

Usage

perl -e <perl code>

Or, we can use shebang in the script, `#!/usr/bin/env perl`

Perl is a cross-platform interpreted language especially useful for processing text, generating reports, and handling Common Gateway Interface (CGI) requests submitted via Web browsers.

Screenshots

Let's code and run a very basic perl program that takes an argument and greets them with a message.

```
$ cat birthday.pl
#!/usr/bin/env perl

$firstName=$ARGV[0];

print "Happy Birthday, $firstName !!! \n";

$
$ ./birthday.pl John
Happy Birthday, John !!!
$
$ ./birthday.pl Sandra
Happy Birthday, Sandra !!!
$
```

Example 22

Command

sed

Usage

```
sed OPTIONS... [SCRIPT] [INPUTFILE...]
```

It's a string editor that can perform actions on file, like, search/replace, deletion, insertion, etc.

Screenshots

```
$ cat myfile
eagle,10
ant,50
duck,20
cock,30
bullock,40
$
$ # Replace bullock and duck with ox
$
$ sed -E 's/bullock|duck/ox/' myfile
eagle,10
ant,50
ox,20
cock,30
ox,40
$
$ # Replace bullock and duck with ox, and then delete all the lines having ox in them
$
$ sed -E -e 's/bullock|duck/ox/' -e '/ox/d' myfile
eagle,10
ant,50
cock,30
$
```

Example 23

Command

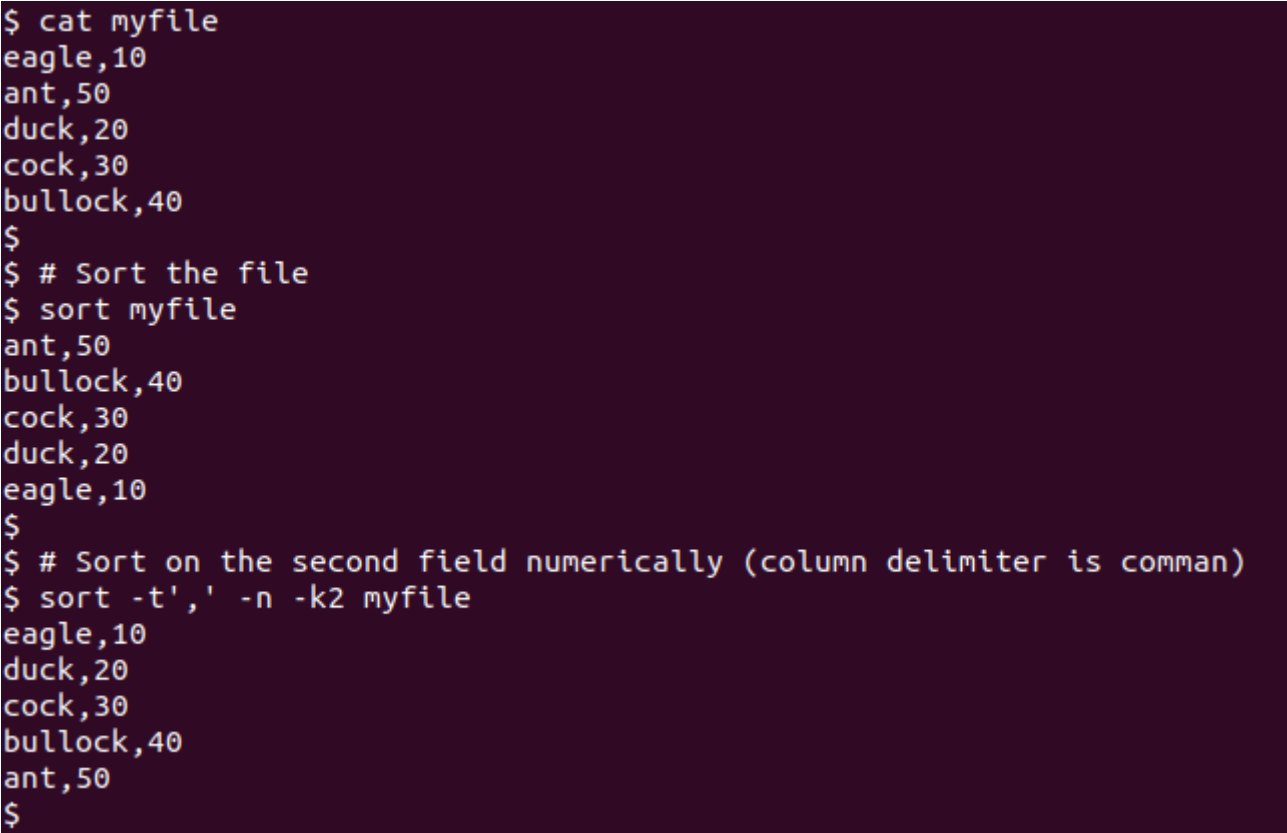
sort

Usage

sort [OPTION]... [FILE]...

Sorts the text files.

Screenshots

A terminal window with a dark purple background and light green text. It shows the execution of the 'cat' and 'sort' commands on a file named 'myfile'. The 'cat' command displays the contents of 'myfile', which are animal names followed by a comma and a number. The first 'sort' command sorts the file alphabetically by the animal names. The second 'sort' command, with options '-t',' -n -k2', sorts the file numerically by the numbers in the second field.

```
$ cat myfile
eagle,10
ant,50
duck,20
cock,30
bullock,40
$
$ # Sort the file
$ sort myfile
ant,50
bullock,40
cock,30
duck,20
eagle,10
$
$ # Sort on the second field numerically (column delimiter is comma)
$ sort -t',' -n -k2 myfile
eagle,10
duck,20
cock,30
bullock,40
ant,50
$
```

Example 24

Command

`su`

Usage

`su [options] [-] [user [argument...]]`

Allows to execute the commands as another user by switching to that user.

Examples

If the current user is abc, they can switch to the user xyz by:

`su xyz`

The system will ask for the password of xyz.

If option - is used, user's same login interface is provided and the directory is changed to the user's home directory.

`su - xyz`

Example 25

Command

tar

Usage

tar [options] [archive-file] [file or directory to be archived]

Creates and extracts archives.

Some of the main options are:

- c : Creates Archive
- x : Extract the archive
- f : creates archive with given filename
- t : displays or lists files in archived file
- u : archives and adds to an existing archive file
- v : Displays Verbose Information
- z : zip, tells tar command that create tar file using gzip

Screenshots

```
$ ls *.txt
file1.txt  file2.txt  file3.txt
$
$ # Create an archive
$
$ tar cvf text.tar *.txt
file1.txt
file2.txt
file3.txt
$
$ ls
file1.txt  file2.txt  file3.txt  text.tar
$
$ # Extract the files to test directory
$
$ mkdir test
$ ls test
$
$ tar xvf text.tar -C test
file1.txt
file2.txt
file3.txt
$
$ ls test
file1.txt  file2.txt  file3.txt
$
```

Example 26

Command

time

Usage

```
time [option] [COMMAND]
```

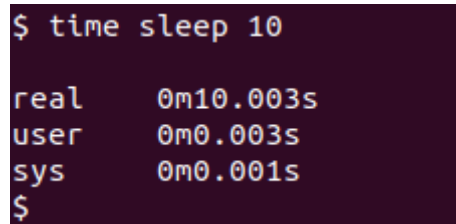
It executes a command and prints a summary of real-time, user CPU time and system CPU time spent the command.

'real' time is the wall clock time taken by a command.

'user' time is the number of CPU seconds that command used in user mode.

'sys' time is the number of CPU seconds that command used in kernel mode.

Screenshots



```
$ time sleep 10
real    0m10.003s
user    0m0.003s
sys     0m0.001s
$
```

A terminal window with a dark background showing the execution of the 'time sleep 10' command. The output displays three lines of timing information: 'real' time as 0m10.003s, 'user' time as 0m0.003s, and 'sys' time as 0m0.001s. The prompt '\$' is visible at the end of the output.

Example 27

Command

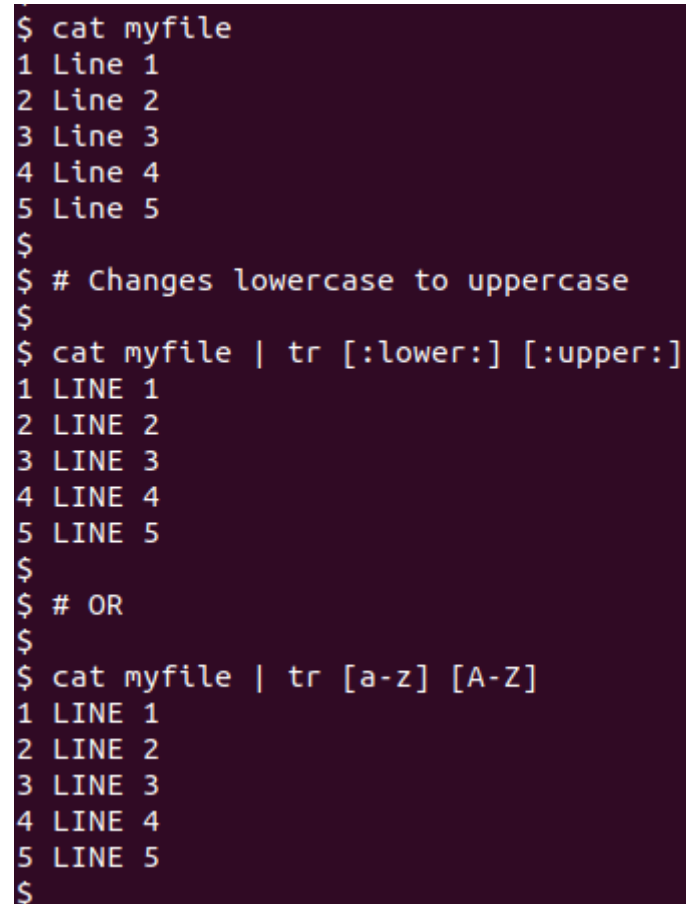
tr

Usage

tr [OPTION]... SET1 [SET2]

It translates or deletes characters.

Screenshot



```
$ cat myfile
1 line 1
2 line 2
3 line 3
4 line 4
5 line 5
$
$ # Changes lowercase to uppercase
$
$ cat myfile | tr [:lower:] [:upper:]
1 LINE 1
2 LINE 2
3 LINE 3
4 LINE 4
5 LINE 5
$
$ # OR
$
$ cat myfile | tr [a-z] [A-Z]
1 LINE 1
2 LINE 2
3 LINE 3
4 LINE 4
5 LINE 5
$
```

Explanation

We have translated each line from lowercase to upper case in two ways.

Now we replace space with tab.

Screenshot

```
$ cat myfile
1 Line 1
2 Line 2
3 Line 3
4 Line 4
5 Line 5
$
$ cat myfile | tr ' ' $'\t'
1      Line      1
2      Line      2
3      Line      3
4      Line      4
5      Line      5
$
```

Explanation

We use space in set1 and tab (\t) in set 2.

Now we delete the spaces in each line.

Screenshot

```
$ cat myfile
1 Line 1
2 Line 2
3 Line 3
4 Line 4
5 Line 5
$
$ cat myfile | tr -d ' '
1Line1
2Line2
3Line3
4Line4
5Line5
$
```

Explanation

We have used -d option to delete the spaces.

Example 28

Command

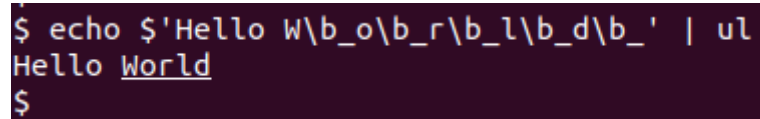
`ul`

Usage

`ul [options] [file...]`

It reads the files (or standard input) and translates occurrences of underscores to the sequence which indicates underlining for the terminal in use (as specified by the environment variable `TERM`).

Screenshots



```
$ echo $'Hello w\b_o\b_r\b_l\b_d\b_' | ul
Hello World
$
```

Example 29, 30

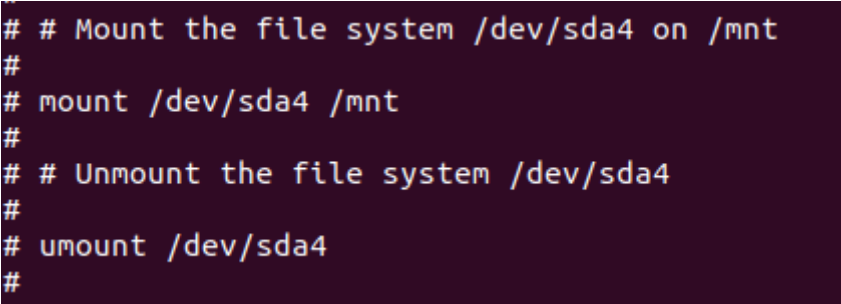
Command

mount
umount

Usage

mount file_system mount_point
mount file_system

Screenshots



```
# # Mount the file system /dev/sda4 on /mnt  
#  
# mount /dev/sda4 /mnt  
#  
# # Unmount the file system /dev/sda4  
#  
# umount /dev/sda4  
#
```

Example 31

Command

```
uniq
```

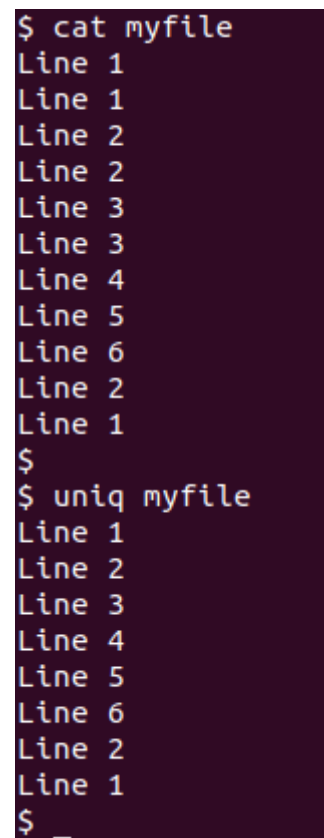
Usage

```
uniq filename
```

It filters out repeated lines in the file.

Note: It only filters out 'adjacent' duplicate lines.

Screenshots



```
$ cat myfile
Line 1
Line 1
Line 2
Line 2
Line 3
Line 3
Line 4
Line 5
Line 6
Line 2
Line 1
$
$ uniq myfile
Line 1
Line 2
Line 3
Line 4
Line 5
Line 6
Line 2
Line 1
$
```

Note: The last two lines aren't filtered out.

Example 32

Command

```
whoami
```

Usage

```
whoami
```

It prints the user name of the effective user ID (currently logged-in user).

For example, if the user name is 'john', it displays 'john'.

Chapter 4

Example 1, 2, 3

Commands

ps, sleep, kill

Usage

ps [options]

It reports the information about the currently running (active) processes.

sleep NUMBER[SUFFIX]

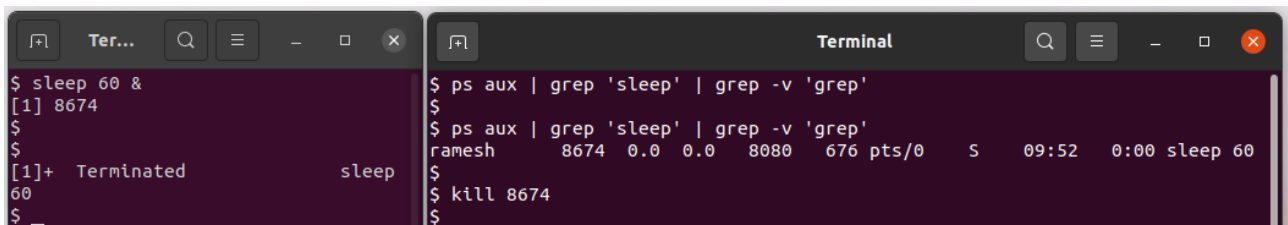
NUMBER is any number, and SUFFIX can be s (seconds), m (minutes), h (hours), d (days). The default is s.

It creates a delay for the specified amount of time.

kill [options] <pid> [...]

Sends a signal to a process. Default signal is TERM which means terminate the process.

Screenshot



Explanation

In the first terminal window we give a **sleep** command (in background) to create a delay of 60 seconds.

In the second window we test the output of **ps** command before **sleep** was issued. And, then after the **sleep** was issued, which shows the pid and other information of the process.

We then issue **kill** command to terminate the process and again check the output of **ps** command.

Example 4

Command

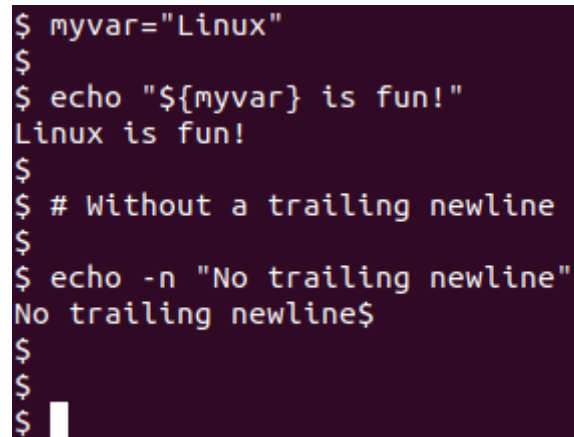
echo

Displays the text and variables given to it as arguments.

Usage

echo [OPTION]... [STRING]...

Screenshot

A terminal window with a dark purple background and light green text. It shows a series of commands and their outputs. The commands are: 'myvar="Linux"', 'echo "\${myvar} is fun!"', '# Without a trailing newline', and 'echo -n "No trailing newline"'. The outputs are: 'Linux is fun!', 'No trailing newline', and a blank line. The prompt character '\$' is visible at the start of each line.

```
$ myvar="Linux"
$
$ echo "${myvar} is fun!"
Linux is fun!
$
$ # Without a trailing newline
$
$ echo -n "No trailing newline"
No trailing newline$
$
$
$
```

Explanation

In the first example **echo** displays a value of the variable along with other text.

In the second example it uses **-n** option so trailing newline is not outputted.

Example 5

Command

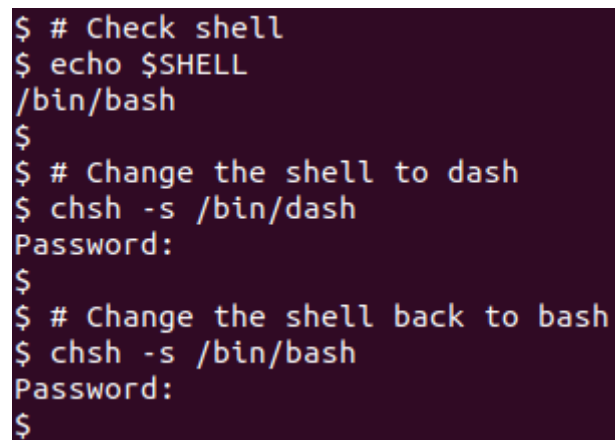
chsh

Usage

chsh [OPTIONS] [LOGIN]

Used to change user's current login shell.

Screenshots



```
$ # Check shell
$ echo $SHELL
/bin/bash
$
$ # Change the shell to dash
$ chsh -s /bin/dash
Password:
$
$ # Change the shell back to bash
$ chsh -s /bin/bash
Password:
$
```

The change is reflected in /etc/passwd

Example 6

Command

nohup

Usage

nohup command [command-argument ...]

It is used to keep the job or command running on the server even when you log out or lost the connection to the server.

Chapter 5

Example 1

Commands

`expr`

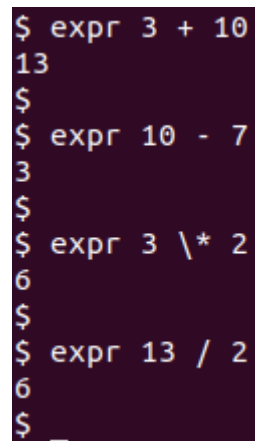
Usage

```
expr EXPRESSION  
expr OPTION
```

Evaluates expressions.

Screenshot

Let's do basic arithmetic using **expr**

A terminal window with a dark background and light-colored text. It shows a series of commands and their outputs. The commands are: 'expr 3 + 10', 'expr 10 - 7', 'expr 3 * 2', and 'expr 13 / 2'. The outputs are: '13', '3', '6', and '6' respectively. The prompt '\$' is visible at the start of each line.

```
$ expr 3 + 10  
13  
$  
$ expr 10 - 7  
3  
$  
$ expr 3 \* 2  
6  
$  
$ expr 13 / 2  
6  
$
```

Example 2

Commands

test

Usage

test EXPRESSION - Used to test file types, compare strings and integers.

Screenshot

Let's check if a file or directory exists using -f and -d option.

```
$ # File 'myfile' does not exist
$
$ ls myfile
ls: cannot access 'myfile': No such file or directory
$
$ # Directory 'mydir' does not exist
$
$ ls mydir
ls: cannot access 'mydir': No such file or directory
$
$ test -f myfile
$
$ echo "$?"
1
$
$ # Create a file
$
$ touch myfile
$
$ ls myfile
myfile
$
$ # Test again
$
$ test -f myfile
$
$ echo "$?"
0
$
$ # Create directory and test again
$ mkdir mydir
$ test -d mydir
$ echo "$?"
0
$
```

Explanation

Here we check if a file or directory exists. When it doesn't exist the return code (\$?) of **test** is 1 otherwise 0.

Now we test two strings.

Screenshot

```
$ # Test directly
$
$ test Linux = Linux
$
$ echo "$?"
0
$
$ test Linux = Unix
$
$ echo "$?"
1
$
$ # Test the variables
$
$ var1="Linux"
$ var2="Unix"
$ var3="Linux"
$
$ test "$var1" = "$var2"
$
$ echo "$?"
1
$
$ test "$var1" = "$var3"
$
$ echo "$?"
0
$
```

Explanation

Here we **test** the strings directly and then the variables. The return code (exit status) we get is 0 for success and 1 for failure.

Now, let's test integers.

Screenshot

```
$ # Compare integers directly
$
$ test 21 -eq 21
$
$ echo $?
0
$
$ test 21 -eq 33
$
$ echo $?
1
$
$ # Compare variables
$
$ var1=1
$ var2=2
$ var3=1
$
$ test "$var1" -eq "$var2"
$
$ echo $?
1
$
$ test "$var1" -eq "$var3"
$
$ echo $?
0
$
```

Explanation

Again, we compare the integers directly and the variables that have integer values stored in them.

The exit status is again 0 for success and 1 for failure.