미니쉘(minishell) 소개

5기들이여 죽지 말아요!!!

Kipark 9월 14일

목차

- 미니쉘이란?
- 미니쉘 구조
  - Parsing
  - Execute
  - Signal
  - Built-In

Parsing
파싱이란?

- 일련의 문자열을 의미있는 token(가장 작은 단위)로 분리하고 규칙에따라 **parse_tree**를 만드는 것
- 미니쉘의 파싱은 크게 2가지로 나눌 수 있다.
  - Lexer
    - 받아온 문자열을 가장 작은 단위인 token으로 분리하는 작업
  - Parser
    - token으로 분리된 문자열이 정상적인 문자열인지 확인하고 parse_tree에 넣어주는 작업

Parsing
Token이란?

- 토큰이란 쉘에서 사용하는 가장 작은 단위의 문자들을 말한다.
  - 예를들어 ls > a 라는 명령어가 있다면 토큰은 각각 다음과 같다.
    - ls - word
    - > - input_redirect
    - a - word
    - 대충 어떤 느낌인지 감이 잡히시죵?
  - 토큰은 일정한 규칙을 가지고 만들어진다. 알아야 할 규칙, 키워드가 몇 가지 있다.
    - IFS(Internal Field Separator)
    - Metacharter
      - **쉘에선 IFS와 Metacharter가 하나의 토큰으로 끊어주는 기준이 된다.**
    - Shell Grammar

Parsing

- 이부분 부터는 각자 미니쉘을 어떻게 작성 대한 생각이라 알아서 잘 기준을 잡고 구현하는걸로!

- **IFS**
  - **IFS는 문자열을 나눌 때 기준이 되는 문자를 정의하는 환경 변수이다.**
  - **공백, 개행, 탭 같은 문자들을 기준으로 자른다.**
    - **(E.d). if (c == ' ' || c == '\n' || c == '\t')**
    - **e.d**
      - **ls > a 는 => ls, >, a 로 값이 끊어진다.**
- **Metacharter**
  - **Metacharter란 쉘에서 이미 예약어로 지정된 문자들로 각각은 독립된 역할을 수행한다**
    - **$, |, >, >>, <, <<, &&, (), “, ‘**
    - **오른쪽 사진 확인!**

Symbol	Meaning
>	Output redirection
>>	Output redirection (append)
<	Input redirection
*	File substitution wildcard; zero or more characters
?	File substitution wildcard; one character
[ ]	File substitution wildcard; any character between brackets
`cmd`	Command Substitution
$(cmd)	Command Substitution
|	The Pipe (|)
;	Command sequence, Sequences of Commands
[ ]	File substitution wildcard; any character between brackets
||	OR conditional execution
&&	AND conditional execution
( )	Group commands, Sequences of Commands
&	Run command in the background, Background Processes
#	Comment
$	Expand the value of a variable
\	Prevent or escape interpretation of the next character
<<	Input redirection

Parsing

- Shell Grammar
  - 쉘은 우리가 사용하는 언어체계처럼 독립된 문법 구조를 가지고 작동된다.
  - 단어 하나하나가 주어, 서술어, 목적어, 보어가 되어 하나의 문장을 만드는것 처럼 쉘도 토큰 하나하나가 모여서 하나의 해석, 실행 가능한 문장으로 완성된다.

```
%token  WORD
%token  ASSIGNMENT_WORD
%token  NAME
%token  NEWLINE
%token  IO_NUMBER


/* The following are the operators mentioned above. */


%token  AND_IF    OR_IF    DSEMI
/*      '&&'      '||'     ';;'    */


%token  DLESS  DGREAT  LESSAND  GREATAND  LESSGREAT  DLESSDASH
/*      '<<'   '>>'    '<&'     '>&'      '<>'       '<<-'   */


%token  CLOBBER
/*      '>|'   */


/* The following are the reserved words. */


%token  If    Then    Else    Elif    Fi    Do    Done
/*      'if'  'then'  'else'  'elif'  'fi'  'do'  'done'   */


%token  Case    Esac    While    Until    For
/*      'case'  'esac'  'while'  'until'  'for'   */


/* These are reserved words, not operator tokens, and are
   recognized when reserved words are recognized. */


%token  Lbrace    Rbrace    Bang
/*      '{'       '}'       '!'   */


%token  In
/*      'in'   */
```

Parsing

- Shell Grammar
  - 실제 bash Grammar 에서 minishell에 필요한 부분만 분리하여서 정리한 BNF표다! 정리가 굉장히 잘 되어있는 블로그니 꼭 보시면 좋을 것 같다.
  - 하나씩 확인해보면 token들이 모여서 하나의 특정한 부분을 가지고 있다.

자료 출처 https://epicarts.tistory.com/163

```
<pipeline>      ::= <cmd>
                |   <pipeline> '|' <cmd>

<cmd>           ::= <simple_cmd>
                |   <simple_cmd> <redirects>

<simple_cmd>    ::= <file_path>
                |   <argv>

<argv>          ::= <file_path> <args>

<redirects>     ::= <io_redirect>
                  | <redirects> <io_redirect>

<io_redirect>   ::= '<'   <filename>
                  | '<<'  <filename>
                  | '>'   <filename>
                  | '>>'  <filename>

<args>          ::= WORD
                  | <args> WORD

<filename>      ::= WORD

<file_path>     ::= WORD
```

Parsing

- BNF표를 확인하면서 문장 해석
  - ls < a | cat -e | sort > test.txt
  - [word, redirect, word, pipe, word, word, pipe, word, redirect, word]
  - 이런 식으로 토큰을 분리하고 BNF 표 대로 하나씩 들어가면서 정리해보자
    - 한번 해보셔용
  - 이런식으로 문법에 대한 규칙을 정하면 어떤 토큰 뒤에 어떤 토큰이 들어 왔을 때 sytax에러가 난다고 알려줄 수 있다. 또한 기준을 세워둘 수 있으니 뭔가 공격이 들어와도 이거대로 만들었다 라고 애기할 수 있음

```
<pipeline>      ::= <cmd>
                |   <pipeline> '|' <cmd>

<cmd>           ::= <simple_cmd>
                |   <simple_cmd> <redirects>

<simple_cmd>    ::= <file_path>
                |   <argv>

<argv>          ::= <file_path> <args>

<redirects>     ::= <io_redirect>
                  | <redirects> <io_redirect>

<io_redirect>   ::= '<'   <filename>
                  | '<<'  <filename>
                  | '>'   <filename>
                  | '>>'  <filename>

<args>          ::= WORD
                  | <args> WORD

<filename>      ::= WORD

<file_path>     ::= WORD
```

Parsing

- 문법표대로 토큰을 나누고 나면 나눈 토큰들을 파스트리에 넣어주는 작업이 필요하다.
- 파스트리는 연결리스트, abs트리 같은 방식으로 구현할 수 있다. 멘데토리 파트에서는 노드를 나누는 기준이 |(pipe) 만 있기 때문에 연결리스트로 구현해도 큰 문제없이 진행할 수 있다 (아마?)
- 이부분은 알아서 잘 해보시길~
- 파스트리까지 넣는 작업이 완료되면 파싱이 끝나게 된다!

```
<pipeline>     ::= <cmd>
                |   <pipeline> '|' <cmd>

<cmd>          ::= <simple_cmd>
                |   <simple_cmd> <redirects>

<simple_cmd>   ::= <file_path>
                |   <argv>

<argv>         ::= <file_path> <args>

<redirects>    ::= <io_redirect>
                 | <redirects> <io_redirect>

<io_redirect>  ::= '<'   <filename>
                 | '<<'  <filename>
                 | '>'   <filename>
                 | '>>'  <filename>

<args>         ::= WORD
                 | <args> WORD

<filename>     ::= WORD

<file_path>    ::= WORD
```

참고 문서 & 블로그

IFS란 무엇인가?
https://inpa.tistory.com/entry/LINUX-%F0%9F%93%9A-IFSInternal-Field-Separator-%EC%95%8C%EA%B8%B0-%EC%89%BD%EA%B2%8C-%EC%A0%95%EB%A6%AC

쉘 Metacharter 란 무엇인가?
https://eunguru.tistory.com/91