COP3402 Systems Software

Lab 2

Outline

- git and github
  - SSH key
  - clone/add/commit/push/pull
- Makefiles make your life easier
- I/O redirection in linux
- binaries.tar
  - compiled complete project
- compiler and vm
  - first compile, then run on vm
- PL0 with examples
- Regular expressions

SSH Keys

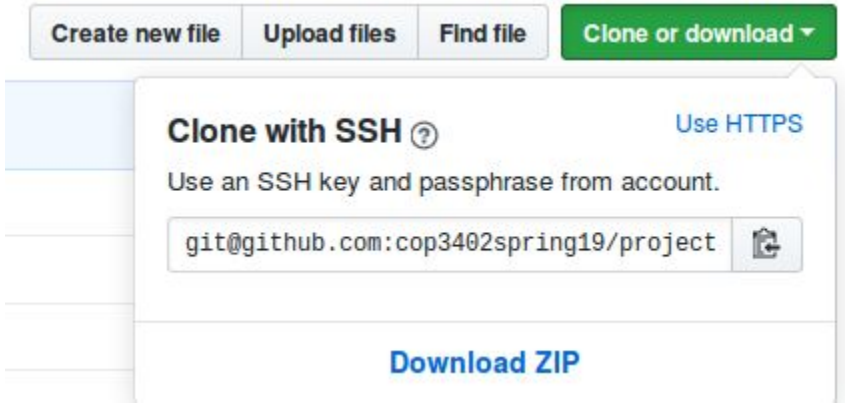
- Using SSH key makes it easier to work with github
  - No need to type username & password when cloning / pushing / pulling

1) Create SSH key on your computer
   - You might already have ssh key
     - $ cd ~/.ssh
     - $ ls
     - If you see id_rsa.pub, you already have a key pair and don't need to create a new one
   - `ssh-keygen -t rsa -b 4096 -C "your_email@example.com"`
     - $ cat ~/.ssh/id_rsa.pub

2) Add the SSH key to your github account

   Github.com > settings > ssh keys

Clone the project




```
git clone git@github.com:cop3402spring19/pro
```

git status/add/commit/push/pull

- git status
  - Check the status - do I have changes to add/commit?
- git add
  - After each change to file: stage the file to be committed
- git commit -m “your commit message”
  - Like saving
- git push
  - Send your local commits to remote repository: github server
- git pull
  - Pull the most recent state of github repository
  - If you are working with a teammate, pull the changes your friend made on the repository to your local machine

Makefile

- It is **faster** to use makefile
- **Just type “make”** instead of compiling each file separately
- make is wise enough to **not recompile** a file if it is not changed

I/O redirection in Linux: **>**, **<**, **2>**

- It is needed because:
  - It becomes cumbersome to see all the output in the terminal
  - We want to save the output to some file
  - Sometimes, we even want to ignore the output (use /dev/null)
- Demo with io_redirection.c
  - ./io_redirection.out **>** stdout.txt
  - ./io_redirection.out **>** stdout.txt **2>** stderr.txt
  - If you just want to kill the stream, use /dev/null
    - ./io_redirection.out **>** stdout.txt **2>** /dev/null

binaries.tar

- Compiled complete project
  - **compiler** and **vm** are executables
  - files with **.o** extension are object files
  - if you miss lexer project, just use lexer.o to continue with the next step
- Download from webcourses > files

First compile, then, run on vm

- **compiler**'s output is machine code (pcode)
  - **Input:** PL0 code
  - **Output:** Machine code
  - Input/output can change if different options are used (e.g. --lex)
- **vm** runs the machine code
  - **Input:** Machine code (pcode)
  - Output: vmout (resulting from **write** statements in PL0 code)
- **Important:** Take a look at overview.md to understand better
  - https://github.com/cop3402spring19/syllabus/blob/master/project/overview.md

PL0 code examples

- Find the examples in syllabus repository
  - syllabus/project/tests
- Demo: **compiler** and **vm**

Regular language / regular expression

What is regular language?

A language that can be defined by regular expressions

What is regular expression?

A sequence of characters that defines a search pattern.

Operation	Regular Expression	Yes	No
Concatenation	aabaab	aabaab	every other string
Logical Or	aa | baab	aa baab	every other string
Replication	ab*a	aa aba abbba	ε ab ababa
Grouping	a(a|b)aab	aaaab abaab	every other string
	(ab)*a	a aba ababa	ε aa abbba

Regular expression: operators

- **L(R)**: Language defined by regular expression **R**
- Union
  - $L(R_1 \mid R_2) = L(R_1) \cup L(R_2)$
- Concatenation
  - $L(R_1R_2)$ = $L(R_1)$ concatenated with $L(R_2)$
- Kleene closure
  - $L(R_1^*)$ = *epsilon* **U** $L(R_1)$ **U** $L(R_1R_1)$ **U** $L(R_1R_1R_1)$ **U** ….

Regular expression: examples

- The language that includes only the strings *ab* and *bb*
  - ab | bb
  - (a | b) b
- All possible binary number representations
  - (0 | 1)*
- The set of strings over {0,1} that end in 3 consecutive 1's.
  - (0 | 1)* 111

Regular expression: usage with grep

- grep “regular expression” file.txt

```
^ (Caret)          =    match expression at the start of a line, as in ^A.
$ (Question)       =    match expression at the end of a line, as in A$.
\ (Back Slash)     =    turn off the special meaning of the next character, as in \^.
[ ] (Brackets)     =    match any one of the enclosed characters, as in [aeiou].
                        Use Hyphen "-" for a range, as in [0-9].
[^ ]               =    match any one character except those enclosed in [ ], as in [^0-9].
. (Period)         =    match a single character of any value, except end of line.
* (Asterisk)       =    match zero or more of the preceding character or expression.
\{x,y\}            =    match x to y occurrences of the preceding.
\{x\}              =    match exactly x occurrences of the preceding.
\{x,\}             =    match x or more occurrences of the preceding.
```

adapted from http://www.robelle.com/smugbook/regexpr.html

Questions?