

*(The following page contains faint, mostly illegible handwritten notes or bleed-through from another document.)*

# terminal

technically called a terminal emulator, it is a program that displays output on the screen and accepts input from the keyboard. It is used to interact with the operating system.



# shell

A command line interface, this program interprets what you typed as a command in the terminal, figures out what you want to run, runs it and sends the output back to the terminal.



# shell

In Unix there are two major types of shells:

1. The **Bourne Shell** (The default prompt is the \$ character)
2. The **C shell** (The default prompt is the % character). tcsh is the superset of csh, enhancing user friendliness and speed.



## other shells

- ❏ Bourne Shell (**sh**)
- ❏ Korn Shell (**ksh**)
- ❏ Bourne Again Shell (**bash**)
- ❏ Z shell (**zsh**)

# bash

bash is an sh-compatible shell that incorporates useful features from the Korn shell (ksh) and C shell (csh). It offers functional improvements over sh for both programming and interactive use.

```
#!/bin/bash
# Example script: greet.sh
# Usage: ./greet.sh <name>

# Check if a name was provided
if [ $# -eq 0 ]; then
    echo "Usage: ./greet.sh <name>"
    exit 1
fi

# Greet the user
echo "Hello, $1!"

# Add a newline
echo
```

# some basic commands in linux

**ls**- the list command used for showing all the major directories filled under a given file system

**cd dir**- allows the user to change between file directories

**pwd**- prints the name of the working directory

**man**- opens manual for commands

**man command**- shows the information about the inputted command

**touch file**- create or update file



# some basic commands in linux

**cat**- concatenate files and print on the standard output

**ps**- display current active processes

**top**- displays all running processes

**more file**- outputs contents of a file

**less file**- outputs contents of a file

**chmod**- changes permission of file

**sudo**- enables us to become super user





[illegible]

*(The following page contains faint, illegible markings, possibly bleed-through from the reverse side.)*

# command line utilities

## grep - *print lines matching a pattern*

## **sed** - *stream editor for filtering and transforming text*

## awk - pattern scanning and processing language

## cut - remove sections from each line of files

## sort - sort lines of text files

## head - output the first part of files

## **tail** - *output the last part of files*



# piping and redirection

## data streams

stdin

stdout

stderr

```
#!/usr/bin/perl
# This script demonstrates the use of pipes and redirection.
# It reads input from stdin, processes it, and writes the output to stdout.
# Error messages are written to stderr.

use strict;
use warnings;

# Read input from stdin
my $input = <<STDIN;

# Process the input
my $output = $input;

# Write the output to stdout
print $output;

# Write error messages to stderr
print STDERR "Error: Invalid input\n";
```

# piping and redirection

# pipe operator |

passes **stdout** of **command1** to **stdin** of **command2**.

## usage:

```
$ command1 | command2
```

[illegible]

# piping and redirection

## redirection operator >

writes stdout of command to file, **overwriting** it.

### usage:

```
$ command > file
```

```
#!/usr/bin/perl
# This script demonstrates the use of the > redirection operator.
# It takes a command and a filename as input and runs the command,
# redirecting its output to the file.
# Usage: perl script.pl command filename
# Example: perl script.pl cat file.txt
# The output of the command 'cat file.txt' will be written to a new
# file named 'file.txt'.

my $command = $ARGV[0];
my $filename = $ARGV[1];

system("$command > $filename");
```

# piping and redirection

## redirection operator >>

writes **stdout** of command to file, **appending** it.

## usage:

```
$ command >> file
```

[illegible]

# piping and redirection

## redirection operator <

**reads** contents of file into stdin of command.

## usage:

\$ command < file

```
#!/bin/bash
# This script demonstrates the use of the < operator to read input from a file.
# It reads the contents of the file 'input.txt' and prints them to the standard output.

# Read the contents of the file 'input.txt' and print them to the standard output.
cat < input.txt
```



# grep

prints matches of **PATTERN** in **file**.

## usage:

```
$ command | grep PATTERN
```

```
$ grep PATTERN < file
```



# regular expressions

Regular expressions are a powerful tool for searching and manipulating text. They are used in many applications, including text editors, web browsers, and data analysis tools. Regular expressions are a sequence of characters that define a search pattern. They can be used to find a specific string of text, or to find all occurrences of a certain pattern. Regular expressions are also used to replace text, to extract data from a string, and to validate input. Regular expressions are a complex topic, but they are essential for many tasks in computer science and data processing. This document provides a comprehensive overview of regular expressions, including their syntax, semantics, and applications. It also includes examples of how to use regular expressions in various programming languages and tools. Regular expressions are a powerful tool for searching and manipulating text. They are used in many applications, including text editors, web browsers, and data analysis tools. Regular expressions are a sequence of characters that define a search pattern. They can be used to find a specific string of text, or to find all occurrences of a certain pattern. Regular expressions are also used to replace text, to extract data from a string, and to validate input. Regular expressions are a complex topic, but they are essential for many tasks in computer science and data processing. This document provides a comprehensive overview of regular expressions, including their syntax, semantics, and applications. It also includes examples of how to use regular expressions in various programming languages and tools.

# literals

/re/g

Regular expressions are fun!

/e./g

regular expressions are fun!

**use the dot sparingly.** more often than not, one can easily replace the dot with something more precise.

```

    <!-- The first line of the document is the root element -->
    <html>
      <!-- The head of the document -->
      <head>
        <!-- The title of the document -->
        <title>The first line of the document is the root element</title>
        <!-- The meta-information of the document -->
        <meta charset="utf-8" />
        <!-- The style information of the document -->
        <link href="style.css" rel="stylesheet" />
        <!-- The script information of the document -->
        <script src="script.js" />
      </head>
      <!-- The body of the document -->
      <body>
        <!-- The first line of the body -->
        <h1>The first line of the body</h1>
        <!-- The second line of the body -->
        <h2>The second line of the body</h2>
        <!-- The third line of the body -->
        <h3>The third line of the body</h3>
        <!-- The fourth line of the body -->
        <h4>The fourth line of the body</h4>
        <!-- The fifth line of the body -->
        <h5>The fifth line of the body</h5>
        <!-- The sixth line of the body -->
        <h6>The sixth line of the body</h6>
        <!-- The seventh line of the body -->
        <h7>The seventh line of the body</h7>
        <!-- The eighth line of the body -->
        <h8>The eighth line of the body</h8>
        <!-- The ninth line of the body -->
        <h9>The ninth line of the body</h9>
        <!-- The tenth line of the body -->
        <h10>The tenth line of the body</h10>
      </body>
    </html>
  
```

# metacharacters

special characters in regex

if required as literals, these must be used with an escape character \

\ ^ \$ . | ? \* + ( ) [ {



# character classes

`/[Rr]e/g`

Regular expressions are fun!

Hyphens specify a range of characters.

`/e[a-zA-Z0-9]/g`

Regular expressions are fun!

^ immediately after a [ negates everything within that class.

`/e[^a-zA-Z0-9]/g`

Regular expressions are fun!



## shorthand character classes

**\d** - digit character

**\w** - word character (alphanumerics and underscores)

**\s** - whitespace character

...etc.

[illegible]

# repetition

- ? - makes the preceding token optional
- \* - matches the preceding token 0 or more times
- + - matches the preceding token 1 or more times
- {5} - matches the preceding token exactly 5 times

[illegible]

# greediness

note: repetitions, are, by default, *greedy*.

this means that it will try to match the longest possible string.

for example, to match any html tag, one might write:

<.+>

However, this might result in problems:

# Header

# The whole line is a match!

[illegible]



# alternation

## usage:

**/I like (cat|dog)s./g**

I like cats.

I like dogs.

I like broccoli. (nobody likes broccoli)

[illegible]

# anchors

used to, well, *anchor* the match at a certain position.

these match *positions* rather than *characters*.

for example, `^` matches the character (so to say) *just before* the start of the string.

\$ matches the character *just after*.



# why are anchors useful?

what if we wanted to check if the *entire* string was a number?

`/[0-9]+/` or `/\d+/`

9324802349, which is great, but-

ab192ab12938191, this fails in cases like this.

`/^\d+$/`

9324802349

ab192ab12938191, it works!



# modifiers

**/g** - global; finds all matches rather than just the first one

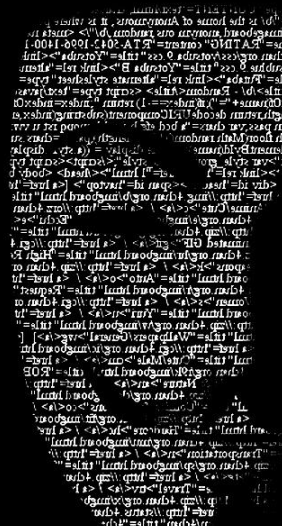
**/m** - multiline; treats every line as a separate string

**/i** - ignore case; disables case sensitivity

## usage:

**/abc/igm**

abcdefABCDEF



# grouping

parentheses ( )

enables use of a **quantifier** on a group of characters.

**/piano(forte)?/**

restricts **alternation**.

**/seventy-(two|three|four)/**



## resources

<https://regexr.com/>

<http://www.regular-expressions.info/>

[illegible]

# cron

```
#!/bin/bash
# Cron job script
# This script runs every day at 10:00 AM
# It checks the status of the service and restarts it if it is stopped

# Set the service name
SERVICE_NAME="my-service"

# Check the status of the service
if systemctl is-active --quiet $SERVICE_NAME; then
    echo "$SERVICE_NAME is running."
else
    echo "$SERVICE_NAME is not running. Restarting..."
    systemctl restart $SERVICE_NAME
fi
```

# cron

used to periodically run jobs (scripts)

automate sysadmin work

download emails, run your backup, clear the trash etc. (anything repetitive)

depends on a crontab (cron table) file

it is a configuration file that specifies the shell commands to be run

```
#!/bin/bash
# Cron job to backup the database
# Run every day at 2 AM
0 2 * * * /usr/bin/backup_db.sh
# Run every Monday at 10 AM
0 10 * * 1 /usr/bin/backup_db.sh
# Run every Friday at 3 PM
0 15 * * 5 /usr/bin/backup_db.sh
# Run every Sunday at 12 PM
0 12 * * 0 /usr/bin/backup_db.sh
# Run every 15 minutes
*/15 * * * * /usr/bin/backup_db.sh
# Run every hour
0 * * * * /usr/bin/backup_db.sh
# Run every day at 1 AM
0 1 * * * /usr/bin/backup_db.sh
# Run every day at 11 PM
0 23 * * * /usr/bin/backup_db.sh
# Run every day at 10 PM
0 22 * * * /usr/bin/backup_db.sh
# Run every day at 9 PM
0 21 * * * /usr/bin/backup_db.sh
# Run every day at 8 PM
0 20 * * * /usr/bin/backup_db.sh
# Run every day at 7 PM
0 19 * * * /usr/bin/backup_db.sh
# Run every day at 6 PM
0 18 * * * /usr/bin/backup_db.sh
# Run every day at 5 PM
0 17 * * * /usr/bin/backup_db.sh
# Run every day at 4 PM
0 16 * * * /usr/bin/backup_db.sh
# Run every day at 3 PM
0 15 * * * /usr/bin/backup_db.sh
# Run every day at 2 PM
0 14 * * * /usr/bin/backup_db.sh
# Run every day at 1 PM
0 13 * * * /usr/bin/backup_db.sh
# Run every day at 12 PM
0 12 * * * /usr/bin/backup_db.sh
# Run every day at 11 AM
0 11 * * * /usr/bin/backup_db.sh
# Run every day at 10 AM
0 10 * * * /usr/bin/backup_db.sh
# Run every day at 9 AM
0 9 * * * /usr/bin/backup_db.sh
# Run every day at 8 AM
0 8 * * * /usr/bin/backup_db.sh
# Run every day at 7 AM
0 7 * * * /usr/bin/backup_db.sh
# Run every day at 6 AM
0 6 * * * /usr/bin/backup_db.sh
# Run every day at 5 AM
0 5 * * * /usr/bin/backup_db.sh
# Run every day at 4 AM
0 4 * * * /usr/bin/backup_db.sh
# Run every day at 3 AM
0 3 * * * /usr/bin/backup_db.sh
# Run every day at 2 AM
0 2 * * * /usr/bin/backup_db.sh
# Run every day at 1 AM
0 1 * * * /usr/bin/backup_db.sh
# Run every day at 12 AM
0 0 * * * /usr/bin/backup_db.sh
# Run every day at 11 PM
0 23 * * * /usr/bin/backup_db.sh
# Run every day at 10 PM
0 22 * * * /usr/bin/backup_db.sh
# Run every day at 9 PM
0 21 * * * /usr/bin/backup_db.sh
# Run every day at 8 PM
0 20 * * * /usr/bin/backup_db.sh
# Run every day at 7 PM
0 19 * * * /usr/bin/backup_db.sh
# Run every day at 6 PM
0 18 * * * /usr/bin/backup_db.sh
# Run every day at 5 PM
0 17 * * * /usr/bin/backup_db.sh
# Run every day at 4 PM
0 16 * * * /usr/bin/backup_db.sh
# Run every day at 3 PM
0 15 * * * /usr/bin/backup_db.sh
# Run every day at 2 PM
0 14 * * * /usr/bin/backup_db.sh
# Run every day at 1 PM
0 13 * * * /usr/bin/backup_db.sh
# Run every day at 12 PM
0 12 * * * /usr/bin/backup_db.sh
# Run every day at 11 AM
0 11 * * * /usr/bin/backup_db.sh
# Run every day at 10 AM
0 10 * * * /usr/bin/backup_db.sh
# Run every day at 9 AM
0 9 * * * /usr/bin/backup_db.sh
# Run every day at 8 AM
0 8 * * * /usr/bin/backup_db.sh
# Run every day at 7 AM
0 7 * * * /usr/bin/backup_db.sh
# Run every day at 6 AM
0 6 * * * /usr/bin/backup_db.sh
# Run every day at 5 AM
0 5 * * * /usr/bin/backup_db.sh
# Run every day at 4 AM
0 4 * * * /usr/bin/backup_db.sh
# Run every day at 3 AM
0 3 * * * /usr/bin/backup_db.sh
# Run every day at 2 AM
0 2 * * * /usr/bin/backup_db.sh
# Run every day at 1 AM
0 1 * * * /usr/bin/backup_db.sh
# Run every day at 12 AM
0 0 * * * /usr/bin/backup_db.sh
```



# crontab

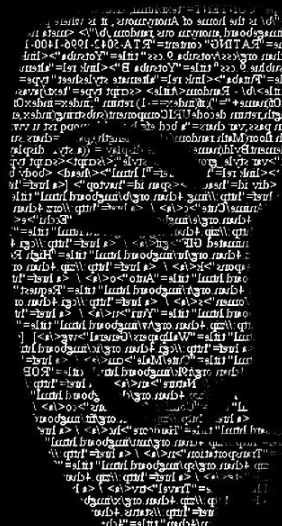
```
$ crontab
```

## flags:

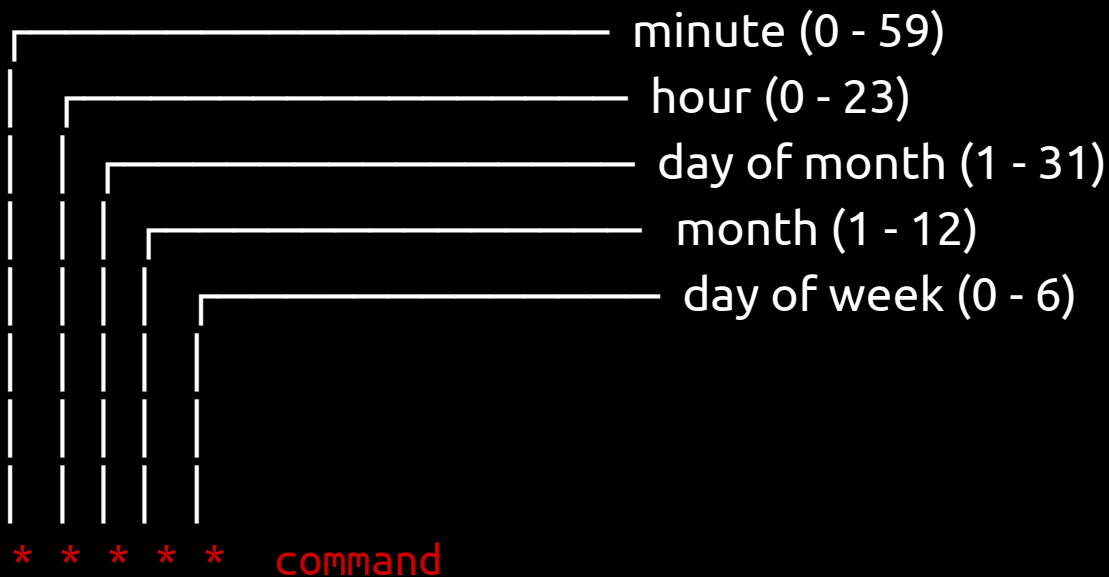
-l, list user's crontab

-e, edit user's crontab

-r, delete user's crontab



# crontab



```
crontab -l
# Example crontab content
# m h dom mon dow command
# * * * * * command
```

# cron

# directories

```
/etc/cron.d
```

```
/etc/cron.daily
```

```
/etc/cron.weekly
```

```
/etc/cron.monthly
```

# files

```
/etc/crontab
```

```
/etc/cron.allow
```

```
/etc/cron.deny
```

[illegible]

# macros

string	equivalent to
@reboot	None
@monthly	0 0 1 * *
@weekly	0 0 * * 0
@daily	0 0 * * *
@hourly	0 * * * *

# what can you do with cron?

take out the trash

download a podcast

periodically mail yourself

check server downtime

change your background automatically

```
#!/bin/bash
# Cron job to take out the trash
rm -rf ~/.cache/*

# Cron job to download a podcast
curl -L -o ~/podcast.mp3 http://example.com/podcast.mp3

# Cron job to periodically mail yourself
cat /dev/urandom | tr -dc 'a-z0-9' | fold -w 64 | xargs -n 1 sh -c 'echo $0' | mail -s "Random string" root

# Cron job to check server downtime
ping -c 1 google.com && echo "Server is up" || echo "Server is down" | mail -s "Server status" root

# Cron job to change your background automatically
curl -L -o ~/background.jpg http://example.com/background.jpg
cp ~/background.jpg ~/.background
```

## resources

<https://help.ubuntu.com/community/CronHowto>

\$ man 8 cron

```
$ man 5 crontab
```

[illegible]

# ssh

secure **shell**

[illegible]

# ssh

network services over an unsecure network

all data transfer via ssh is encrypted

most commonly used for logging into a remote system

any network operation/service can be secured with ssh





# ssh

## ssh - remote login

## sftp - *secure file transfer over ssh*

## scp - *secure copy (remote file copy program)*



# expect

# scripting language

automates interactions between two programs

used here to automate sshing into a remote system, getting some log data and exiting



# ssh demo using expect

```
#!/bin/bash
# ssh demo using expect
# This script demonstrates how to use expect to automate SSH connections.
# It connects to a remote host and runs a series of commands.

HOST="192.168.1.100"
USER="root"
PASS="password"

# Function to run a command on the remote host
run_cmd() {
    expect -c "
        spawn ssh -i /path/to/key $USER@$HOST
        expect {
            'password:' {
                send "$PASS\r"
            }
            eof {
                exit 1
            }
        }
        expect {
            '#' {
                send "$1\r"
            }
            eof {
                exit 1
            }
        }
    "
}

# Run a series of commands
run_cmd "ls -la"
run_cmd "cat /etc/passwd"
run_cmd "exit"

# End of script
```

# resources

<https://www.ssh.com/ssh/>

<https://help.ubuntu.com/community/SSH>

```

1  #!/bin/bash
2  # This script is used to install the SSH server on a Ubuntu system.
3  # It will install the SSH server and configure it to allow password authentication.
4  # It will also create a new user and add it to the sudo group.
5  # It will then restart the SSH service and test the connection.
6  # It will finally print a message to the terminal.
7  # It will exit with a status code of 0 if successful, and 1 if not.
8  # It will also print the version of the script.
9  # It will also print the date and time of execution.
10 # It will also print the user who executed the script.
11 # It will also print the IP address of the machine.
12 # It will also print the hostname of the machine.
13 # It will also print the architecture of the machine.
14 # It will also print the kernel version of the machine.
15 # It will also print the release version of the machine.
16 # It will also print the codename of the machine.
17 # It will also print the flavor of the machine.
18 # It will also print the variant of the machine.
19 # It will also print the variant-bzr of the machine.
20 # It will also print the variant-cmd of the machine.
21 # It will also print the variant-cmd of the machine.
22 # It will also print the variant-cmd of the machine.
23 # It will also print the variant-cmd of the machine.
24 # It will also print the variant-cmd of the machine.
25 # It will also print the variant-cmd of the machine.
26 # It will also print the variant-cmd of the machine.
27 # It will also print the variant-cmd of the machine.
28 # It will also print the variant-cmd of the machine.
29 # It will also print the variant-cmd of the machine.
30 # It will also print the variant-cmd of the machine.
31 # It will also print the variant-cmd of the machine.
32 # It will also print the variant-cmd of the machine.
33 # It will also print the variant-cmd of the machine.
34 # It will also print the variant-cmd of the machine.
35 # It will also print the variant-cmd of the machine.
36 # It will also print the variant-cmd of the machine.
37 # It will also print the variant-cmd of the machine.
38 # It will also print the variant-cmd of the machine.
39 # It will also print the variant-cmd of the machine.
40 # It will also print the variant-cmd of the machine.
41 # It will also print the variant-cmd of the machine.
42 # It will also print the variant-cmd of the machine.
43 # It will also print the variant-cmd of the machine.
44 # It will also print the variant-cmd of the machine.
45 # It will also print the variant-cmd of the machine.
46 # It will also print the variant-cmd of the machine.
47 # It will also print the variant-cmd of the machine.
48 # It will also print the variant-cmd of the machine.
49 # It will also print the variant-cmd of the machine.
50 # It will also print the variant-cmd of the machine.
51 # It will also print the variant-cmd of the machine.
52 # It will also print the variant-cmd of the machine.
53 # It will also print the variant-cmd of the machine.
54 # It will also print the variant-cmd of the machine.
55 # It will also print the variant-cmd of the machine.
56 # It will also print the variant-cmd of the machine.
57 # It will also print the variant-cmd of the machine.
58 # It will also print the variant-cmd of the machine.
59 # It will also print the variant-cmd of the machine.
60 # It will also print the variant-cmd of the machine.
61 # It will also print the variant-cmd of the machine.
62 # It will also print the variant-cmd of the machine.
63 # It will also print the variant-cmd of the machine.
64 # It will also print the variant-cmd of the machine.
65 # It will also print the variant-cmd of the machine.
66 # It will also print the variant-cmd of the machine.
67 # It will also print the variant-cmd of the machine.
68 # It will also print the variant-cmd of the machine.
69 # It will also print the variant-cmd of the machine.
70 # It will also print the variant-cmd of the machine.
71 # It will also print the variant-cmd of the machine.
72 # It will also print the variant-cmd of the machine.
73 # It will also print the variant-cmd of the machine.
74 # It will also print the variant-cmd of the machine.
75 # It will also print the variant-cmd of the machine.
76 # It will also print the variant-cmd of the machine.
77 # It will also print the variant-cmd of the machine.
78 # It will also print the variant-cmd of the machine.
79 # It will also print the variant-cmd of the machine.
80 # It will also print the variant-cmd of the machine.
81 # It will also print the variant-cmd of the machine.
82 # It will also print the variant-cmd of the machine.
83 # It will also print the variant-cmd of the machine.
84 # It will also print the variant-cmd of the machine.
85 # It will also print the variant-cmd of the machine.
86 # It will also print the variant-cmd of the machine.
87 # It will also print the variant-cmd of the machine.
88 # It will also print the variant-cmd of the machine.
89 # It will also print the variant-cmd of the machine.
90 # It will also print the variant-cmd of the machine.
91 # It will also print the variant-cmd of the machine.
92 # It will also print the variant-cmd of the machine.
93 # It will also print the variant-cmd of the machine.
94 # It will also print the variant-cmd of the machine.
95 # It will also print the variant-cmd of the machine.
96 # It will also print the variant-cmd of the machine.
97 # It will also print the variant-cmd of the machine.
98 # It will also print the variant-cmd of the machine.
99 # It will also print the variant-cmd of the machine.
100 # It will also print the variant-cmd of the machine.

```