CS202-001 OS

Review Session 1

Jinli Xiao

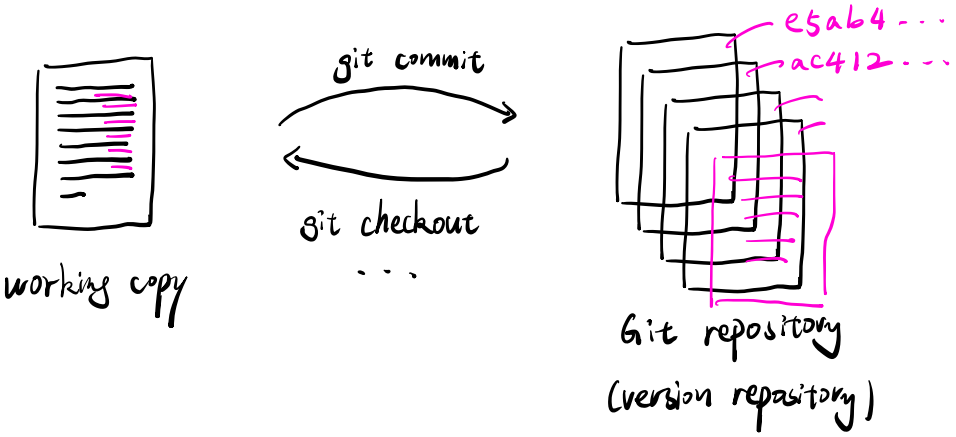
- [x] 0. Record + Attendence
- [x] 1. Introduction
- [x] 2. Logistics
- [x] 3. Motivation + tips
- [x] 4. Lab Infrastructure
- [x] 5. Lab Overview
- [x] 6. Q & A

---

4. Lab Infrastructure

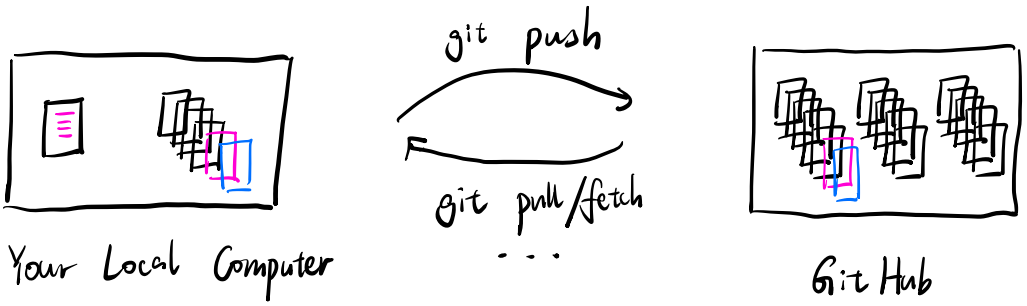
Git

- Version control system written by Linus Torvalds
- able to store every version & highlight differences

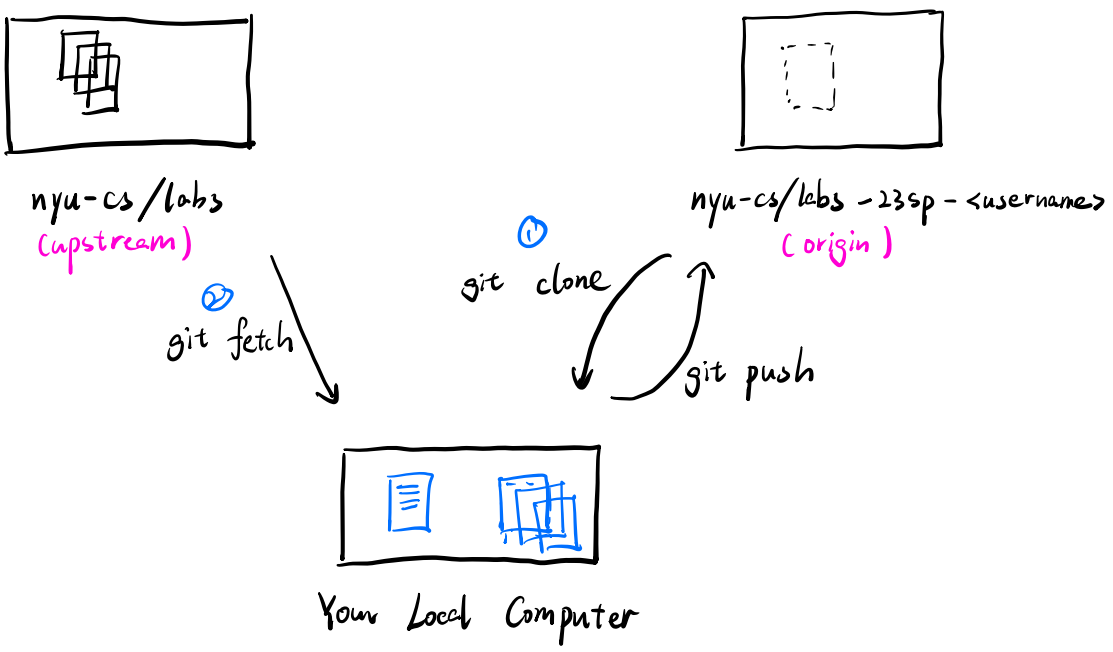


GitHub

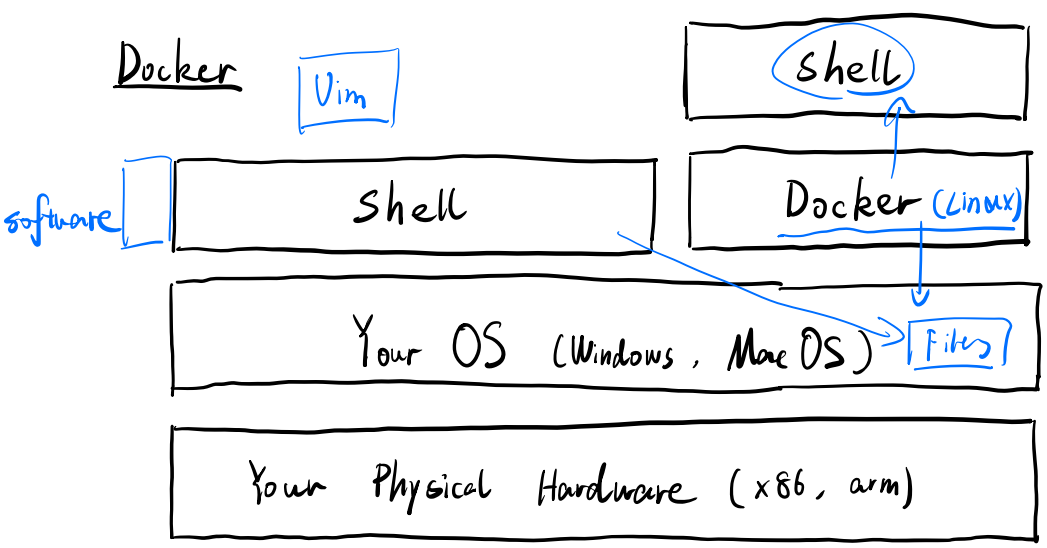
- Git ≠ GitHub
- GitHub hosts Git repositories.
- Alternatives include GitLab / Bitbucket / etc.



Lab Git Workflow



Docker



Scripts & Makefile

- script are a set of commands that can run in shell
- make is a special script that eases the compilation process

Makefile

```
OBJS = MovieList.o Movie.o NameList.o Name.o Iterator.o
CC = g++
DEBUG = -g
CFLAGS = -Wall -c $(DEBUG)
LFLAGS = -Wall $(DEBUG)

p1 : $(OBJS)
    $(CC) $(LFLAGS) $(OBJS) -o p1

MovieList.o : MovieList.h MovieList.cpp Movie.h NameList.h Name.h Iterator.h
    $(CC) $(CFLAGS) MovieList.cpp

Movie.o : Movie.h Movie.cpp NameList.h Name.h
    $(CC) $(CFLAGS) Movie.cpp

NameList.o : NameList.h NameList.cpp Name.h
    $(CC) $(CFLAGS) NameList.cpp

Name.o : Name.h Name.cpp
    $(CC) $(CFLAGS) Name.cpp

Iterator.o : Iterator.h Iterator.cpp MovieList.h
    $(CC) $(CFLAGS) Iterator.cpp

clean:
    \rm *.o *~ p1

tar:
    tar cfv p1.tar Movie.h Movie.cpp Name.h Name.cpp NameList.h \
            NameList.cpp  Iterator.cpp Iterator.h
```

<target>: dependencies
command

---

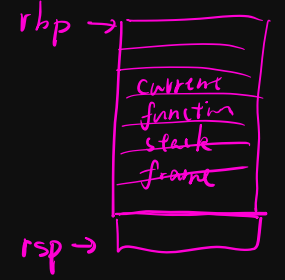
5. Lab 1 Overview

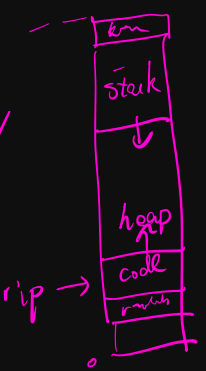
Debugging & gdb

```c
#include <stdio.h>
#include <stdlib.h>

/*
 * Source: Example by Xiangyu Gao
 */

void swap(int para1, int para2) {
  int tmp = para1;
  para1 = para2;
  para2 = tmp;
}

void second_swap(int* para1, int* para2) {
  int tmp = *para1;
  *para1 = *para2;
  *para2 = tmp;
}

int main() {
  int first = 0;
  int second = 10;

  swap(first, second);
  printf("result after first swap: first = %d, second = %d\n", first, second);

  second_swap(&first, &second);
  printf("result after second swap: first = %d, second = %d\n", first, second);

  return 0;
}

```

```c
#include <stdio.h>
#include <stdlib.h>

/*
 * Source: https://stackoverflow.com/questions/5580761
 */

void p1(int* p) {
  p = (int *) malloc(sizeof(int));
  *p = 10;
}

void p2(int** p) {
  *p = (int *) malloc(sizeof(int));
  **p = 10;
}

int main() {
  int* p = NULL;
  p1(p);
  printf("%d\n", *p);

  p2(&p);
  printf("%d\n", *p);

  free(p);

  return 0;
}

```

```
main():

    # set up frame pointer (aka "base pointer")
    pushq %rbp
    movq  %rsp, %rbp

    # push any call-clobbered register that we will need onto the
    # stack, for example:
    pushq %rcx
    pushq %r8
    pushq %r9

    call f

    # restore call-clobbered registers that we saved. In the above
    # example, it would be:
    popq %r9
    popq %r8
    popq %rcx

    # epilogue: restore call-preserved registers
    movq %rbp, %rsp
    popq %rbp
    ret


f():

    # set up frame pointer (aka "base pointer")
    pushq %rbp
    movq  %rsp, %rbp

    ...
```





cd ~/.ssh/

```
main:
    pushq   %rbp                # prologue: store caller's frame pointer
    movq    %rsp, %rbp          # prologue: set frame pointer for new frame

    subq    $16, %rsp           # make stack space

    movq    $0, -8(%rbp)        # x = 0 (x lives at address rbp - 8)
    movq    $8, -16(%rbp)       # arg = 8 (arg lives at address rbp - 16)

    leaq   -16(%rbp), %rdi      # load the address of (rbp-16) into %rdi
                                # this implements "get ready to pass (&arg)
                                # to f"

    call    f                   # invoke f

    movq    %rax, -8(%rbp)      # x = (return value of f)

    # eliding the rest of main()

f:
    pushq   %rbp                # prologue: store caller's frame pointer
    movq    %rsp, %rbp          # prologue: set frame pointer for new frame

    subq   $32, %rsp            # make stack space
    movq   %rdi, -24(%rbp)      # Move ptr to the stack
                                # (ptr now lives at rbp - 24)
    movq   $0, -8(%rbp)         # x = 0 (x's address is rbp - 8)

    movq    -24(%rbp), %r8      # move 'ptr' to %r8
    movq    (%r8), %r9          # dereference 'ptr' and save value to %r9
    movq    %r9, %rdi           # Move the value of *ptr to rdi,
                                # so we can call g

    call    g                   # invoke g

    movq   %rax, -8(%rbp)       # x = (return value of g)
    movq   -8(%rbp), %r10       # compute x + 1, part I
    addq    $1, %r10            # compute x + 1, part II
    movq    %r10, %rax          # Get ready to return x + 1

    movq    %rpb, %rsp          # epilogue: undo stack frame
    popq    %rbp                # epilogue: restore frame pointer from caller
    ret                         # return

g:
    pushq   %rbp                # prologue: store caller's frame pointer
    movq    %rsp, %rbp          # prologue: set frame pointer for new frame

    ....

    movq    %rbp, %rsp          # epilogue: undo stack frame
    popq    %rbp                # epilogue: restore frame pointer from caller
    ret                         # return
```