CSE 450
Translation of Programming Languages

Lecture 3: Git Setup and More Lexing
Lecture Overview

- **Git version control**
- SSH keys
- Continuous Integration with Travis CI
- Walkthrough on git setup on CSE servers
- More Lex
- RegEx Review
Why do you need to know git for CSE450

- Because you will be turning in your projects with it
- The only way to submit your projects (on time or late) is to push the code to your private GitHub repository
Better Reasons

- Collaboration
- Storing Versions
- Restoring Previous Versions
- Understanding What Happened
- Backup
Git and versions

Checkins over time

Version 1
- A

Version 2
- A1
- B

Version 3
- A1
- B
- C1

Version 4
- A2
- B1
- C2

Version 5
- A2
- B2
- C3
Staging area

http://git-scm.com
Demonstration of making a commit
BRANCHES AND COMMITS
Demonstration of Branches and merging
What command do you use to view commits?

- `git commit`
- `git lg`
- `git status`
- `git log`
Remotes

```plaintext
0b743  a6b4c  f4265

git clone janedoe@git.ourcompany.com:project.git

My Computer

origin/master  Remote branch

0b743  a6b4c  f4265

master  Local branch
```
git.ourcompany.com

```
0b743 → a6b4c → f4265 → 31b8e → 190a3
```

```
git fetch origin
```

```
My Computer
```

```
0b743 → a6b4c → f4265 → 31b8e → 190a3
```

```
origin/master
```

```
a38de → 893cf
```

```
master
```
Lecture Overview

- Git version control
- **SSH keys**
- Continuous Integration with Travis CI
- Walkthrough on git setup on CSE servers
- More Lex
- RegEx Review
SSH Keys (Public / Private Keys)

- Authentication is annoying (typing in usernames and passwords)
  - Doubly so when you need to do it often (like when using git)

- SSH keys provide a safe alternative to usernames and passwords

- After the one-time setup, you never need to type in your GitHub password into the terminal again
Public and Private Keys

Remote Server

Local Computer

What does the remote server (e.g. github.com) need to authenticate you?

- Public Key and Private Key
- Just Public Key
- Just Private Key
- Username and Password
Lecture Overview

- Git version control
- SSH keys
- Continuous Integration with Travis CI
- Walkthrough on git setup on CSE servers
- More Lex
- RegEx Review
Continuous integration

- Travis CI provides a useful service: Continuous Integration
- After the setup, every time you push commits to GitHub, it will run the tests provided and report the results.
- You can configure Travis CI to email you this report.
- This is the only sure way to confirm you are passing the provided tests.

http://www.mera.com/services/processes/integration
See Git Setup
Lecture Overview

- Git version control
- SSH keys
- Continuous Integration with Travis CI
- **Walkthrough on git setup on CSE servers**
- More Lex
- RegEx Review
Lecture Overview

- Git version control
- SSH keys
- Continuous Integration with Travis CI
- Walkthrough on git setup on CSE servers
- More Lex
- RegEx Review
tokens = ('WEEKDAY', 'WEEKEND', 'OTHER', 'WHITESPACE')

def t_WHITESPACE(t):
    r'[	

\]'
    pass

def t_WEEKDAY(t):
    r'''Monday|Tuesday|
    Wednesday|
    Thursday|Friday'''
    print(t.value + " is a week day")

def t_WEEKEND(t):
    r'Saturday|Sunday'
    print(t.value + " is a week end day")

def t_OTHER(t):
    r'[a-zA-Z]+'
    print(t.value + " is not a day")

data = '''Monday Someday
Tuesday Stop Saturday..."
Counting Words

data = "one
two three"

# Build the lexer
import ply.lex as lex
lexer = lex.lex()

# Give thelexer some input
lexer.input(data)

# Tokenize
while True:
    tok = lexer.token()
    if not tok:
        break  # No more input
    print(tok)

print("line count = ",format(LINE_COUNT))
print("word count = ",format(WORD_COUNT))
print("char count = ",format(CHAR_COUNT))

CHAR_COUNT = 0
WORD_COUNT = 0
LINE_COUNT = 0
tokens = ("word", "eol", "other")
def t_word(t):
    r'[^ \t\n]+'
    global WORD_COUNT
    global CHAR_COUNT
    WORD_COUNT += 1
    CHAR_COUNT += len(t.value)
def t_eol(t):
    r'\n'
    global CHAR_COUNT
    global LINE_COUNT
    CHAR_COUNT += 1
    LINE_COUNT += 1
def t_other(t):
    r'.'
    global CHAR_COUNT
    CHAR_COUNT += 1
Lecture Overview

- Git version control
- SSH keys
- Continuous Integration with Travis CI
- Walkthrough on git setup on CSE servers
- More Lex
- RegEx Review
Designing Patterns

Designing the proper regular expressions for patterns can be tricky, but you are provided with a broad range of building blocks.

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>A dot will match any single character except a newline.</td>
</tr>
<tr>
<td>*</td>
<td>Star and plus are used to match zero/one or more of the preceding expressions.</td>
</tr>
<tr>
<td>+</td>
<td>Matches zero or one copy of the preceding expression.</td>
</tr>
<tr>
<td></td>
<td>A logical 'or' statement - matches either the pattern before it, or the pattern after it.</td>
</tr>
</tbody>
</table>
Brackets are used to denote a character class, which matches any single character within the brackets. If the first character is a '^', this negates the brackets, causing them to match any character except those listed. The '-' can be used in a set of brackets to denote a range. Escape sequences must use a '\'.

Group everything within the parentheses as a single unit for the rest of the expression.
Designing Patterns

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x}</td>
<td>Match the preceding expression (x) times.</td>
</tr>
<tr>
<td>{x,y}</td>
<td>Match the preceding expression (x) to (y) times ((x &lt; y))</td>
</tr>
<tr>
<td>^</td>
<td>Matches the very beginning of a line.</td>
</tr>
<tr>
<td>$</td>
<td>Matches the end of a line.</td>
</tr>
</tbody>
</table>
Example Patterns

\[0-9]\]
A single digit.

\[0-9]\+
An integer.

\[0-9]\+\(\backslash\.[0-9]+\)?
An integer or floating point number (always starting with a digit).

\[+\-\]?\[0-9]\+\(\backslash\.[0-9]+\)?\(\[eE][+\-]?[0-9]+\)?
Positive or negative integer, floating point, or scientific notation
More patterns...

What regular expression can we use to detect comments?

`#.*`

What about literal strings?

Does this work? `".*"`

What about: `"[^\"]*"`

No multi-line strings: `"[^"\n]*"`

Allow escape chars: `"([^\n]\ | [^"\])\)*"`
Problems for you

Give a regular expression that defines all email addresses?

What sets of binary strings are defined by the following regular expression:

0 (0 | 1)* 0

(0 | 1)* 0 (0 | 1)*

0? 1+

(0? 1*)+
More problems for you

Given the regular expressions:

\[ r_1 = 0 (10)^* 1^+ \]
\[ r_2 = (01)^+ 0? \]

1. Find a string corresponding to both \( r_1 \) and \( r_2 \).
2. Find a string corresponding to \( r_1 \) but not \( r_2 \).
3. Find a string corresponding to \( r_2 \) but not \( r_1 \).
4. Find a string corresponding to neither \( r_1 \) nor \( r_2 \).