yui

hello world!

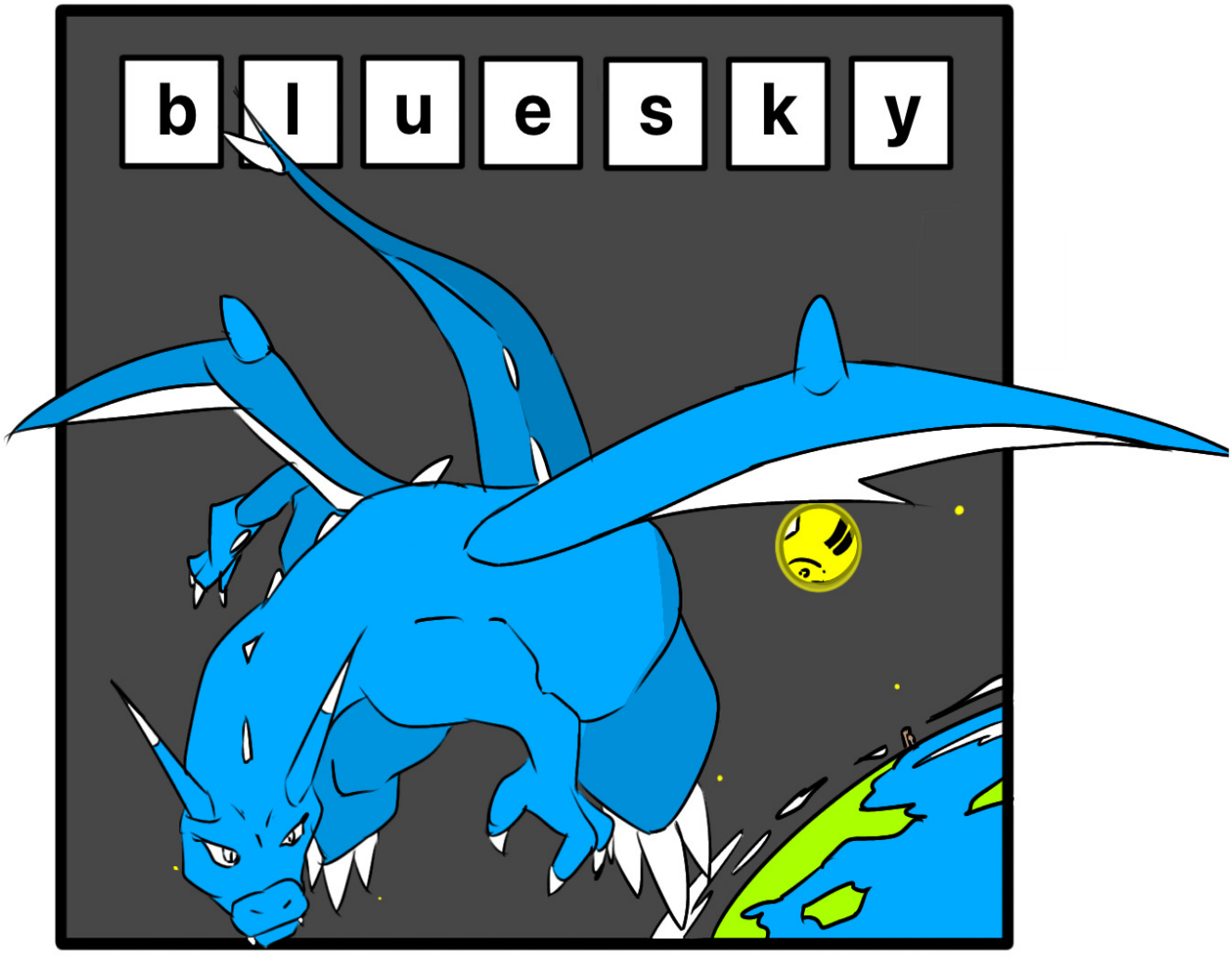


Table of Contents

hello world! bluesky

download | web

lang : en / ja

part 1

part 1

This book is an introduction to programming for users of bluesky.

It mainly uses the `rust` programming language and the os terminal environment.

The content is to create a simple program for card game that can be played on `bluesky` or `mastodon`.

If you make this program, you can train the cards you have.

In this manual, you will learn to hit the bluesky api, create commands in rust, and so on.

The Quick Start in this chapter provides the minimum necessary explanation for technicians.

First-time users should skip this chapter and start with part 2.

updated : 2023-07-29

quick start

quick start

```
handle=yui.syui.ai
curl -sL
"https://bsky.social/xrpc/com.atproto.repo.listRecords?
repo=${handle}&collection=app.bsky.feed.post&limit=1"
```

Send the following mention to @yui.syui.ai.

```
# card account create
@yui.syui.ai /card

# get egg card
@yui.syui.ai /card egg
```

You will then receive an egg card. Anyone can perform this hidden command. If you already have one, it will be displayed.

This card can be grown by converting your did to `base64` and sending it to `@yui.syui.ai`.

```
$ echo did:plc:4hqjfn7m6n5hno3doamuhgef|base64
ZGlkOnBsYzo0aHFqZm43bTZuNWhubzNkb2FtdWhnZWYK

@yui.syui.ai /egg
ZGlkOnBsYzo0aHFqZm43bTZuNWhubzNkb2FtdWhnZWYK
```

Note that this will consume one day's battle points.

If you send this by command, it will look like this

env

```
data=`curl -sL -X POST -H "Content-Type: application/json" -
d "{\"identifier\":\"$handle\",\"password\":\"$pass\"}"
https://bsky.social/xrpc/com.atproto.server.createSession`
token=`echo $data|jq -r .accessJwt`
did=`echo $data|jq -r .did`
base=`echo $did|base64`

handle_m=yui.syui.ai
did_m=`curl -sL -X GET -H "Content-Type: application/json" -
H "Authorization: Bearer $token"
"https://bsky.social/xrpc/app.bsky.actor.getProfile?
actor=${handle_m}"|jq -r .did`
at=@${handle_m}
s=0
e=`echo $at|wc -c`
text="$at /egg $base"
col=app.bsky.feed.post
created_at=`date --iso-8601=seconds`
```

json

```
json="{
        \"did\": \"$did\",
        \"repo\": \"$handle\",
        \"collection\": \"$col\",
        \"record\": {
            \"text\": \"$text\",
            \"\$type\": \"$col\",
            \"createdAt\": \"$created_at\",
            \"facets\": [
```

```
            {
                \"\$type\": \"app.bsky.richtext.facet\",
                \"index\": {
                    \"byteEnd\": $e,
                    \"byteStart\": $s
                },\"features\": [
                {
                    \"did\": \"$did_m\",
                    \"\$type\":
\"app.bsky.richtext.facet#mention\"
                }
                ]
            }
            ]
        }
}"
```

post

```
curl -sL -X POST -H "Content-Type: application/json" \
         -H "Authorization: Bearer $token" \
         -d "$json" \

https://bsky.social/xrpc/com.atproto.repo.createRecord
```

example

example

Here is an example of the use of lexicons.

option

```
# reverse
curl -sL
"https://bsky.social/xrpc/com.atproto.repo.listRecords?
repo=${handle}&collection=app.bsky.feed.post&reverse=true"
```

login

```
handle=yui.syui.ai
pass=xxx
curl -sL -X POST -H "Content-Type: application/json" \
         -d "
{\"identifier\":\"$handle\",\"password\":\"$pass\"}" \

https://bsky.social/xrpc/com.atproto.server.createSession

# token
token=`curl -sL -X POST -H "Content-Type: application/json"
-d "{\"identifier\":\"$handle\",\"password\":\"$pass\"}"
https://bsky.social/xrpc/com.atproto.server.createSession|jq
-r .accessJwt`

# did
did=`curl -sL -X POST -H "Content-Type: application/json" -d
"{\"identifier\":\"$handle\",\"password\":\"$pass\"}"
https://bsky.social/xrpc/com.atproto.server.createSession|jq
-r .did`

# profile
curl -sL -X GET -H "Content-Type: application/json" \
         -H "Authorization: Bearer $token" \

"https://bsky.social/xrpc/app.bsky.actor.getProfile?
actor=${handle}"

# notify
curl -sL -X GET -H "Content-Type: application/json" \
         -H "Authorization: Bearer $token" \

https://bsky.social/xrpc/app.bsky.notification.listNotificat
ions
```

post

```
col=app.bsky.feed.post

created_at=`date --iso-8601=seconds`

json="{
    \"repo\": \"$handle\",
        \"did\": \"$did\",
        \"collection\": \"$col\",
        \"record\": {
            \"text\": \"hello world\",
            \"createdAt\": \"$created_at\"
```

```
        }
}"

# post
curl -sL -X POST -H "Content-Type: application/json" \
         -H "Authorization: Bearer $token" \
         -d "$json" \

https://bsky.social/xrpc/com.atproto.repo.createRecord
```

mention

example.json

```
{
  "did": "did:plc:4hqjfn7m6n5hno3doamuhgef",
  "repo": "yui.syui.ai",
  "collection": "app.bsky.feed.post",
  "record": {
    "text": "test",
    "$type": "app.bsky.feed.post",
    "createdAt": "2023-07-20T13:05:45+09:00",
    "facets": [
      {
        "$type": "app.bsky.richtext.facet",
        "index": {
          "byteEnd": 13,
          "byteStart": 0
        },
        "features": [
          {
            "did": "did:plc:4hqjfn7m6n5hno3doamuhgef",
            "$type": "app.bsky.richtext.facet#mention"
          }
        ]
      }
    ]
  }
}

# mention
col=app.bsky.feed.post
handle_m=yui.syui.ai
did_m=`curl -sL -X GET -H "Content-Type: application/json" -
H "Authorization: Bearer $token"
"https://bsky.social/xrpc/app.bsky.actor.getProfile?
actor=${handle_m}"|jq -r .did`
at=@${handle_m}
s=0
e=`echo $at|wc -c`

json="{
        \"did\": \"$did\",
        \"repo\": \"$handle\",
        \"collection\": \"$col\",
        \"record\": {
            \"text\": \"$text\",
            \"\$type\": \"app.bsky.feed.post\",
            \"createdAt\": \"$created_at\",
            \"facets\": [
            {
                \"\$type\": \"app.bsky.richtext.facet\",
                \"index\": {
                    \"byteEnd\": $e,
                    \"byteStart\": $s
                },\"features\": [
```

```
                {
                    \"did\": \"$did_m\",
                    \"\$type\":
\"app.bsky.richtext.facet#mention\"
                }
                ]
            }
            ]
        }
}"

curl -sL -X POST -H "Content-Type: application/json" \
         -H "Authorization: Bearer $token" \
         -d "$json" \

https://bsky.social/xrpc/com.atproto.repo.createRecord
```

output

```
{"uri":"at://did:plc:4hqjfn7m6n5hno3doamuhgef/app.bsky.feed.
post/3k2wkbvcasf24","cid":"bafyreiecswq5qhk7f4xxztevzbfynocs
gmjrmr3hwqoluhhzvqgowalivi"}
```

reply

example.json

```
{
  "repo": "yui.syui.ai",
  "did": "did:plc:4hqjfn7m6n5hno3doamuhgef",
  "collection": "app.bsky.feed.post",
  "record": {
    "text": "reply",
    "createdAt": "2023-07-20T13:05:45+09:00",
    "reply": {
      "root": {
        "cid":
"bafyreiecswq5qhk7f4xxztevzbfynocsgmjrmr3hwqoluhhzvqgowalivi
",
        "uri":
"at://did:plc:4hqjfn7m6n5hno3doamuhgef/app.bsky.feed.post/3k
2wkbvcasf24"
      },
      "parent": {
        "cid":
"bafyreiecswq5qhk7f4xxztevzbfynocsgmjrmr3hwqoluhhzvqgowalivi
",
        "uri":
"at://did:plc:4hqjfn7m6n5hno3doamuhgef/app.bsky.feed.post/3k
2wkbvcasf24"
      }
    }
  }
}

# reply
col=app.bsky.feed.post
uri=at://did:plc:4hqjfn7m6n5hno3doamuhgef/app.bsky.feed.post
/3k2wkbvcasf24
cid=bafyreiecswq5qhk7f4xxztevzbfynocsgmjrmr3hwqoluhhzvqgowal
ivi

json="{
    \"repo\": \"$handle\",
        \"did\": \"$did\",
        \"collection\": \"$col\",
```

```
        \"record\": {
            \"text\": \"reply\",
            \"createdAt\": \"$created_at\",
            \"reply\": {
                \"root\": {
                    \"cid\": \"$cid\",
                    \"uri\": \"$uri\"
                },
                \"parent\": {
                    \"cid\": \"$cid\",
                    \"uri\": \"$uri\"
                }
            }
        }
}"

curl -sL -X POST -H "Content-Type: application/json" \
         -H "Authorization: Bearer $token" \
         -d "$json" \

https://bsky.social/xrpc/com.atproto.repo.createRecord
```

like

```
# reply
col=app.bsky.feed.like
uri=at://did:plc:4hqjfn7m6n5hno3doamuhgef/app.bsky.feed.post
/3k2wkbvcasf24
cid=bafyreiecswq5qhk7f4xxztevzbfynocsgmjrmr3hwqoluhhzvqgowal
ivi

json="{
    \"repo\": \"$handle\",
        \"did\": \"$did\",
        \"collection\": \"$col\",
        \"record\": {
            \"createdAt\": \"$created_at\",
            \"subject\": {
                \"cid\": \"$cid\",
                \"uri\": \"$uri\"
            }
        }
}"

curl -sL -X POST -H "Content-Type: application/json" \
         -H "Authorization: Bearer $token" \
         -d "$json" \

https://bsky.social/xrpc/com.atproto.repo.createRecord
```

follow

```
col=app.bsky.graph.follow
handle_m=yui.syui.ai
did_m=`curl -sL -X GET -H "Content-Type: application/json" -
H "Authorization: Bearer $token"
"https://bsky.social/xrpc/app.bsky.actor.getProfile?
actor=${handle_m}"|jq -r .did`

json="{
    \"repo\": \"$handle\",
        \"did\": \"$did\",
        \"collection\": \"$col\",
        \"record\": {
            \"createdAt\": \"$created_at\",
```

```
            \"subject\": \"$did_m\"
        }
}"

curl -sL -X POST -H "Content-Type: application/json" \
         -H "Authorization: Bearer $token" \
         -d "$json" \

https://bsky.social/xrpc/com.atproto.repo.createRecord
```

unfollow

```
$ curl -sL -X GET -H "Content-Type: application/json" \
         -H "Authorization: Bearer $token" \

"https://bsky.social/xrpc/app.bsky.graph.getFollowers?
actor=${handle}&cursor=${cursor}" \
         |jq -r ".cursor"

1688489398761::bafyreieie7opxd5mojipvk3xe3h65u3qvpungskqxaml
depctfbd6xhdcu

cursor=1688489398761::bafyreieie7opxd5mojipvk3xe3h65u3qvpung
skqxamldepctfbd6xhdcu

$ curl -sL -X GET -H "Content-Type: application/json" \
         -H "Authorization: Bearer $token" \

"https://bsky.social/xrpc/app.bsky.graph.getFollowers?
actor=${handle}&cursor=${cursor}" \
         |jq -r ".followers|.[0].viewer.followedBy"

at://did:plc:uqzpqmrjnptsxezjx4xuh2mn/app.bsky.graph.follow/
3k2wkjr6cnj2x

col=app.bsky.graph.follow
rkey=at://did:plc:uqzpqmrjnptsxezjx4xuh2mn/app.bsky.graph.fo
llow/3k2wkjr6cnj2x

handle_m=yui.syui.ai
did_m=`curl -sL -X GET -H "Content-Type: application/json" -
H "Authorization: Bearer $token"
"https://bsky.social/xrpc/app.bsky.actor.getProfile?
actor=${handle_m}"|jq -r .did`

json="{
    \"repo\": \"$handle\",
        \"did\": \"$did\",
        \"collection\": \"$col\",
        \"rkey\":\"$rkey\",
        \"record\": {
            \"createdAt\": \"$created_at\",
            \"subject\": \"$did_m\"
        }
}"

curl -sL -X POST -H "Content-Type: application/json" \
         -H "Authorization: Bearer $token" \
         -d "$json" \

https://bsky.social/xrpc/com.atproto.repo.deleteRecord
```

part 2

part 2

In this chapter, we will explain the most commonly used words and the environment.

Mainly, this explanation is aimed at the different operating environments for different OSs.

This chapter provides a summary of the installation of packages and other information used in this manual.

Please refer to this chapter if you are not familiar with it.

bluesky

bluesky

The `at` will henceforth be `atproto`.

bluesky is positioned as a model service for `atproto`, which is currently being developed and operated by bsky.team.

The goal of bsky.team is to enable `atproto` to be adopted behind the scenes of various services and to allow communication between services.

Until now, accounts were only valid within a service. Therefore, it was necessary to switch accounts for each service. This is an attempt to change this.

bluesky will work with `pds`, `plc`, and `bgs`.

To describe the role of each, `pds` is the main body of bluesky.

plc is like `dns` and registers handle and did, and performs name resolution.

Basically, bluesky works only with pds.

However, when an account is created, it connects to plc, so if there is no connection to plc, an error will occur.

plc is not necessarily needed when an account is created; it is needed when a handle is registered or changed.

bgs builds the timeline of the account when connecting to other pds.

graph TD; A[pds]-->B[plc]; C[pds]-->B[plc]; D[pds]-->B[plc];
graph TD; A[pds]-->B[bgs]; C[pds]-->B[bgs]; D[pds]-->B[bgs];

dns

What is a dns above is a server that performs name resolution on the Internet.

You connect to the internet by the number `ip address`.

For example, to connect to google, it is `172.217.25.174`.

```
$ dig google.com
google.com.             291     IN      A
172.217.25.174
```

Try this number in your browser to see if it works. It should lead to `google.com`.

However, since numbers are difficult for humans to remember and handle, they are usually given names that are replaced by letters of the alphabet.

The server responsible for connecting the name to the number `ip address` is called dns.

The following is a command that displays the route to the desired host. You can see that it is connected via several servers.

```
$ traceroute google.com
20.27.177.113
17.253.144.10
172.217.25.174
```

If you want to know your ip address, you can use `ipinfo.io`.

```
$ curl -sL ipinfo.io
20.27.177.113
```

plc

These are the most commonly used plc's at this time. All are provided by bsky.team.

https://plc.directory

https://plc.bsky-sandbox.dev

Specifically, it is used as follows

```
https://plc.directory/export
```

```
https://plc.directory/export?after=1970-01-01T00:00:00.000Z
```

```
https://plc.directory/did:plc:oc6vwdlmk2kqyida5i74d3p5
```

```
https://plc.directory/did:plc:oc6vwdlmk2kqyida5i74d3p5/log
```

.env

```
#DID_PLC_URL=https://plc.directory
DID_PLC_URL=https://plc.bsky-sandbox.dev
```

bast service

Since both api and pds are publicly available for bluesky, various services are being developed.

The following is a list of representative services.

https://firesky.tv : Global Timeline Stream. You can configure and filter it in many ways.

https://bsky.jazco.dev : User Visualization

https://bsky.jazco.dev/stats : Number of user posts

https://vqv.app : User profile aggregation, etc.

https://atscan.net : Scanning of pds and did

https://skybridge.fly.dev : url to do bluesky in mastodon client

https://tapbots.com/ivory : Support for mastodon client ivory

https://skyfeed.app : Generation of feeds

terminal

terminal

The goal here is to provide the necessary commands and environment for each OS.

The `terminal` is the famous cmd (command prompt) in windows. In simple terms, it refers to a black screen. It is also called `terminal` or `term.

There are various terminals, or applications (software) in `terminal'.

Personally, I recommend wezterm, but here we will use an os-specific one.

package manager

First, we will explain `package manager`.

In this case, you need to install a package manager for each OS.

Note that packages and programs here can also be referred to as apps.

Think of a package manager as something that simplifies the installation of an app.

Usually, an app works by building or compiling a source (source) and executing the binary that is created.

In the case of windows, `.exe` is a binary.

Binaries differ depending on the operating system.

Incidentally, source is often abbreviated to `src` and binary is often abbreviated to `bin`.

To return to the topic, the package manager automatically handles which packages (binaries) are downloaded from where and where they are placed.

The reason why this is done is that it takes time to build a source.

Therefore, most packages are simply downloaded from the server (server) as binaries that have already been built on the OS in question.

Most of them are called package managers.

From now on, you will install this package manager and use it from the `terminal`.

windows

This section describes the required environment for windows users.

- winget
- scoop
- windows terminal
- wsl

Be careful with the windows environment; remember that windows basically does not work as per the docs.

For example, most of the commands in github/microsoft will not work. It may not work.

Therefore, you will need to configure and read them according to your environment.

First, install winget as a windows package manager.

Press `win+r` and type `powershell` to start powershell. powershell will henceforth be abbreviated as `pwsh`.

Execute the following command.

pwsh

```
Install-Module -Name Microsoft.WinGet.Client

Untrusted repository
You are installing the modules from an untrusted repository.
If you trust this repository, change its
InstallationPolicy value by running the Set-PSRepository
cmdlet. Are you sure you want to install the modules from
'PSGallery'?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend
[?] Help (default is "N"): A
```

Next, install windows terminal.

search package

```
winget search "windows terminal"

 Name                      Id
Version      Source
 ------------------------------------------------------------
---------------------
 Windows Terminal          9N0DX20HK701
Unknown      msstore
 Windows Terminal Preview 9N8G5RFZ9XK3
Unknown      msstore
 Windows Terminal          Microsoft.WindowsTerminal
1.16.10261.0 winget
 Windows Terminal Preview Microsoft.WindowsTerminal.Preview
1.17.10234.0 winget
```

install terminal

```
winget install 9N0DX20HK701
or
winget install Microsoft.WindowsTerminal
```

Since windows is very unwieldy with shell, we will run linux (ubuntu) with wsl. Basically, rust and shell are explained assuming a linux environment.

setting wsl

```
wsl --install
wsl --install -d Ubuntu
```

If you prefer a windows environment instead of linux, you can install `curl` etc. from scoop or other package managers.

install scoop

```
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser #
Optional: Needed to run a remote script the first time
irm get.scoop.sh | iex
```

pwsh

```
scoop install curl git rust
```

Install and update `pwsh`.

```
winget install Microsoft.PowerShell
winget upgrade --all
```

mac

- terminal
- homebrew

For `mac`, use the default terminal.

Open `finder` and press `cmd+shift+u`. You will find `terminal(terminal.app)' in it.

First install the package manager homebrew.

install brew

```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/inst
all.sh)"
```

```
brew install curl git zsh rust
```

linux

I will omit the description for linux users as it needs no explanation.

I will use archlinux.

```
pacman -Syu curl git zsh rust
```

```
$ cargo version
cargo 1.70.0
```

shell

shell

Now that you think you have the package manager installed, try installing `curl` first.

```
# windows
scoop install curl

# mac
brew install curl

# linux(ubuntu)
sudo apt install curl
```

Then, execute the following command in terminal.

```
curl https://bsky.social/xrpc/_health
```

The result should return the pds version of bluesky(bsky.social) as follows

output

```
{"version":"b2ef3865bc143bfe4eef4a46dbd6a44053fa270d"}
```

If `curl` does not work properly, it may be that the installed binaries do not have a path.

This is also likely to occur mainly on windows.

Here is a little explanation about paths.

path

When the terminal is started, a program called shell is waiting there.

The user executes commands through this shell.

There are various kinds of shells.

For example, windows has microsoft's `cmd` and `pwsh` shells.

For unix(mac), linux(ubuntu), there are `bash`, `zsh` and so on.

The shell can omit directories added to `PATH` when executing commands.

For example, suppose `curl` is installed in `/usr/bin/curl`. In this case, shell should execute the following command

```
/usr/bin/curl --help
```

However, if `/usr/bin` is added to the `PATH`, the directory description can be omitted.

```
curl --help
```

To find the location of the main body of the program (binary), use the following command.

```
which curl
```

However, it cannot be used unless the path is passed.

To pass path, put the directory in question in an environment variable.

```
PATH=$PATH:/usr/bin
```

Note that a directory is sometimes called a `dir` or `folder`.

Notation "$"

Next, a note on the description format of shell.

```
which curl
```

```
$ which curl
`
```

These have the same meaning.

If you are describing the execution of a shell in writing, it is customary to prefix it with `` `$ ``.

This `` `$ `` means "run in shell".

For example, if you want to include the result of the command with the execution, it would be as follows.

```
$ which curl
/usr/bin/curl
```

This is because it is often the case that you want to put the command and the result together, and if there is no `$`, it will be difficult to tell which is the command and which is the result.

In this manual, `` `$ `` is omitted as much as possible to avoid the harm of copying.

However, it is believed that all code layouts should include `$` when executed in shell.

shebang

Next, we will discuss shell script and shebang.

This area varies from shell to shell, but we will assume `bash`.

Please write the following in a text file, give it execute permission, and run it.

test.sh

```
#! /bin/bash
curl https://bsky.social/xrpc/_health
```

The following command grants execute permission and executes it.

```
chmod +x test.sh
. /test.sh
```

Then the version of bsky.social will be output.

```
{"version": "b2ef3865bc143bfe4eef4a46dbd6a44053fa270d"}
```

The first line of the text file `#!/bin/bash` is what is called a shiban.

Here, it specifies which programming language the text file is to be executed in.

The following is a brief description of the programming language.

rust

rust

Next, install the programming language `rust`.

```
brew install rust
```

Rust runs through a package manager called `cargo`.

Use `cargo` to check the version.

```
$ cargo version
cargo 1.71.0
```

Rust is said to be a very difficult language among various programming languages.

Its characteristics are that it is stable and works once built, but it takes a long time to get it running.

It may also take longer than other languages to add new implementations.

lang

Programming languages are sometimes abbreviated as `lang`.

For example, there is a programming language called `go`.

However, the word `go` has many meanings. Therefore, it is sometimes called `golang` or `go-lang`.

part 3

part 3

In this chapter, we will write specific code in `rust` to get the program moving.

hello world

init

First, create a program template in rust.

```
mkdir -p ~/rust
cd ~/rust
cargo init
```

```
.
├── Cargo.toml
└── src
    └── main.rs
```

You can create these files yourself or with `init`.

Cargo.toml

```
[package]
name = "rust"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at https://doc.rust-
lang.org/cargo/reference/manifest.html

[dependencies]
```

src/main.rs

```
fn main() {
    println!("Hello, world!");
}
```

editor

Next, let's check the contents of the program.

To check, use `editor` (editor). I use `vim`, but I would recommend visual studio.

```
brew install vim
vim src/main.rs
```

src/main.rs

```
fn main() {
    println!("Hello, world!");
}
```

This is a program that outputs the string `hello world!

build

You can `build` this src and convert it to `binary`, i.e., the app itself, so that you can run it on that computer.

```
cargo build
```

target/debug/rust

```
target
└── debug
    ├── rust ← binary
    └── rust.d
```

rust is a very good language because it is one-binary, meaning that the compiled result is a single file.

```
$ ./target/debug/rust
Hello, world!
```

seahorse

seahorse

Next, introduce the framework ksk001100/seahorse.

This framework is for writing a cli (command line interface).

A cli is simply the same command as `which` or `curl` that you have been executing. We will now create our own command.

It may sound difficult to some, but it is easy if you use a wonderful framework called `seahorse`.

First you install `seahorse`, but to install the library in rust, write the package name in `Cargo.toml`. This will automatically install the library when you build.

Note that library is sometimes abbreviated as `lib`.

Cargo.toml

```
[package]
name = "rust"
version = "0.1.0"
edition = "2021"

[dependencies]
seahorse = "*"
```

Then we write the body code that uses `seahorse`.

src/main.rs

```
use seahorse::{App, Context};
use std::env;

fn main() {
    let args: Vec<String> = env::args().collect();
    let app = App::new(env!("CARGO_PKG_NAME"))
        .action(s)

        ;
    app.run(args);
}

fn s(_c: &Context) {
    println!("Hello, world!");
}
```

The contents are very simple. When the command is executed, `Hello, world!` is output.

```
$ cargo build
$ ./target/debug/rust
Hello, world!
```

What is different now, for example, is that the `help` option is automatic.

```
$ ./target/debug/rust -h

Name:
        rust
Flags:
        -h, --help : Show help
```

To help readers understand the awesomeness of `seahorse`, ask them to think about the application themselves.

src/main.rs

```
use seahorse::{App, Context, Command};
use std::env;

fn main() {
    let args: Vec<String> = env::args().collect();
    let app = App::new(env!("CARGO_PKG_NAME"))
        .action(s)

        .command(
                Command::new("yes")
                .alias("y")
                .action(y),
                )
        .command(
                Command::new("no")
                .alias("n")
                .action(n),
                )

        ;
        app.run(args);
}

fn s(_c: &Context) {
    println!("Hello, world!");
}

fn y(_c: &Context) {
    println!("yes");
}

fn n(_c: &Context) {
    println!("no");
}
```

Try writing this and executing the command you created.

```
$ ./target/debug/rust

$ ./target/debug/rust y

$ ./target/debug/rust n
```

The differences and key points of the code are as follows

```
  use seahorse::{App, Context
+ , Command
  };

+
.command(
        Command::new("yes")
        .alias("y")
        .action(y),
        )

fn y(_c: &Context) {
    println!("yes");
}
```

Feel free to rewrite or add these values to create your own commands.

Here, the value specified in `Command::new` means the option name.

In this case, `rust yes` is the issue of this command.

You can specify omission by `alias("y")`. In this case, `rust y`.

The `action(y)` specifies the function `fn y`, the contents of which will be executed. The processing of the command body is written in `action`.

By the way, `action` does not necessarily have to be a function.

For example, try adding the following code in place. The command is `rust t` or `rust t foo`.

src/main.rs

```
.command(
        Command::new("test")
        .alias("t")
        .action(|c| println!("Hello, {:?}", c.args)),
        )
```

```
$ ./target/debug/rust t bluesky
Hello, ["bluesky"]
```

cli

- CLI

The term `cli` has many meanings. It can refer to the cli tool as described above, or it can refer to terminal operations in general.

cui and gui

- CUI, GUI

There are two kinds of cui and gui. The one we are using now is `cui`.

The term `cli` is used in almost the same way.

cui means terminal operation, and gui means operation on graphical os.

It is divided into c-ui and ui, which is just ui. ui stands for user interface.

All common os such as windows and mac are based on gui operation.

author

The author of seahorse is ksk.

Thanks for creating a great framework.

reqwest

reqwest

Before explaining seanmonstar/reqwest in RUST, try the following command.

```
$ curl -sL
"https://bsky.social/xrpc/com.atproto.repo.listRecords?
repo=support.bsky.team&collection=app.bsky.feed.post"
```

This hits the api to get the timeline for support.bsky.social.

You can understand that `reqwest` is mainly a rust lib to hit the api.

Now, let's write the actual code.

Cargo.toml

```
[package]
name = "rust"
version = "0.1.0"
edition = "2021"

[dependencies]
seahorse = "*"
reqwest = "*"
tokio = { version = "1", features = ["full"] }
```

```
use seahorse::{App, Context, Command};
use std::env;

fn main() {
    let args: Vec<String> = env::args().collect();
    let app = App::new(env!("CARGO_PKG_NAME"))
        .action(s)
        .command(
                Command::new("yes")
                .alias("y")
                .action(y),
                )
        .command(
                Command::new("no")
                .alias("n")
                .action(n),
                )
        .command(
                Command::new("test")
                .alias("t")
                .action(|c| println!("Hello, {:?}",
c.args)),
                )
        .command(
                Command::new("bluesky")
                .alias("b")
                .action(c_list_records),
                )

        ;
        app.run(args);
}

fn s(_c: &Context) {
    println!("Hello, world!");
}
```

```
fn y(_c: &Context) {
    println!("yes");
}

fn n(_c: &Context) {
    println!("no");
}

#[tokio::main]
async fn list_records() -> reqwest::Result<()> {
    let client = reqwest::Client::new();
    let handle= "support.bsky.team";
    let col = "app.bsky.feed.post";
    let body =
client.get("https://bsky.social/xrpc/com.atproto.repo.listRe
cords")
        .query(&[("repo", &handle),("collection", &col)])
        .send()
        .await?
        .text()
        .await?;
    println!("{}", body);
    Ok(())
}

fn c_list_records(_c: &Context) {
    list_records().unwrap();
}
```

This is then `cargo build` and run the command as usual.

```
$ ./target/debug/rust b
```

The following is an `example`, i.e., code with useless command options removed.

The main points of the code are as follows.

```
.command(
        Command::new("bluesky")
        .alias("b")
        .action(c_list_records),
        )

#[tokio::main]
async fn list_records() -> reqwest::Result<()> {
    let client = reqwest::Client::new();
    let handle= "support.bsky.team";
    let col = "app.bsky.feed.post";
    let body =
client.get("https://bsky.social/xrpc/com.atproto.repo.listRe
cords")
        .query(&[("repo", &handle),("collection", &col)])
        .send()
        .await?
        .text()
        .await?;
    println!("{}", body);
    Ok(())
}

fn c_list_records(_c: &Context) {
    list_records().unwrap();
}
```

query

Try adding `query`. Now the output will be on one line and in order of oldest to newest.

src/main.rs

```
async fn list_records() -> reqwest::Result<()> {
    let client = reqwest::Client::new();
    let handle= "support.bsky.team";
    let col = "app.bsky.feed.post";
    let body =
client.get("https://bsky.social/xrpc/com.atproto.repo.listRe
cords")
        //.query(&[("repo", &handle),("collection", &col)])
        .query(&[("repo", &handle),("collection", &col),
("limit", &"1"),("revert", &"true")])
        .send()
        .await?
        .text()
        .await?;
    println!("{}", body);
    Ok(())
}
```

part 4

part 4

In this chapter, you will write up the RUST code using `seahorse`, `reqwest`, and bring the program to completion.

bluesky's lexicons will be important.

If you are not sure, please refer to part 1.

ai

ai

This section is designed to be original with playful elements. Each of you can set it to whatever you like.

First of all, the name of the command application. So far we have used `rust`. Because the program name created by `cargo init` is `rust`. This will automatically give you a folder name.

Let's change this to `ai`.

Cargo.toml

```
[package]
name = "ai"
```

If you set any name you like, please read the command name, etc. differently in the following explanations.

```
$ cargo build
$ ./target/debug/ai -h

Name:
        ai
Flags:
        -h, --help : Show help
Commands:
        y, yes     :
        n, no      :
        t, test    :
        b, bluesky :
```

cleanup

Next, let's reduce the number of command options that we don't need, although we don't want to leave any behind.

```
use seahorse::{App, Context, Command};
use std::env;

fn main() {
    let args: Vec<String> = env::args().collect();
    let app = App::new(env!("CARGO_PKG_NAME"))
        .action(c_list_records)
        .command(
                Command::new("bluesky")
                .alias("b")
                .action(c_list_records),
                )

        ;
        app.run(args);
}

#[tokio::main]
async fn list_records() -> reqwest::Result<()> {
    let client = reqwest::Client::new();
    let handle= "support.bsky.team";
    let col = "app.bsky.feed.post";
    let body =
client.get("https://bsky.social/xrpc/com.atproto.repo.listRe
cords")
```

```
        .query(&[("repo", &handle),("collection", &col),
("limit", &"1"),("revert", &"true")])
        .send()
        .await?
        .text()
        .await?;
    println!("{}", body);
    Ok(())
}

fn c_list_records(_c: &Context) {
    list_records().unwrap();
}
```

ascii

I will include the so-called `ascii art`.

Generate it using TheZoraiz/ascii-image-converter.

```
$ ascii-image-converter ai.png -H 50
```

Change `.action(c_list_records)` to `.action(c_ascii_art)` in `src/main.rs`.

src/main.rs

```
//.action(c_list_records)
.action(c_ascii_art)
```

Note that in rust, the `//` at the beginning of a sentence is a comment.

Comments are not interpreted as code. Therefore, when you write a note, use this.

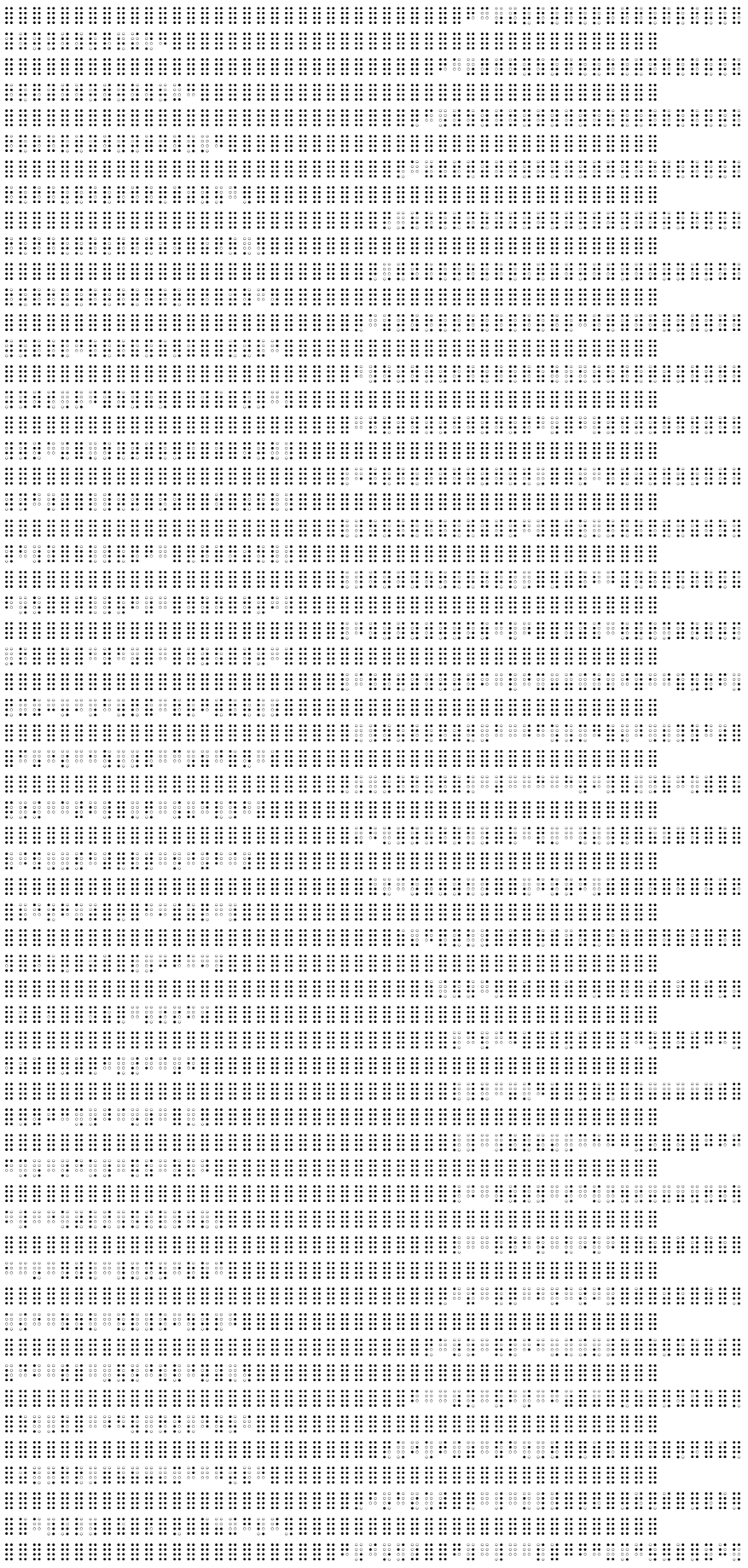
src/main.rs

```
// This is the ASCII art of the eye
// See the function "c_ascii_art" for the contents
.action(c_ascii_art)
```

Then, add the following code (function) to the last line of `src/main.rs`.

src/main.rs

```
fn c_ascii_art(_c: &Context) {
   let body = "
⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡟⠀⣤⠀⠋⢿⣿⣿⣿
⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠛⠋⠛⠙⣤⣤⠁⠀⠀⠀⠀⠀⠀⠀⠀⠉
⠄⣤⣤⡤⠉⠁⠛⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⣠⣶⠏⠉⣶⣤⣤⣤⣤⣤⣤⣤⣤⣤⣤⣤⣶⣶
⣶⣶⣶⣶⠉⠉⣶⣶⠁⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠀⣿⣿⣶⣶⣶⣶⣶⣶⣶⣶⣶⣶⣶⣶⣶⣶⣶⣶
⣤⣤⣤⣤⣤⣶⣿⣿⠁⠀⡀⠈⠛⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠿⢁⣶⣤⣿⣿⢿⠋⠋⠉⠙⠉⠉⠉⠉⠉⠉⠉⠛⡇
⡇⠉⠉⠀⠀⠘⣿⣿⣿⣿⣿⢻⣶⣶⣶⠉⠿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠛⣠⣰⠘⠉⣶⣶⣶⣶⣾⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
⠛⠛⠛⠛⠛⣶⣶⣶⣶⣶⣶⣶⣶⣶⣶⣶⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢿⢀⣀⣤⣴⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠛⠃⠛⠋⡀⣀⣀⣤⣠⣠⣤⣤⣶
⡏⠉⠉⠛⢸⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⠟⠉⣀⣀⣀⣼⣼⣷⣾⢷⣾⣷⣾⣿⣷⢷⣿
⣿⢿⣶⣶⣤⣤⣤⠉⠛⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
```



```
⣠⣈⠂⣸⢽⠀⢼⣟⣯⣷⣿⣟⣿⣿⣽⣿⣿⡷⡌⢣⡈⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡟⠁⡔⢁⡔⢡⣾⣿⣿⣧⡐⢿⠀⣞⠀⠀⣏⢦⡶⣞⣿⣽⣳⣱⣽⣿⢿⣻⣾⣴⣳
⣷⣿⠀⣺⡝⠀⣽⣿⢿⣻⣽⣿⣽⣾⡿⣯⣿⣿⣿⡀⢳⡀⠻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⠋⢀⠎⢠⠎⣠⣿⣿⣿⣿⣿⣿⣆⠀⡯⠀⠐⠧⠻⠛⠛⢋⢋⠃⠘⠙⠿⣿⢿⣾⡿⣿
⣽⡿⠀⣷⠃⠀⣿⡿⣿⡿⣿⣽⣯⣷⣿⣿⡿⣿⣻⡇⢀⠹⣆⠙⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿
";
    println!("{}", body);
}
```

Done. Now when you run `ai`, you will see ai.

```
$ cargo build
$ ./target/debug/ai
```

config

config

Add the code for bluesky's authentication system.

Specifically, write information in `~/.config/ai/config.toml` and create a command option to put authentication information in `~/.config/ai/token.toml`.

~/.config/ai/config.toml

```
handle = "yui.syui.ai"
pass = "xxx"
host = "bsky.social"
```

Cargo.toml

```
[package]
name = "ai"
version = "0.1.0"
edition = "2021"

[dependencies]
seahorse = "*"
reqwest = { version = "*", features = ["blocking", "json"] }
tokio = { version = "1", features = ["full"] }
serde_derive = "1.0"
serde_json = "1.0"
serde = "*"
config = { git = "https://github.com/mehcode/config-rs",
branch = "master" }
shellexpand = "*"
toml = "*"
```

src/data.rs

```
use config::{Config, ConfigError, File};
use serde_derive::{Deserialize, Serialize};

#[derive(Debug, Deserialize)]
#[allow(unused)]
pub struct Data {
    pub host: String,
    pub pass: String,
    pub handle: String,
}

#[derive(Serialize, Deserialize)]
#[allow(non_snake_case)]
pub struct Token {
    pub did: String,
    pub accessJwt: String,
    pub refreshJwt: String,
    pub handle: String,
}

#[derive(Serialize, Deserialize)]
#[allow(non_snake_case)]
pub struct Tokens {
    pub did: String,
    pub access: String,
    pub refresh: String,
    pub handle: String,
}
```

```
impl Data {
    pub fn new() -> Result<Self, ConfigError> {
        let d = shellexpand::tilde("~") +
"/.config/ai/config.toml";
        let s = Config::builder()
            .add_source(File::with_name(&d))

.add_source(config::Environment::with_prefix("APP"))
            .build()?;
        s.try_deserialize()
    }
}
```

src/main.rs

```
pub mod data;
use seahorse::{App, Context, Command};
use std::env;
use std::fs;
use std::io::Write;
use std::collections::HashMap;

use data::Data as Datas;
use crate::data::Token;
use crate::data::Tokens;

fn main() {
    let args: Vec<String> = env::args().collect();
    let app = App::new(env!("CARGO_PKG_NAME"))
        //.action(c_ascii_art)
        .command(
                Command::new("bluesky")
                .alias("b")
                .action(c_list_records),
                )
        .command(
                Command::new("login")
                .alias("l")
                .action(c_access_token),
                )

        ;
        app.run(args);
}

#[tokio::main]
async fn list_records() -> reqwest::Result<()> {
    let client = reqwest::Client::new();
    let handle= "support.bsky.team";
    let col = "app.bsky.feed.post";
    let body =
client.get("https://bsky.social/xrpc/com.atproto.repo.listRe
cords")
        .query(&[("repo", &handle),("collection", &col),
("limit", &"1"),("revert", &"true")])
        .send()
        .await?
        .text()
        .await?;
    println!("{}", body);
    Ok(())
}

fn c_list_records(_c: &Context) {
    list_records().unwrap();
```

```
}

#[tokio::main]
async fn access_token() -> reqwest::Result<()> {
    let file = "/.config/ai/token.toml";
    let mut f = shellexpand::tilde("~").to_string();
    f.push_str(&file);

    let data = Datas::new().unwrap();
    let data = Datas {
        host: data.host,
        handle: data.handle,
        pass: data.pass,
    };
    let url = "https://".to_owned() + &data.host +
&"/xrpc/com.atproto.server.createSession";

    let mut map = HashMap::new();
    map.insert("identifier", &data.handle);
    map.insert("password", &data.pass);

    let client = reqwest::Client::new();
    let res = client
        .post(url)
        .json(&map)
        .send()
        .await?
        .text()
        .await?;

    let json: Token = serde_json::from_str(&res).unwrap();
    let tokens = Tokens {
        did: json.did.to_string(),
        access: json.accessJwt.to_string(),
        refresh: json.refreshJwt.to_string(),
        handle: json.handle.to_string(),
    };

    let toml = toml::to_string(&tokens).unwrap();
    let mut f = fs::File::create(f.clone()).unwrap();
    f.write_all(&toml.as_bytes()).unwrap();

    Ok(())
}

fn c_access_token(_c: &Context) {
    access_token().unwrap();
}
```

mention

mention

Now it's time to create a command to `post` to bluesky. To be precise, it is `mention`.

Now, let's create a new file and read it in `src/main.rs`.

Cargo.toml

```
[package]
name = "ai"
version = "0.1.0"
edition = "2021"

[dependencies]
seahorse = "*"
reqwest = { version = "*", features = ["blocking", "json"] }
tokio = { version = "1", features = ["full"] }
serde_derive = "1.0"
serde_json = "1.0"
serde = "*"
config = { git = "https://github.com/mehcode/config-rs",
branch = "master" }
shellexpand = "*"
toml = "*"
iso8601-timestamp = "0.2.10"
```

src/data.rs

```
use config::{Config, ConfigError, File};
use serde_derive::{Deserialize, Serialize};

#[derive(Debug, Deserialize)]
#[allow(unused)]
pub struct Data {
    pub host: String,
    pub pass: String,
    pub handle: String,
}

#[derive(Serialize, Deserialize)]
#[allow(non_snake_case)]
pub struct Token {
    pub did: String,
    pub accessJwt: String,
    pub refreshJwt: String,
    pub handle: String,
}

#[derive(Serialize, Deserialize)]
#[allow(non_snake_case)]
pub struct Tokens {
    pub did: String,
    pub access: String,
    pub refresh: String,
    pub handle: String,
}

#[derive(Serialize, Deserialize)]
#[allow(non_snake_case)]
pub struct Labels {
}
```

```
#[derive(Serialize, Deserialize)]
#[allow(non_snake_case)]
pub struct Declaration {
    pub actorType: String,
    pub cid: String,
}

#[derive(Serialize, Deserialize)]
#[allow(non_snake_case)]
pub struct Viewer {
    pub muted: bool,
}

#[derive(Serialize, Deserialize)]
#[allow(non_snake_case)]
pub struct Profile {
    pub did: String,
    pub handle: String,
    pub followsCount: Option<i32>,
    pub followersCount: Option<i32>,
    pub postsCount: i32,
    pub indexedAt: Option<String>,
    pub avatar: Option<String>,
    pub banner: Option<String>,
    pub displayName: Option<String>,
    pub description: Option<String>,
    pub viewer: Viewer,
    pub labels: Labels,
}

impl Data {
    pub fn new() -> Result<Self, ConfigError> {
        let d = shellexpand::tilde("~") +
"/.config/ai/config.toml";
        let s = Config::builder()
            .add_source(File::with_name(&d))

.add_source(config::Environment::with_prefix("APP"))
            .build()?;
        s.try_deserialize()
    }
}

impl Tokens {
    pub fn new() -> Result<Self, ConfigError> {
        let d = shellexpand::tilde("~") +
"/.config/ai/token.toml";
        let s = Config::builder()
            .add_source(File::with_name(&d))

.add_source(config::Environment::with_prefix("APP"))
            .build()?;
        s.try_deserialize()
    }
}

pub fn token_toml(s: &str) -> String {
    let s = String::from(s);
    let tokens = Tokens::new().unwrap();
    let tokens = Tokens {
        did: tokens.did,
        access: tokens.access,
        refresh: tokens.refresh,
        handle: tokens.handle,
    };
    match &*s {
```

```
        "did" => tokens.did,
        "access" => tokens.access,
        "refresh" => tokens.refresh,
        "handle" => tokens.handle,
        _ => s,
    }
}
```

src/profile.rs

```
extern crate reqwest;
use crate::token_toml;

pub async fn get_request(handle: String) -> String {

    let token = token_toml(&"access");
    let url =
"https://bsky.social/xrpc/app.bsky.actor.getProfile".to_owne
d() + &"?actor=" + &handle;
    let client = reqwest::Client::new();
    let res = client
        .get(url)
        .header("Authorization", "Bearer ".to_owned() +
&token)
        .send()
        .await
        .unwrap()
        .text()
        .await
        .unwrap();

    return res
}
```

src/mention.rs

```
extern crate reqwest;
use crate::token_toml;
use serde_json::json;
use iso8601_timestamp::Timestamp;

pub async fn post_request(text: String, at: String, udid:
String, s: i32, e: i32) -> String {

    let token = token_toml(&"access");
    let did = token_toml(&"did");
    let handle = token_toml(&"handle");

    let url =
"https://bsky.social/xrpc/com.atproto.repo.createRecord";
    let col = "app.bsky.feed.post".to_string();

    let d = Timestamp::now_utc();
    let d = d.to_string();

    let post = Some(json!({
        "did": did.to_string(),
        "repo": handle.to_string(),
        "collection": col.to_string(),
        "record": {
            "text": at.to_string() + &" ".to_string() +
&text.to_string(),
            "$type": "app.bsky.feed.post",
            "createdAt": d.to_string(),
            "facets": [
            {
```

```
                    "$type": "app.bsky.richtext.facet",
                    "index": {
                        "byteEnd": e,
                        "byteStart": s
                    },"features": [
                    {
                        "did": udid.to_string(),
                        "$type":
"app.bsky.richtext.facet#mention"
                    }
                    ]
                }
                ]
            },
        }));

        let client = reqwest::Client::new();
        let res = client
            .post(url)
            .json(&post)
            .header("Authorization", "Bearer ".to_owned() +
&token)
            .send()
            .await
            .unwrap()
            .text()
            .await
            .unwrap();

        return res
}
```

src/main.rs

```
pub mod data;
pub mod mention;
pub mod profile;

use seahorse::{App, Command, Context, Flag, FlagType};
use std::env;
use std::fs;
use std::io::Write;
use std::collections::HashMap;

use data::Data as Datas;
use crate::data::Token;
use crate::data::Tokens;
use crate::data::Profile;
use crate::data::token_toml;

fn main() {
    let args: Vec<String> = env::args().collect();
    let app = App::new(env!("CARGO_PKG_NAME"))
        //.action(c_ascii_art)
        .command(
            Command::new("bluesky")
            .alias("b")
            .action(c_list_records),
            )
        .command(
            Command::new("login")
            .alias("l")
            .action(c_access_token),
            )
        .command(
            Command::new("profile")
```

```
            .alias("p")
            .action(c_profile),
            )
        .command(
            Command::new("mention")
            .alias("m")
            .action(c_mention)
            .flag(
                Flag::new("post", FlagType::String)
                .description("post flag\n\t\t\t$ ai m
syui.bsky.social -p text")
                .alias("p"),
                )
            )

        ;
    app.run(args);
}

#[tokio::main]
async fn list_records() -> reqwest::Result<()> {
    let client = reqwest::Client::new();
    let handle= "support.bsky.team";
    let col = "app.bsky.feed.post";
    let body =
client.get("https://bsky.social/xrpc/com.atproto.repo.listRe
cords")
        .query(&[("repo", &handle),("collection", &col),
("limit", &"1"),("revert", &"true")])
        .send()
        .await?
        .text()
        .await?;
    println!("{}", body);
    Ok(())
}

fn c_list_records(_c: &Context) {
    list_records().unwrap();
}

#[tokio::main]
async fn access_token() -> reqwest::Result<()> {
    let file = "/.config/ai/token.toml";
    let mut f = shellexpand::tilde("~").to_string();
    f.push_str(&file);

    let data = Datas::new().unwrap();
    let data = Datas {
        host: data.host,
        handle: data.handle,
        pass: data.pass,
    };
    let url = "https://".to_owned() + &data.host +
&"/xrpc/com.atproto.server.createSession";

    let mut map = HashMap::new();
    map.insert("identifier", &data.handle);
    map.insert("password", &data.pass);
    let client = reqwest::Client::new();
    let res = client
        .post(url)
        .json(&map)
        .send()
        .await?
        .text()
```

```
        .await?;
    let json: Token = serde_json::from_str(&res).unwrap();
    let tokens = Tokens {
        did: json.did.to_string(),
        access: json.accessJwt.to_string(),
        refresh: json.refreshJwt.to_string(),
        handle: json.handle.to_string(),
    };
    let toml = toml::to_string(&tokens).unwrap();
    let mut f = fs::File::create(f.clone()).unwrap();
    f.write_all(&toml.as_bytes()).unwrap();

    Ok(())
}

fn c_access_token(_c: &Context) {
    access_token().unwrap();
}

fn profile(c: &Context) {
    let m = c.args[0].to_string();
    let h = async {
        let str = profile::get_request(m.to_string()).await;
        println!("{}",str);
    };
    let res =
tokio::runtime::Runtime::new().unwrap().block_on(h);
    return res
}

fn c_profile(c: &Context) {
    access_token().unwrap();
    profile(c);
}

fn mention(c: &Context) {
    let m = c.args[0].to_string();
    let h = async {
        let str = profile::get_request(m.to_string()).await;
        println!("{}",str);
        let profile: Profile =
serde_json::from_str(&str).unwrap();
        let udid = profile.did;
        let handle = profile.handle;
        let at = "@".to_owned() + &handle;
        let e = at.chars().count();
        let s = 0;
        if let Ok(post) = c.string_flag("post") {
            let str =
mention::post_request(post.to_string(), at.to_string(),
udid.to_string(), s, e.try_into().unwrap()).await;
            println!("{}",str);
        }
    };
    let res =
tokio::runtime::Runtime::new().unwrap().block_on(h);
    return res
}

fn c_mention(c: &Context) {
    access_token().unwrap();
    mention(c);
}
```

This time, we don't support any hosts other than `bsky.social` because it is troublesome. Mainly `profile.rs` and `mention.rs`. Please be careful about that.

src/profile.rs

```
let url =
"https://bsky.social/xrpc/app.bsky.actor.getProfile".to_owne
d() + &"?actor=" + &handle;
```

base64

base64

Next, write the code to convert the characters specified in the command options to base64 and `mention`.

This completes the program.

First, add the `base64` package.

Cargo.toml

```
[package]
name = "ai"
version = "0.1.0"
edition = "2021"

[dependencies]
seahorse = "*"
reqwest = { version = "*", features = ["blocking", "json"] }
tokio = { version = "1", features = ["full"] }
serde_derive = "1.0"
serde_json = "1.0"
serde = "*"
config = { git = "https://github.com/mehcode/config-rs",
branch = "master" }
shellexpand = "*"
toml = "*"
iso8601-timestamp = "0.2.10"
base64 = "*"
```

Then, in `src/main.rs`, in the `mention`, write the code to convert the `did` to base64.

These are set to suboptions.

Here is a summary of the main points.

example

```
.command(
        Command::new("mention")
        .alias("m")
        .action(c_mention)
        .flag(
            Flag::new("base", FlagType::String)
            .description("base flag\n\t\t\t$ ai m
syui.bsky.social -p text -b 123")
            .alias("b"),
            )
        .flag(
            Flag::new("egg", FlagType::Bool)
            .description("egg flag\n\t\t\t$ ai m
syui.bsky.social -e")
            .alias("e"),
            )

let did = token_toml(&"did");
let body = "/egg ".to_owned() + &encode(did.as_bytes());
```

Allow `-b` to specify the string to be converted. Be sure to enclose the string in double quotation marks, for example `-b "foo bar"`. Use `-e` to get the `did` and convert it automatically for mention.

``sh

convert the specified string to base64 for mention

$ ai m yui.syui.ai -b "did:plc:4hqjfn7m6n5hno3doamuhgef" @yui.syui.ai /egg
ZGlkOnBsYzo0aHFqZm43bTZuNWhubzNkb2FtdWhnZWY=

MENTION your did as base64

$ ai m yui.syui.ai -e @yui.syui.ai /egg
ZGlkOnBsYzo0aHFqZm43bTZuNWhubzNkb2FtdWhnZWY=

Now, let's write the whole code.

!FILENAME src/main.rs
```rust
pub mod data;
pub mod mention;
pub mod profile;
//pub mod ascii;

use seahorse::{App, Command, Context, Flag, FlagType};
use std::env;
use std::fs;
use std::io::Write;
use std::collections::HashMap;

use data::Data as Datas;
use crate::data::Token;
use crate::data::Tokens;
use crate::data::Profile;
use crate::data::token_toml;
//use crate::ascii::c_ascii;

extern crate base64;
use base64::encode;

fn main() {
    let args: Vec<String> = env::args().collect();
    let app = App::new(env!("CARGO_PKG_NAME"))
        //.action(c_ascii_art)
        .command(
            Command::new("bluesky")
            .alias("b")
            .action(c_list_records),
            )
        .command(
            Command::new("login")
            .alias("l")
            .action(c_access_token),
            )
        .command(
            Command::new("profile")
            .alias("p")
            .action(c_profile),
            )
        .command(
            Command::new("mention")
            .alias("m")
            .action(c_mention)
            .flag(
```

```
                Flag::new("post", FlagType::String)
                .description("post flag\n\t\t\t$ ai m
syui.bsky.social -p text")
                .alias("p"),
                )
            .flag(
                Flag::new("base", FlagType::String)
                .description("base flag\n\t\t\t$ ai m
syui.bsky.social -p text -b 123")
                .alias("b"),
                )
            .flag(
                Flag::new("egg", FlagType::Bool)
                .description("egg flag\n\t\t\t$ ai m
syui.bsky.social -e")
                .alias("e"),
                )
            )

        ;
    app.run(args);
}

#[tokio::main]
async fn list_records() -> reqwest::Result<()> {
    let client = reqwest::Client::new();
    let handle= "support.bsky.team";
    let col = "app.bsky.feed.post";
    let body =
client.get("https://bsky.social/xrpc/com.atproto.repo.listRe
cords")
        .query(&[("repo", &handle),("collection", &col),
("limit", &"1"),("revert", &"true")])
        .send()
        .await?
        .text()
        .await?;
    println!("{}", body);
    Ok(())
}

fn c_list_records(_c: &Context) {
    list_records().unwrap();
}

#[tokio::main]
async fn access_token() -> reqwest::Result<()> {
    let file = "/.config/ai/token.toml";
    let mut f = shellexpand::tilde("~").to_string();
    f.push_str(&file);

    let data = Datas::new().unwrap();
    let data = Datas {
        host: data.host,
        handle: data.handle,
        pass: data.pass,
    };
    let url = "https://".to_owned() + &data.host +
&"/xrpc/com.atproto.server.createSession";

    let mut map = HashMap::new();
    map.insert("identifier", &data.handle);
    map.insert("password", &data.pass);
    let client = reqwest::Client::new();
    let res = client
        .post(url)
```

```
            .json(&map)
            .send()
            .await?
            .text()
            .await?;
        let json: Token = serde_json::from_str(&res).unwrap();
        let tokens = Tokens {
            did: json.did.to_string(),
            access: json.accessJwt.to_string(),
            refresh: json.refreshJwt.to_string(),
            handle: json.handle.to_string(),
        };
        let toml = toml::to_string(&tokens).unwrap();
        let mut f = fs::File::create(f.clone()).unwrap();
        f.write_all(&toml.as_bytes()).unwrap();

        Ok(())
    }

    fn c_access_token(_c: &Context) {
        access_token().unwrap();
    }

    fn profile(c: &Context) {
        let m = c.args[0].to_string();
        let h = async {
            let str = profile::get_request(m.to_string()).await;
            println!("{}",str);
        };
        let res =
    tokio::runtime::Runtime::new().unwrap().block_on(h);
        return res
    }

    fn c_profile(c: &Context) {
        access_token().unwrap();
        profile(c);
    }

    fn mention(c: &Context) {
        let m = c.args[0].to_string();
        let h = async {
            let str = profile::get_request(m.to_string()).await;
            let profile: Profile =
    serde_json::from_str(&str).unwrap();
            let udid = profile.did;
            let handle = profile.handle;
            let at = "@".to_owned() + &handle;
            let e = at.chars().count();
            let s = 0;
            if let Ok(base) = c.string_flag("base") {
                let body = "/egg ".to_owned() +
    &encode(base.as_bytes());
                let str =
    mention::post_request(body.to_string(), at.to_string(),
    udid.to_string(), s, e.try_into().unwrap()).await;
                println!("{}",str);
            }
            if let Ok(post) = c.string_flag("post") {

                let str =
    mention::post_request(post.to_string(), at.to_string(),
    udid.to_string(), s, e.try_into().unwrap()).await;
                println!("{}",str);
            }
            if c.bool_flag("egg") {
```

```
            let did = token_toml(&"did");
            let body = "/egg ".to_owned() +
&encode(did.as_bytes());
            println!("{}", body);
            let str =
mention::post_request(body.to_string(), at.to_string(),
udid.to_string(), s, e.try_into().unwrap()).await;
            println!("{}",str);
        }
    };
    let res =
tokio::runtime::Runtime::new().unwrap().block_on(h);
    return res
}

fn c_mention(c: &Context) {
    access_token().unwrap();
    mention(c);
}

//fn c_ascii_art(_c: &Context) {
//    c_ascii();
//}
```

```
cargo build
```

Done.

Now, if you specify `yui.syui.ai` as the `mention` and use the `-e` option, it will automatically convert your did to base64 and send it to you.

```
./target/debug/ai m yui.syui.ai -e
```

However, this makes it difficult to execute the command.

In order to be able to run this command from anywhere, we will put `binary`, i.e., `.ai`, which we can do when we `cargo build`. /target/debug/aiin$PATH`.

linux

```
$ echo $PATH|tr : '\n'
/usr/bin
/usr/local/bin

$ sudo cp -rf ./target/debug/ai /usr/local/bin/
$ ai -h

Name:
        ai
Flags:
        -h, --help : Show help
Commands:
        b, bluesky :
        l, login   :
        p, profile :
        m, mention :
```

windows

```
$ENV:Path.Split(";")
C:\Users\syui\scoop\apps\rust\current\bin

cp ~/scoop/rust/current/bin/
ai -h
```

Let's play around with making your own commands with `rust` like this.

end

end

I would like to end with a sentence.

Continuing is precious.

It is not possible to do everything from the beginning.

Even if you can't do it, even if you don't understand it, by continuing, you will grow.

However, it is not easy to keep going.

I have been doing this for a year. You did it.

"....."

Maybe no one will say anything.

"It's been two years. You did great.

"....."

Maybe no one will praise you.

"It lasted three years. It was hard work.

"....."

"...lasted five years. It lasted five years. There were hard times, sad times.

"....."

But can you keep going?

It's okay if you can't continue.

But it is precious to be able to continue.

If you would like, please try your best.

I hope this text can give you some courage.

!