AtiLink

- A command line utility and a daemon server to facilitate file transfer.
- This is designed for files with relatively huge size.

Assumptions

- One time file transfer; i.e., the connection breaks, transfer stops (after certain retries). There is no syncing procedure involved.
- The transfer happens between two servers. No more servers are involved
- One path (either source or destination) is equipped with a socket address.

Design

Open the design file in excalidraw.com.

TODO

- [ ] Create docker images for *server* and *client* and simulate real world environment.
- [ ] Introduce ssh.
- [ ] Fix bug where empty directories are skipped. That is, if an empty directory is not picked up in recursive iteration.
- [ ] Fix bug where server is not able to read inline path symbols like `~` and `*`.
- [ ] Make remote path reading absolute instead of relative.

Testing

- All integration tests are performed in release mode.
- `atilink --source 127.0.0.1:9099@/home/user/Downloads/Programming Assignment - RSE.pdf --destination client/data`

- `atilink --source server/data/Programming Assignment - RSE.pdf --destination 127.0.0.1:9099@/home/user/Downloads/data`
- Bare minimum unit tests are jotted in the same code files. These can be executed by running `cargo test` from project root.
- To test using binaries:
  - Clone the project.
  - Ensure that you have Cargo installed.
  - Create binaries using command `cargo build --release`. Make sure to use the release flags to get production performance.
  - Binaries are present on the path `target/release/`.

Configuration

- Configurations provided as arguments takes precedence over those defined in file.

Command Line

Server

- `--source` or `-s` to define the source of files. Only one path can denote source.
- `--destination` or `-d` to define the file destination. Only one path can denote destination.
- Remote path should be relative to the server binary.
- Remote address can be provided by prefixing the path with address with `@` as delimiter.

Example

1. `cargo r --release --bin client -- -s [::1]:9099@server/data -d client/data/`
2. `cargo r --release --bin client -- -s server/data -d [::1]:9099@client/data/`

Bug

Make sure that the remote paths are ABSOLUTE to the server binary rather than relative.

Client

- `-p` or `--port` is used define the port the server will listen to. By default this is configured to `[::1]:9099`.
- `-d` or `--debug` is used to run the server in debug mode.

File

- Configuration can be defined in the file `config.toml`.

Edge Cases

- The file source paths not containing socket addresses should exist on localhost.
- If a directory path is provided as a source, it should get get all the paths of the contained files. (RECURSION)
- Source path cannot be empty.
- Destination path cannot be empty.
- Only one path is allowed as source as well as for destination.
- Socket address can be provided either in sources or in destination.
- Socket address validation.
- File can be transferred from client to server OR from server to client.
- If compression algorithm is not provided, don't use compression.
- If checksum algorithm is not provided, don't use validation.
- Validity of file on remote system.

Code Structure

- The code structure is divided into three sub projects.
  - Client responsible for providing command line interface.
  - Server is a daemon process running on remote system.
  - Common provides code-base utilized by both server and client.
- Using end of file marker to notify the receiver about completion. This is done to avoid sending file size in the beginning since, getting file size can be time taking.
- Since the file size can be huge, compression is done for individual chunks rather than loading the complete file and compressing it.
- Enabling **compression** leads to addition of bytes to the start of each chunk since each chunk possess different length after encoding.
- `Base64` encoding is not required since not text based interpretation happens at any point.
- File transfer metadata take place beforehand to decide upon the compression algorithms to incorporate.

Server

- A daemon process constantly listening to `9099` port. Port can be configured by `-p` or `--port` as cli arguments.
- Any transfer is initialized by a `Role` sent by client. This decides whether the server acts as a `Source` or as a `Sink`.

Client

- A command line utility to send files remotely or to receive a remote file.
- The client is responsible for setting the configurations to be used including compression and checksum algorithms. These are then communicated to the remote destination server.
- Also responsible for validations of input variables

Observations

- Smaller chunks are error prone. This is due to the reason that smaller chunks leads to huge read-write operations on stream. Any bit up-down can manipulate the length and disrupt the process. *Fix:* Increase the chunk size
- While uploading data we have absolute path. While downloading we have relative path.

Stats

- Client OS: Arch Linux
- Server OS: Arch Linux
- Tests are performed in release mode. `cargo run --release`.

S. no.	**File Size**	**Compression**	**Checksum**	**Max. Chunk Size**	**Time Taken**
1.	21Gb	None	None	1mb	70.917130829s
2.	21Gb	None	None	500kb	70.190991056s
3.	21Gb	None	None	100kb	61.774868846s
4.	21Gb	None	None	50kb	58.836703634s
5.	21Gb	None	None	1kb	55.567423464s
6.	21Gb	None	None	100b	40.021080039s
7.	21Gb	None	None	10b	39.79148465s
8.	21Gb	GZip	None	1mb	645.960120126s
9.	21Gb	GZip	None	100kb	644.537627798s
10.	21Gb	GZip	None	10b	639.749744939s
11.	21Gb	None	Sha256	1mb	82.673628001s
12.	21Gb	None	Sha256	100kb	63.069202152s
13.	21Gb	None	Sha256	10b	71.960773039s
14.	21Gb	Zlib	Sha256	1mb	648.733519664s

Observations

- No *Compression* and *Checksum* provides faster times at the cost of bytes transferred. This can be justified by the fact that

individual chunks are not compressed and validated over and over again.
- Compression increases the transfer time significantly.
- Checksums doesn't affect the transfer times since these are of small fixed lengths (64 bytes).

Note

Time can vary for different use cases.

1. Storage device being used (ssd, hard disk, external drive) which can manipulate rw speeds.
2. The type of files transferred.
3. Network bandwidth.