Minishell Abstract Syntax Tree (AST) - Visual Guide

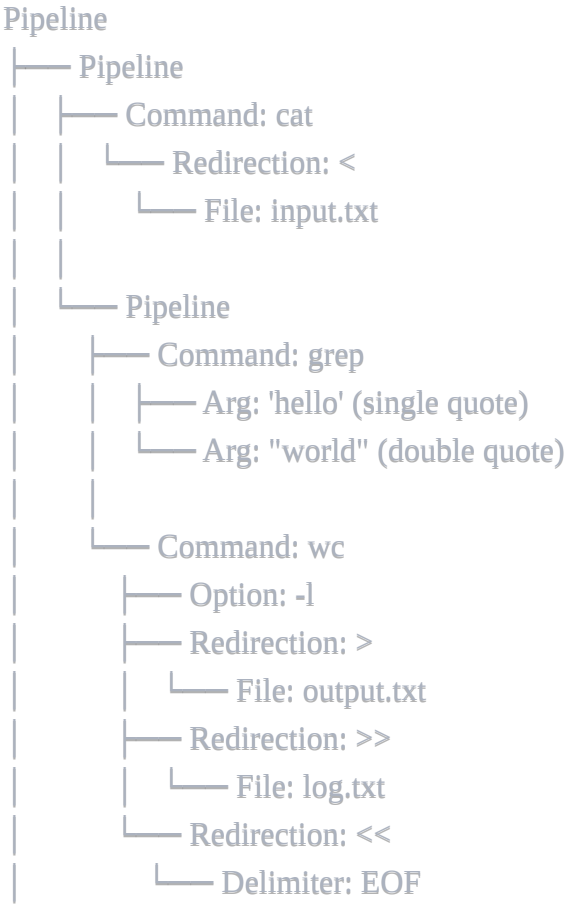
Complex Command Example

Let's visualize this complex command that uses all features:

bash

```
cat < input.txt | grep 'hello' "world" | wc -l > output.txt >> log.txt << EOF
```

Complete AST Structure



Detailed Node-by-Node Breakdown

Example 1: Simple Command with Redirections

bash

```
echo "Hello World" > output.txt
```

```
Command
├── name: "echo"
├── args:
│   └── StringLiteral(DOUBLE_QUOTE): "Hello World"
└── redirections:
    └── Redirection
        ├── type: REDIR_OUT (>)
        └── target: "output.txt"
```

Example 2: Pipeline with Multiple Commands

bash

```
ls -la | grep "test" | wc -l
```

```
Pipeline
├── left: Pipeline
│   ├── left: Command
│   │   ├── name: "ls"
│   │   └── args:
│   │       └── Option: "-la"
│   │
│   └── right: Command
│       ├── name: "grep"
│       └── args:
│           └── StringLiteral(DOUBLE_QUOTE): "test"
│
└── right: Command
    ├── name: "wc"
    └── args:
        └── Option: "-l"
```

Example 3: All Redirection Types

bash

```
cat < infile > outfile 2>> error.log << EOF
line1
line2
EOF
```

```
Command
├── name: "cat"
├── args: []
└── redirections:
    ├── Redirection
    │   ├── type: REDIR_IN (<)
    │   └── target: "infile"
    │
    ├── Redirection
    │   ├── type: REDIR_OUT (>)
    │   └── target: "outfile"
    │
    ├── Redirection
    │   ├── type: REDIR_APPEND (>>)
    │   ├── fd: 2
    │   └── target: "error.log"
    │
    └── Redirection
        ├── type: REDIR_HEREDOC (<<)
        ├── delimiter: "EOF"
        └── content: ["line1", "line2"]
```

Example 4: Mixed Quotes and Variables

bash

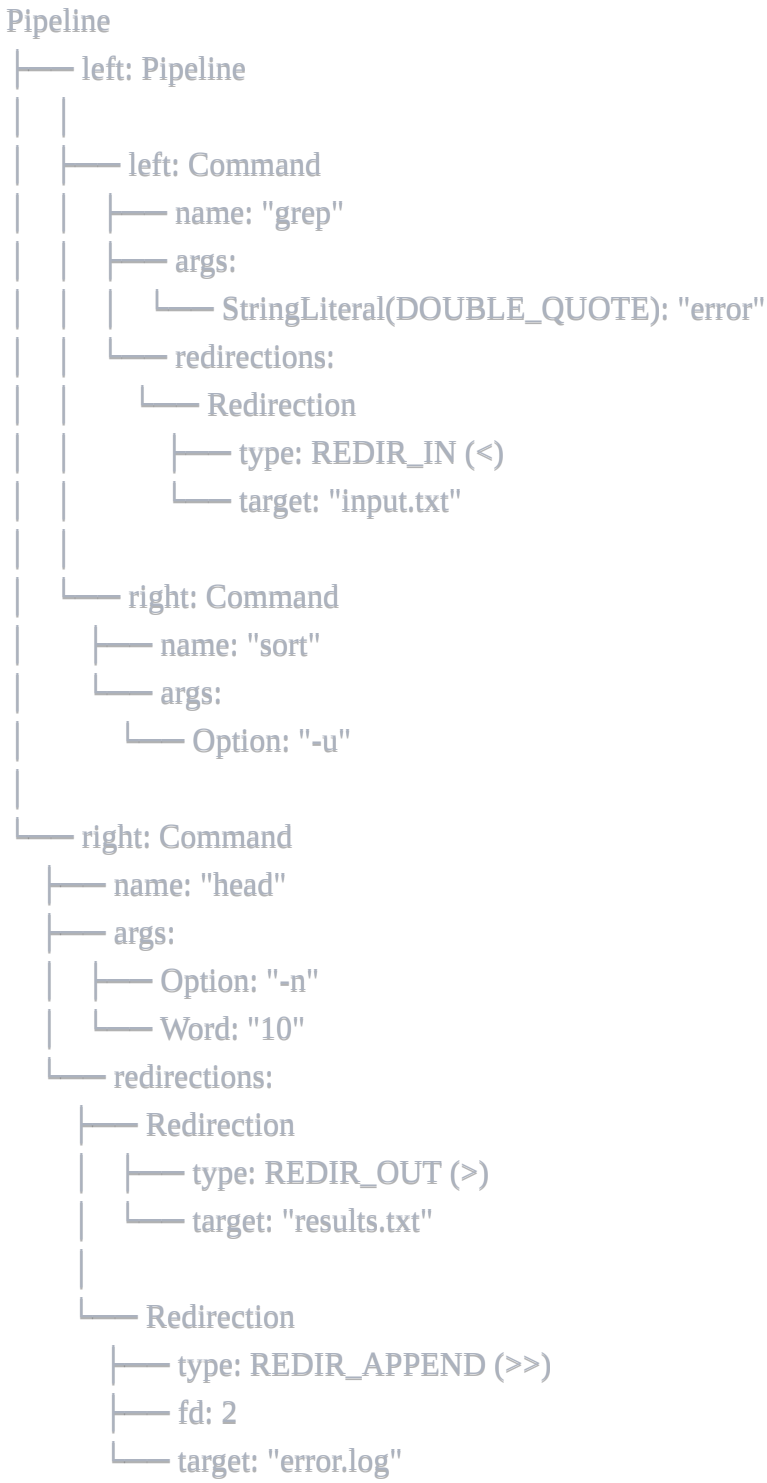
```
echo "Hello $USER" 'Single quotes' $HOME | cat -e
```

```
Pipeline
├── left: Command
│   ├── name: "echo"
│   └── args:
│       ├── StringLiteral(DOUBLE_QUOTE): "Hello $USER"
│       │   └── expansion: true
│       │
│       ├── StringLiteral(SINGLE_QUOTE): "Single quotes"
│       │   └── expansion: false
│       │
│       └── Variable: "$HOME"
│           └── expansion: true
│
└── right: Command
    ├── name: "cat"
    └── args:
        └── Option: "-e"
```

Example 5: Complex Real-World Example

bash

```
< input.txt grep "error" | sort -u | head -n 10 > results.txt 2>> error.log
```



Node Type Definitions

Key AST Node Types:

1. **Pipeline Node**
   - Connects two commands with `|`
   - Has `left` and `right` children
2. **Command Node**
   - Command name (string)
   - Array of arguments

    - Array of redirections
3. **Argument Types**
    - `Word`: Unquoted string
    - `StringLiteral(SINGLE_QUOTE)`: Single-quoted (no expansion)
    - `StringLiteral(DOUBLE_QUOTE)`: Double-quoted (with expansion)
    - `Variable`: Environment variable like `$HOME`
    - `Option`: Flag like `-l`, `-la`
4. **Redirection Node**
    - Type: `<`, `>`, `>>`, `<<`
    - Target file or delimiter
    - Optional file descriptor (for `2>`, `2>>`)

Visual Symbols Legend

```
├── : Branch (not last child)
└── : Branch (last child)
│   : Vertical line (continuation)
```

How to Read/Traverse the AST

Reading Direction: Top to Bottom, Left to Right

The AST is read in **execution order** using different traversal strategies:

Example: Reading a Simple Pipeline

bash

```
cat file.txt | grep "hello" | wc -l
```

```
Pipeline (Root)
├── left: Pipeline
│   ├── left: Command (cat file.txt)      ← Execute FIRST
│   └── right: Command (grep "hello")     ← Execute SECOND
└── right: Command (wc -l)                ← Execute THIRD
```

Reading Steps:

1. Start at root: "This is a Pipeline"
2. Go to `left` child: "Another Pipeline"
3. Go to that `left`: "Command: cat" - **EXECUTE**
4. Pipe output to `right`: "Command: grep" - **EXECUTE**
5. Return to root's `right`: "Command: wc" - **EXECUTE**

Execution Order: Post-Order Traversal for Pipelines

For pipelines, use **left-to-right execution**:

c

```c
void execute_pipeline(PipelineNode *pipe) {
    // Execute left side first
    execute_ast(pipe->left);

    // Connect pipe
    // Execute right side
    execute_ast(pipe->right);
}
```

Example: Reading Redirections

bash

```bash
cat < input.txt > output.txt
```

```
Command: cat
├── redirections[0]: < input.txt    ← Process FIRST (setup input)
└── redirections[1]: > output.txt   ← Process SECOND (setup output)
```

Reading Steps:

1. Identify command: "cat"
2. **Before executing**: Setup redirections in order
    - Redirection 0: Open `input.txt` for reading, dup to stdin
    - Redirection 1: Open `output.txt` for writing, dup to stdout
3. Execute the command

Example: Reading Heredoc

bash

```
cat << EOF | grep "line"
line1
line2
EOF
```

```
Pipeline
├── left: Command: cat
│   └── redirections:
│       └── Redirection: <<
│           ├── delimiter: "EOF"
│           └── content: ["line1", "line2"]
│
└── right: Command: grep
    └── args: ["line"]
```

Reading Steps:

1. See Pipeline - need to setup pipe between commands
2. Look at left command (cat)
3. See heredoc redirection
4. **Before executing cat**:
    - Create temp file or pipe
    - Write content lines to it

    - Redirect to stdin
5. Execute cat (reads from heredoc)
6. Pipe output to grep

Example: Complex Multi-Level Pipeline

bash

```
cmd1 | cmd2 | cmd3
```

```
Pipeline
├── left: Pipeline
│   ├── left: cmd1    ← Depth-first: Execute 1st
│   └── right: cmd2   ← Depth-first: Execute 2nd
└── right: cmd3       ← Execute 3rd
```

Traversal Algorithm:

```
1. Visit root Pipeline
2. Recurse into left child (another Pipeline)
   a. Recurse into its left (cmd1) - EXECUTE
   b. Recurse into its right (cmd2) - EXECUTE
3. Back to root, visit right child (cmd3) - EXECUTE
```

Reading Arguments with Quotes

bash

```
echo "Hello $USER" 'literal' word
```

```
Command: echo
└── args:
    ├── [0] StringLiteral(DOUBLE_QUOTE): "Hello $USER" (expand: true)
    ├── [1] StringLiteral(SINGLE_QUOTE): "literal" (expand: false)
    └── [2] Word: "word"
```

Reading Steps:

1. Identify command: "echo"
2. Process arguments in array order [0, 1, 2]:
    - Arg[0]: Double quotes → expand `$USER` variable
    - Arg[1]: Single quotes → keep literal "literal"
    - Arg[2]: Plain word → use as-is "word"
3. Execute: `echo "Hello john" literal word`

Traversal Pseudo-Code

Main Execution Function

c

```
void execute_ast(ASTNode *node) {
    if (!node)
        return;

    if (node->type == NODE_PIPELINE) {
        // Setup pipe
        int pipefd[2];
        pipe(pipefd);

        // Fork for left command
        if (fork() == 0) {
            dup2(pipefd[1], STDOUT_FILENO);
            close(pipefd[0]);
            close(pipefd[1]);
            execute_ast(node->left);  // Execute left first
            exit(0);
        }

        // Fork for right command
        if (fork() == 0) {
            dup2(pipefd[0], STDIN_FILENO);
            close(pipefd[0]);
            close(pipefd[1]);
            execute_ast(node->right);  // Execute right second
            exit(0);
        }

        close(pipefd[0]);
        close(pipefd[1]);
        wait(NULL);
        wait(NULL);
    }
    else if (node->type == NODE_COMMAND) {
        // 1. Setup redirections first
        setup_redirections(node->redirections);

        // 2. Expand arguments (handle quotes, variables)
        char **argv = expand_arguments(node->args);

        // 3. Execute command
        execve(node->name, argv, environ);
```

```
    }
}
```

Redirection Processing

c

```c
void setup_redirections(Redirection *redirs, int count) {
    // Process in ORDER (left to right)
    for (int i = 0; i < count; i++) {
        if (redirs[i].type == REDIR_IN) {        // <
            int fd = open(redirs[i].target, O_RDONLY);
            dup2(fd, STDIN_FILENO);
            close(fd);
        }
        else if (redirs[i].type == REDIR_OUT) {   // >
            int fd = open(redirs[i].target, O_WRONLY | O_CREAT | O_TRUNC, 0644);
            dup2(fd, STDOUT_FILENO);
            close(fd);
        }
        else if (redirs[i].type == REDIR_APPEND) { // >>
            int fd = open(redirs[i].target, O_WRONLY | O_CREAT | O_APPEND, 0644);
            dup2(fd, STDOUT_FILENO);
            close(fd);
        }
        else if (redirs[i].type == REDIR_HEREDOC) { // <<
            // Create pipe, write content, redirect
            int pipefd[2];
            pipe(pipefd);
            write(pipefd[1], redirs[i].content, strlen(redirs[i].content));
            close(pipefd[1]);
            dup2(pipefd[0], STDIN_FILENO);
            close(pipefd[0]);
        }
    }
}
```

Key Reading Rules

1. **Top to Bottom**: Start at root, traverse downward
2. **Left to Right**: Execute left before right for pipelines
3. **Redirections First**: Setup before command execution
4. **Array Order**: Process arguments/redirections sequentially
5. **Depth-First**: For nested pipelines, go deep before wide

---

Implementation Tips for 42 Minishell

1. **Quote Handling**: Single quotes prevent expansion, double quotes allow it
2. **Heredoc**: Needs special handling - read lines until delimiter
3. **Pipeline Precedence**: Pipes connect commands left-to-right
4. **Redirection Order**: Matters! Process left-to-right as they appear
5. **Memory Management**: Each node needs proper malloc/free handling