

Programming Languages

OCaml Setting Manual

김재호

OCaml Setting

- PL 스터디에서 사용하는 OCaml을 배워보기도 전에 개발환경 세팅에서 너무 많은 시간과 노력이 들어가게 되어 뒤늦게나마 문서화하는 OCaml Setting Manual
- 급하게 만들어 문제가 있을 수 있습니다. 이미 세팅을 완료한 분들은 피드백 등의 도움 주시면 감사하겠습니다.

OCaml Setting

- OCaml 개발환경을 세팅하기 위해서는, 일단 세 가지 파트로 나누어 생각해야 한다
 - Part 1. OPAM (Ocam! PAckage Manager) 설치를 위한 터미널 작업
 - Part 2. 주 사용 에디터인 VS Code 설치 및 세팅
 - Part 3. 과제 레포에 있는 build.sh를 이용한 OCaml 컴파일러 및 라이브러리 세팅
- 각 파트는 어느 정도 독립적이지만, 순서대로 진행해 주시기 바라며 특히 Windows와 Mac의 세팅 방법이 다르기 때문에 본인 OS에 맞는 과정을 따라 주시기 바랍니다.
- 파트 3은 Windows와 Mac 공통으로 작성되어 있습니다.

Part 1 - for Windows

Part 1. OPAM 설치를 위한 터미널 작업

for Windows

- 일단, Windows에서도 리눅스와 거의 같은 파일시스템 및 CLI 환경을 갖춰야 이후 여러 상황에서 OS에 관계없이 공통적으로 진행할 수 있기 때문에, WSL을 설치해 리눅스 환경을 갖추도록 하자
- 자세한 설명은 공식 문서 페이지인 <https://docs.microsoft.com/ko-kr/windows/wsl/install> 에 잘 나와 있으니, 해당 매뉴얼을 따라가다 문제가 생기면 링크를 타고 들어가자

Part 1. OPAM 설치를 위한 터미널 작업

for Windows

- 이제 WSL을 설치하기 위해, 명령 프롬프트를 관리자 권한으로 실행한다
- 명령 프롬프트 대신 PowerShell을 사용해도 좋다
- 명령 프롬프트가 켜지면, 다음 커맨드를 실행한다 (앞으로 커맨드를 실행한다는 것은, 프롬프트 또는 터미널 창에 커맨드를 그대로 입력한 뒤 엔터를 친다는 의미로 받아들이자)
- `wsl --install`
- 위 커맨드를 실행하면, 뭔가 복잡한 메시지들이 출력되며 설치가 진행될 것이다
- 완료되면, 당신의 노트북에는 WSL 및 특정 버전의 Ubuntu라는 OS가 설치된 것이다

Part 1. OPAM 설치를 위한 터미널 작업

for Windows

- 이제 메시지에 뜬 대로 (아무것도 건들지 말고) PC를 다시시작하고 나면, 아마 명령 프롬프트가 다시 켜지며 마저 설치를 진행할 것이다
- 여기까지 마치면, Ubuntu가 실행되며 사용자 id와 password를 지정하라 한다
- 이 때 id는 자신이 자주 쓰는 영어(+ 숫자) id로 아무렇게나 정해서 엔터 치면 되고, password 역시 자신이 자주 쓰는 비밀번호를 입력하여 엔터치면 되는데, 이 때 주의할 점은 우분투의 보안 정책 때문에 password를 입력할 때에는 입력이 되고 있음에도 화면에 아무런 변화가 없을 것이다. 입력이 되고 있을 것이라 믿고 정확히 입력해 주도록 하자. 어차피 한 번 더 입력하도록 시키기 때문에 오타가 났다면 다시 입력할 기회가 있을 것이다
- 여기까지 완료했다면, 이제 여러분은 Windows 안에서 Ubuntu를 사용할 수 있다

Part 1. OPAM 설치를 위한 터미널 작업

for Windows

- 이제 다음 커맨드를 순서대로 실행하자. 각각 우분투의 패키지 매니저 시스템에서 사용하는 저장소를 갱신하고, OPAM을 설치하는 커맨드이다. 아마 sudo라는 커맨드 때문에 password를 요구할 텐데 조금 전 정해진 password를 입력해 주면 된다. 이 때에도 입력하는 게 화면에 보이지 않으니 당황하지 말자. 또한 아래 커맨드 실행중 뭔가를 물어볼 수도 있는데 “이대로 설치할까요?” 라고 묻는 것이니 그냥 엔터 쳐주면 된다
- `sudo apt update`
- `sudo apt install opam`

Part 1. OPAM 설치를 위한 터미널 작업

for Windows

- 여기서 잠깐,
- 혹시 사용자 id와 비밀번호를 등록하는 절차가 의도치 않게 생략되었거나, 잘못된 비밀번호를 등록한 것 같아 sudo로 시작하는 커맨드에서 아무리 맞는 비밀번호를 입력해도 틀렸다고 한다면, 아래 링크로 들어가 “Linux 배포용 암호를 잊은 경우 다음을 수행합니다” 부터 따라가보자
- <https://docs.microsoft.com/ko-kr/windows/wsl/setup/environment#set-up-your-linux-username-and-password>

Part 1. OPAM 설치를 위한 터미널 작업

for Windows

- OPAM이 잘 설치되었는지 확인하기 위해, 다음 커맨드를 실행한다
- `opam --version`
- 잘 설치되었다면, 아마 2.0.X 버전이 설치되었기 때문에 2.0.X라고 뜰 것이다
- Part 1 끝!

Part 1 - for Mac

Part 1. OPAM 설치를 위한 터미널 작업

for Mac

- Mac은 이미 파일시스템 및 커맨드 구조가 Linux와 거의 비슷하게 되어 있기 때문에, WSL 같은 걸 따로 설치할 필요는 없지만 다음의 두 툴을 설치해야 한다
 - X code에서 제공하는 Command Line Tools
 - Mac의 패키지 관리 툴 (즉 ubuntu에서 쓰는 apt 같은 것) Homebrew
- 차근차근히 설치해 보자

Part 1. OPAM 설치를 위한 터미널 작업

for Mac

- 터미널을 실행한다
- 다음의 커맨드를 실행한다
- `xcode-select --install`
- 팝업 창이 뜨면, 설치 버튼을 클릭해 준다. 이후 동의 버튼을 클릭해 설치를 진행한다
- 우리가 특히 사용해야 하는 게 git이기 때문에, 다음 커맨드를 실행하여 설치가 잘 됐는지 확인한다
- `git --version`

Part 1. OPAM 설치를 위한 터미널 작업

for Mac

- 이제 homebrew를 설치하자
- 다음의 커맨드를 순서대로 실행한다
- `cd ~`
- `/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"`
- 설치는 완료 되었고, 다음 커맨드를 실행해 설치 확인 겸 저장소 갱신을 해준다
- `brew update`

Part 1. OPAM 설치를 위한 터미널 작업

for Mac

- 이제 OPAM 설치를 위해 다음 커맨드를 실행해 주도록 한다
- `brew install opam`
- 잘 설치가 되었다면, 다음 커맨드를 실행했을 때 2.0.X라고 뜰 것이다
- `opam --version`
- Part 1 끝!

Part 2 - for Windows


Part 2. VS Code 설치 및 세팅

for Windows

- 이젠 프로그래밍을 위한 VS Code를 설치하고, 여러 extension을 미리 설치할 차례이다
- 아래 링크에 접속해 Windows 버전 다운로드를 클릭하고, 설치를 진행하면 된다
- <https://code.visualstudio.com/download>
- 아마 설치 과정에서 어려운 일은 없을 것이다


Part 2. VS Code 설치 및 세팅

for Windows

- 설치 후 실행해 보면, 맨 왼쪽에 (5개 정도의) 아이콘으로 된 탭들이 보일 것이다
- 이 중 대략  이런 식으로 생긴 아이콘을 클릭한다. 이 탭은 VS Code에 날개를 달아줄 extension을 검색하고 설치 및 관리할 수 있는 탭이다
- 여기의 검색창에 “WSL”을 입력하면, 맨 위에 Microsoft에서 만들었다는 듯한 “Remote - WSL” extension이 보일 것이다. 설치(install) 버튼을 클릭하여 설치한다

Part 2. VS Code 설치 및 세팅

for Windows

- 설치가 완료되었다면, VS Code 창의 맨 왼쪽 아래 구석에 >< 같은 게 적혀 있는 칸이 생겼을 것이다. 이것을 클릭한다
- 클릭하면 가운데 위쪽으로 뭔가 명령을 기다리는 듯한 창이 뜨는데, “새 WSL 창 열기” 비스무리한 내용을 하고 있는 메뉴를 클릭한다
- 이제 새로운 창이 열릴텐데, 이 창은 여러분이 설치한 Ubuntu 파일 시스템 내부에 접근한 에디터이며, 일종의 서버에 연결했다고 생각해도 좋다
- 이제 다시  를 클릭한 뒤, 검색창에 “OCaml Platform”이라 검색하여 OCaml Labs라는 곳에서 만들었다는 “OCaml Platform” extension을 설치한다. 아마 아까와 달리 install for WSL 비스무리한 버튼으로 뜰텐데, 클릭하면 된다
- Part 2 끝!

Part 2 - for Mac


Part 2. VS Code 설치 및 세팅

for Mac

- 이젠 프로그래밍을 위한 VS Code를 설치하고, 여러 extension을 미리 설치할 차례이다
- 아래 링크에 접속해 Mac Universal 버전 다운로드를 클릭하고, 설치를 진행하면 된다
- <https://code.visualstudio.com/download>
- 아마 설치 과정에서 어려운 일은 없을 것이다

Part 2. VS Code 설치 및 세팅

for Windows

- 설치 후 실행해 보면, 맨 왼쪽에 (5개 정도의) 아이콘으로 된 탭들이 보일 것이다
- 이 중 대략  이런 식으로 생긴 아이콘을 클릭한다. 이 탭은 VS Code에 날개를 달아줄 extension을 검색하고 설치 및 관리할 수 있는 탭이다
- 검색창에 “OCaml Platform”이라 검색하여 OCaml Labs라는 곳에서 만들었다는 “OCaml Platform” extension을 설치한다.
- Part 2 끝!

Part 3 - for All

Part 3. build.sh를 이용한 OCaml 환경 세팅

- 이제 마지막 세팅을 해야 할 차례인데, 원래라면 OPAM의 여러 커맨드를 직접 한 땀 한 땀 실행해야 하지만, 그 모든 과정을 과제 레포 속 build.sh에 담아뒀다
- 그런데, 과제 레포를 받기 위해서는 Git을 다룰 수 있어야 하고, GitHub 계정이 필요하다!
- 일단 GitHub를 가입하자. 그냥 아래 링크로 들어가서 Sign Up 하고, 자주 쓰는 메일 계정으로 가입하면 된다
- <https://github.com>
- Git은 정상적으로 진행됐다는 가정 하에 터미널에 이미 설치되어 있을 것이다. 다음 커맨드를 실행해 확인해 보자
- `git --version`

Part 3. build.sh를 이용한 OCaml 환경 세팅

- GitHub가 작년 즈음부터 보안 정책 강화로 인해 git clone 등의 커맨드를 실행할 때 그냥 암호로 로그인하는 게 아니라, Personal Access Token이란 것으로 로그인하게 한다
- 그러므로 우리도 역시나 PAT를 만들어 두도록 하자
- GitHub 홈페이지에 다시 들어가서 로그인을 하면, 메인 화면이 뜰 것이고 오른쪽 위에 자신의 프사 같은 게 동그랗게 떠있을 것이다. 이를 클릭한 뒤 Settings를 클릭한다
- 이후 뜨는 세팅 페이지의 왼쪽 메뉴들 중 가장 아래의 Developer settings를 클릭한다
- 이후 뜨는 페이지의 왼쪽 메뉴들 중 Personal access tokens를 클릭한다

Part 3. build.sh를 이용한 OCaml 환경 세팅

- 잘 따라왔다면, 화면 중앙에서 오른쪽 즈음에 Generate new token 버튼이 보일 것이다
- 클릭한 뒤, note 부분에 아무렇게나 이름을 붙이고, Expiration은 보안을 위해 해당 토큰의 만료 기간을 설정하는 것인데, 우리는 귀찮으니 그냥 No expiration을 선택하자
- 그 아래 모든 체크박스에 체크한다. 상위 체크박스를 체크하면 그에 해당하는 하위 체크박스는 알아서 전체 체크되니 그리 오래 걸리지 않을 것이다
- 이후 맨 아래 Generate token 버튼을 클릭하면, 토큰이 만들어진다

Part 3. build.sh를 이용한 OCaml 환경 세팅

- **중요!!** 이 페이지를 벗어나면, 힘겹게 발급받은 토큰을 다시 확인할 방법은 전혀 없다!
- 그러므로 지금 그 페이지를 캡처해 두거나, 발급된 토큰을 복사해 어딘가에 저장해 두길 추천한다
- 아무튼, 발급된 복잡한 문자열 형태의 토큰을 복사해 둔다

Part 3. build.sh를 이용한 OCaml 환경 세팅

- 이제 정말 OCaml 세팅을 위해 assignment 레포를 받아올 차례이다
- 아래 링크의 README.md 파일의 스케줄 부분을 보면, 각 과제가 Assignment 부분에 링크되어 있음을 확인할 수 있을 것이다
- <https://github.com/oojahooo-classroom/Programming-Languages>
- 이 중 Assignment 1을 클릭한다
- 이제 Join the classroom 페이지가 뜰텐데, 자신의 이름을 클릭하면 이제 우리 classroom에 당신의 이름과 GitHub 계정이 연결된다
- 이후 Accept 어찌구 버튼을 클릭하면, 당신의 계정이 접근할 수 있는 과제 1 레포가 새로 생긴다!

Part 3. build.sh를 이용한 OCaml 환경 세팅

- 다시 GitHub 홈으로 돌아가면, 왼쪽 부분 당신의 repositories 목록에 oojahooo-classroom/pl-assignment-1-당신의아이디 레포가 생겼을 것이다
- 해당 레포를 클릭해 들어간다
- Code 버튼을 클릭한 뒤, 해당 레포의 주소를 복사해 준다
- 다시 터미널로 돌아가, 원하는 곳에 (모르겠으면 그냥 터미널 키자마자 나오는 화면에서)
- `git clone` 복사한레포주소
- 를 실행해 준다

Part 3. build.sh를 이용한 OCaml 환경 세팅

- 이제 드디어 GitHub 아이디와 비밀번호를 입력하라 할텐데, 아이디는 그냥 GitHub 아이디를 입력하고, 비밀번호에 전에 발급받은 personal access token를 붙여넣어 준다
- 뭔가 막 뜨고 마지막에 done이 뜨면 성공!
- 이제 앞으로 git에서 더이상 사용자 등록 및 로그인 관련 이슈가 생기지 않도록, 다음 커맨드를 순서대로 실행한다
- `git config --global user.name "your name"`
- `git config --global user.email "your email address"`
- `git config --global credential.helper store`

Part 3. build.sh를 이용한 OCaml 환경 세팅

- 이제 다시 작업중인 디렉토리에서, 방금 clone한 레포의 작업 공간으로 들어가 보자
- 다음 커맨드를 실행해서 디렉토리를 변경하면 된다
- `cd pl-assignment-어쩌구`
- 당연히 어쩌구를 그대로 입력하라는 게 아니라, 지금 각자의 레포 이름대로 폴더가 생성되었을 텐데, 그 폴더 이름을 입력하면 된다. 다 입력하기 귀찮으면 대충 pl-까지만 치고 탭을 쳐보라
- 이후 다음 커맨드를 실행해 build.sh의 스크립트를 실행하자
- `./build.sh`

Part 3. build.sh를 이용한 OCaml 환경 세팅

- build.sh 내용은 정확히 알 필요는 없지만, 대략 다음과 같은 일을 진행한다
 - opam의 초기 세팅을 진행하는 `opam init`을 실행
 - `opam switch`를 이용해 독립적인 OCaml 환경 추가 (이 말은, 하나의 컴퓨터의 여러 버전의 OCaml 컴파일러 환경을 세팅하거나, 여러 라이브러리를 설치해둔 환경을 만들어두고 그때그때 필요한 환경에서 작업이 가능하다는 뜻이다. 파이썬에도 `anaconda`가 비슷한 일을 해준다)
 - OCaml 컴파일러 (버전 4.14.0) 설치, 개발에 도움을 주는 OCaml 라이브러리들 설치
 - 이 때 설치되는 라이브러리들이 뭔지는 기회가 되면 설명하도록 하겠다

Part 3. build.sh를 이용한 OCaml 환경 세팅

- 조금은 오래 걸리겠지만, 아마 터무니없이 길지는 않은 시간 안에 전부 실행이 완료될 것이다. 이제 OCaml 개발을 위한 모든 설치는 끝이다!
- 이제 다시 VS Code를 들어가 보자
- Windows는 다시 왼쪽 아래 구석을 클릭해 WSL에서 새 창 열기를 해준다
- 이제 왼쪽 아이콘 탭들 중 가장 위에 있는 아이콘을 클릭하면 새 작업디렉토리를 열 수 있는 버튼이 있을 것이다. 이를 클릭한다
- WSL이라면 home/자신이설정한id이름/pl-assignment-어쩌구 폴더에 들어가 연다
- 맥이라면 자신이 작업하던 디렉토리의 pl-assignment-어쩌구 폴더에 들어가 연다

Part 3. build.sh를 이용한 OCaml 환경 세팅

- 폴더가 잘 열렸다면, 폴더의 src 폴더의 hw1.ml 파일을 클릭해 띄워 보자
- 대충 아래 사진처럼 코드가 알록달록해 보이고, pascal에 마우스를 올렸을 때 `int * int -> int`라고 뜨면 성공이다!

```
exception Not_implemented

(* Problem 1 *)
int * int → int
let pascal : int * int → int =
  fun (n, m) → raise Not_implemented (* IMPLEMENT HERE! *)
```

Part 3. build.sh를 이용한 OCaml 환경 세팅

- 만약 뭔가 제대로 안 된다면, 다음 두 가지를 진행해 보자
 - Ctrl + ` 를 입력하여 (는 한/영이 영어인 상태에서 ~~₩~~ 키로 입력 가능하다) 터미널을 새로 열어준 뒤, 터미널에서 `eval $(opam env)` 를 실행해 준다. 이 커맨드는 앞으로 터미널에서 OCaml 작업을 해줘야 할 때마다 맨 처음 실행해 줘야 하므로 기억해 두자
 - VS Code 맨 아래 상태표시줄처럼 보이는 곳에 `opam(pl-tutoring-4.14.0)` 비스무리하게 적힌 곳이 있을텐데 그곳을 클릭한 뒤 맨 위 `pl-tutoring-4.14.0`을 다시 클릭해 주자