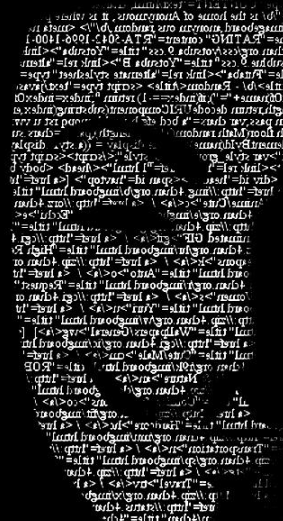
introduction

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:

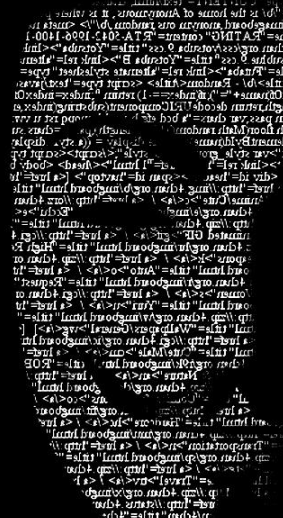
1. The Bourne Shell (The default prompt is the $ character)
2. The C shell (The default prompt is the % character). tcsh is the superset of csh, enhancing user friendliness and speed.

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.



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*

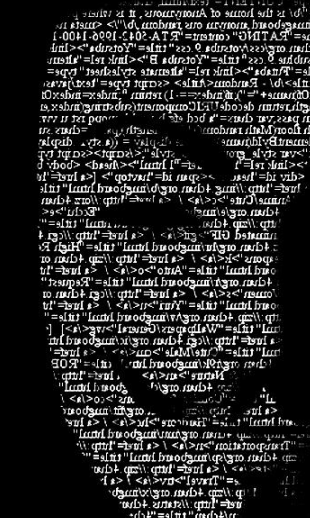
some basic commands in linux

mv- *move/rename files or directories*

rm- *remove files or directories*

mkdir- *make new directory*

rmdir- *remove directory*



text manipulation

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*

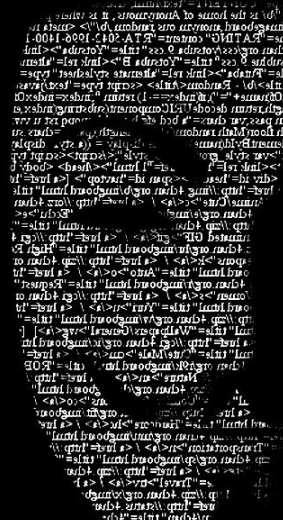
piping and redirection

data streams

`stdin`

`stdout`

`stderr`



piping and redirection

pipe operator |

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

usage:

```
$ command1 | command2
```

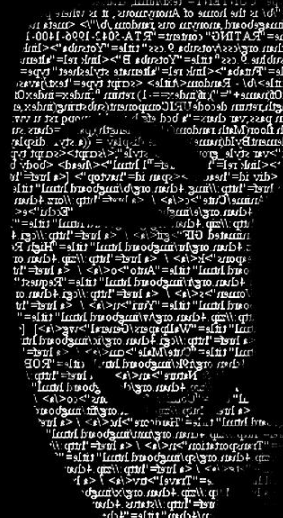
piping and redirection

redirection operator >

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

usage:

```
$ command > file
```



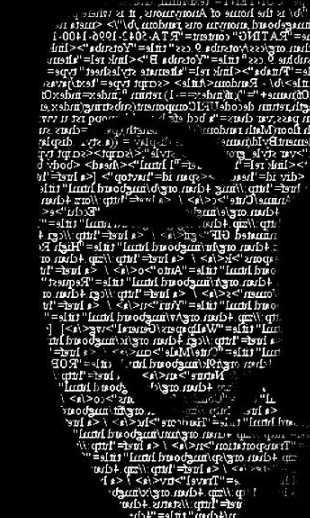
piping and redirection

redirection operator >>

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

usage:

```
$ command >> file
```



piping and redirection

redirection operator <

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

usage:

```
$ command < file
```

grep

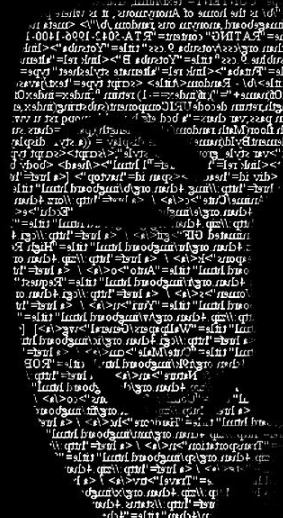
prints matches of **`PATTERN`** in **`file`**.

usage:

```
$ command | grep PATTERN

$ grep PATTERN < file
```

regular expressions



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.

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.

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

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:

```
<h1>Header</h1>
```

The whole line is a match!

alternation

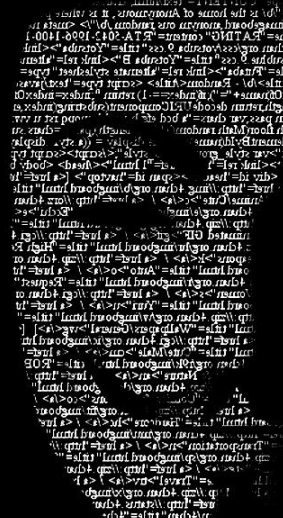
usage:

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

I like cats.

I like dogs.

I like broccoli. (nobody likes broccoli)



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+/**

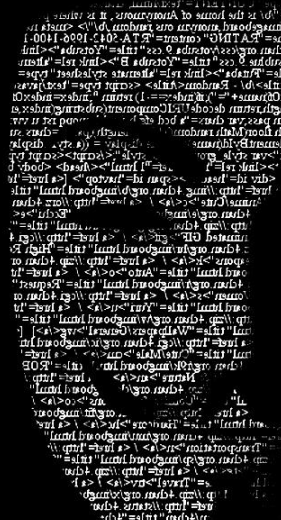
9324802349, which is great, but-

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/

cron

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

crontab

```
$ crontab
```

flags:

```
-l, list user’s crontab

-e, edit user’s crontab

-r, delete user’s crontab
```

crontab

```
┌──────────────────────────── minute (0 - 59)
│ ┌────────────────────────── hour (0 - 23)
│ │ ┌──────────────────────── day of month (1 - 31)
│ │ │ ┌────────────────────── month (1 - 12)
│ │ │ │ ┌──────────────────── day of week (0 - 6)
│ │ │ │ │
│ │ │ │ │
* * * * *  command
```

cron

directories

/etc/cron.d

/etc/cron.daily

/etc/cron.weekly

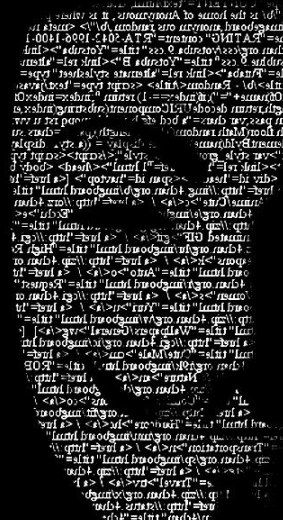
/etc/cron.monthly

files

/etc/crontab

/etc/cron.allow

/etc/cron.deny



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?

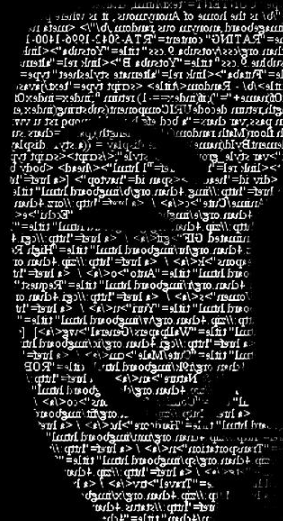
take out the trash

download a podcast

periodically mail yourself

check server downtime

change your background automatically

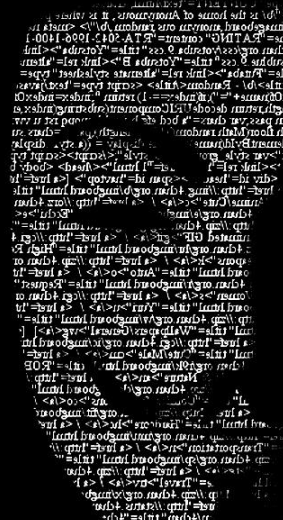


resources

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

```
$ man 8 cron

$ man 5 crontab
```



ssh

secure **sh**ell

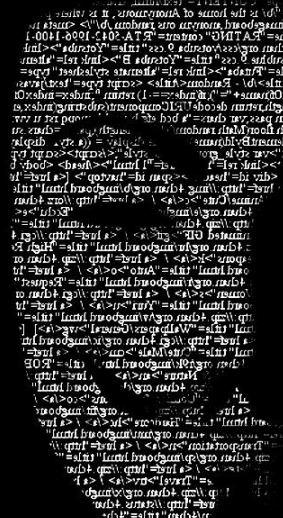
ssh

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*

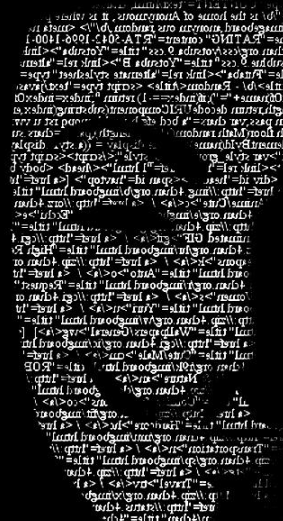
scp - *secure copy (remote file copy program)*

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

resources

https://www.ssh.com/ssh/

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