Git，GitHub与代码仓库

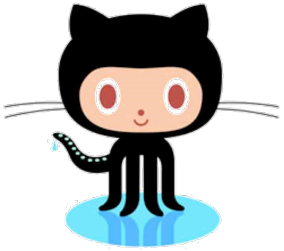
本讲概述

- Git，GitHub与代码仓库
  - Git选讲
  - Git在实验中的应用
- 项目的构建Make

The Community Way

开源社区 (百度百科)：根据相应的开源软件许可证协议公布软件源代码的网络平台，同时也为网络成员提供一个自由学习交流的空间。

- 从GitHub获取代码
  - 传统工具链 + “现代”编程体验
    - ~~有的时候网络不太靠谱~~



github
SOCIAL CODING

```
git clone -b 2022 git@github.com:NJU-ProjectN/ics-pa.git ics2022
git clone https://github.com/NJU-ProjectN/ics-workbench
```

GitLab

gitee

https://git.nju.edu.cn/

The Community Way (cont'd)

GitHub is a development platform inspired by the way you work. From open source to business, you can host and review code, manage projects, and build software alongside 50 million developers.

- 无所不能的代码聚集地
  - 有整个计算机系统世界的代码
    - 硬件、操作系统、分布式系统、库函数、应用程序……
- 学习各种技术的最佳平台
  - 海量的文档、学习资料、博客（新世界的大门）
    - 提供友好的搜索（例子：**awesome C**）

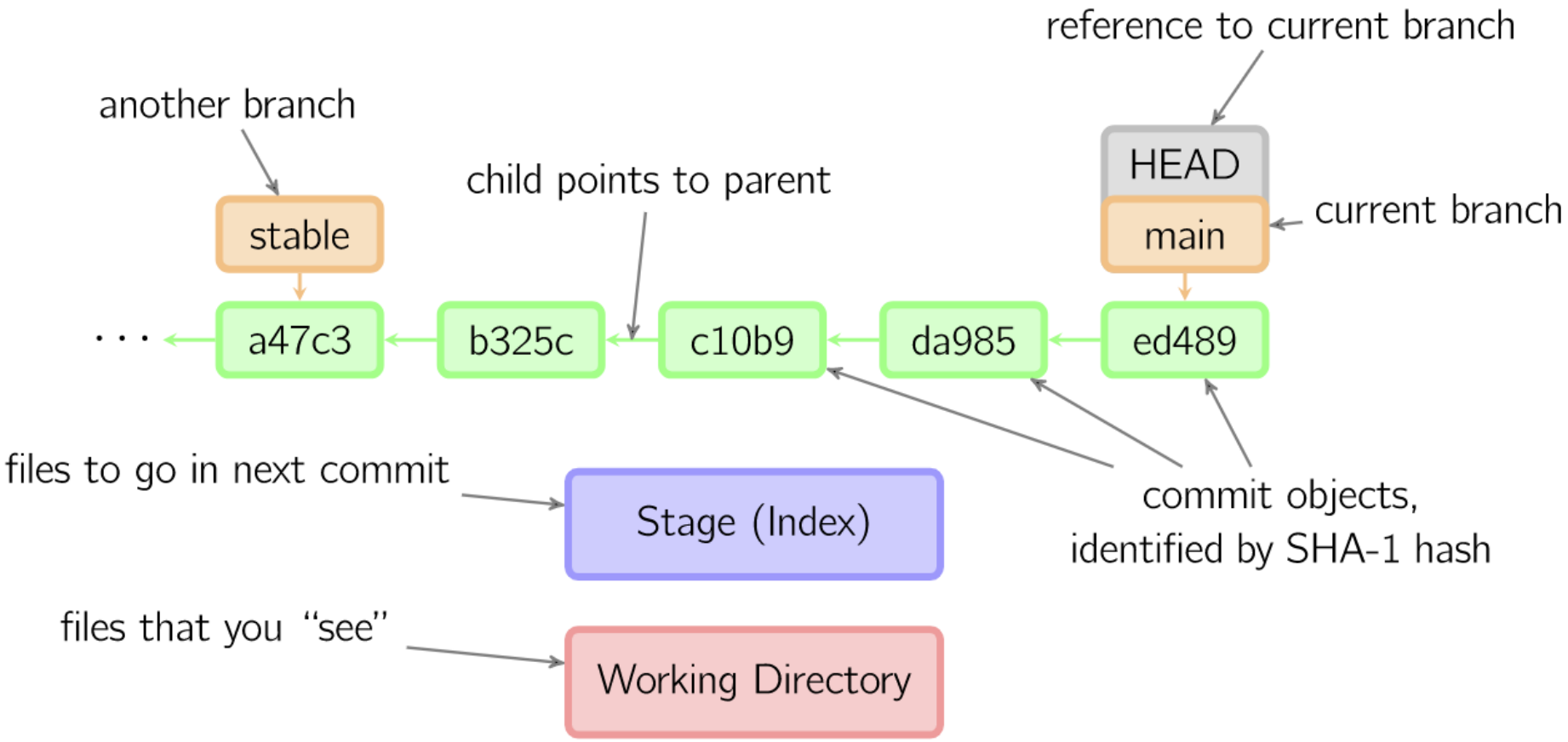
学习Git?

```
git clone -b 2021 https://github.com/NJU-ProjectN/ics-pa ics2021
```

内心独白：又要学新东西，我拒绝= =

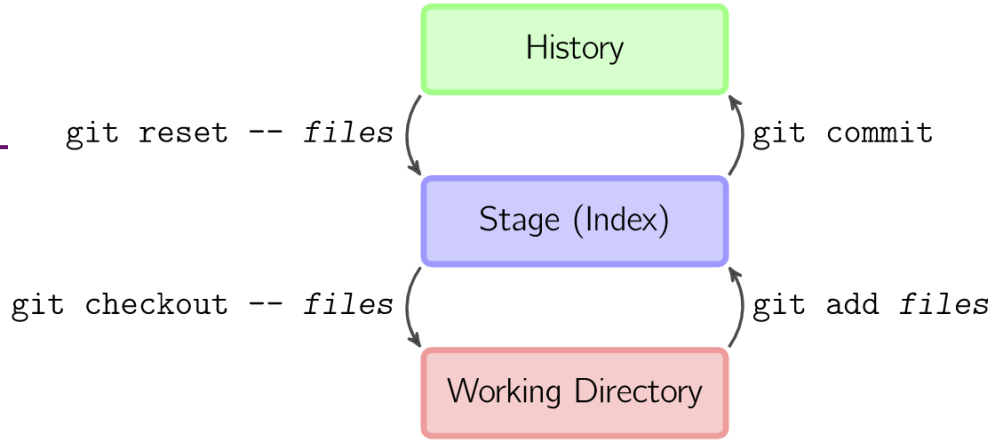
- RTFM?STFW!
  - 百度：得到一堆不太靠谱的教程
  - 大家已经见识过开源社区的力量了
    - A Visual Git Reference
      - 英文、中文、日文……
    - 好的文档是存在的
      - 还记得 tldr 吗?

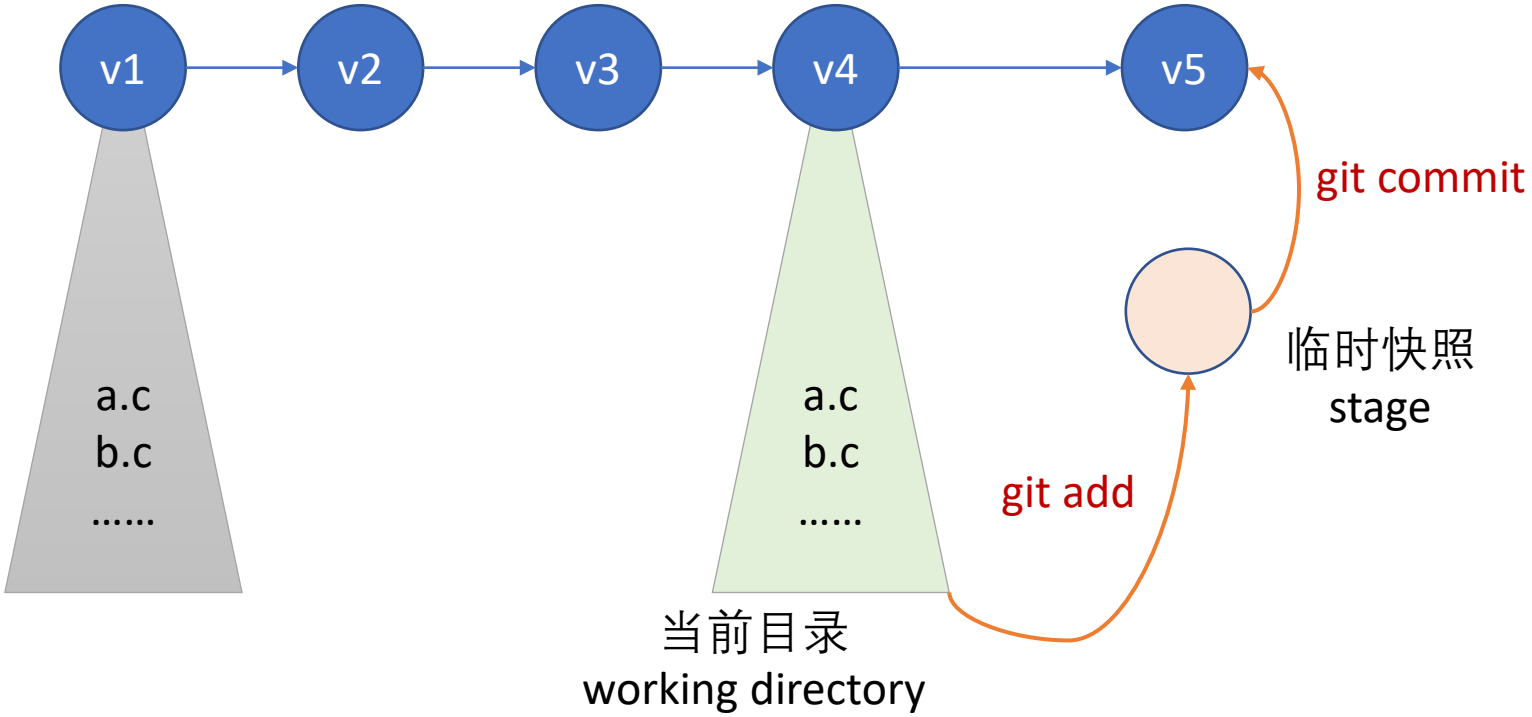
A Visual Git Reference



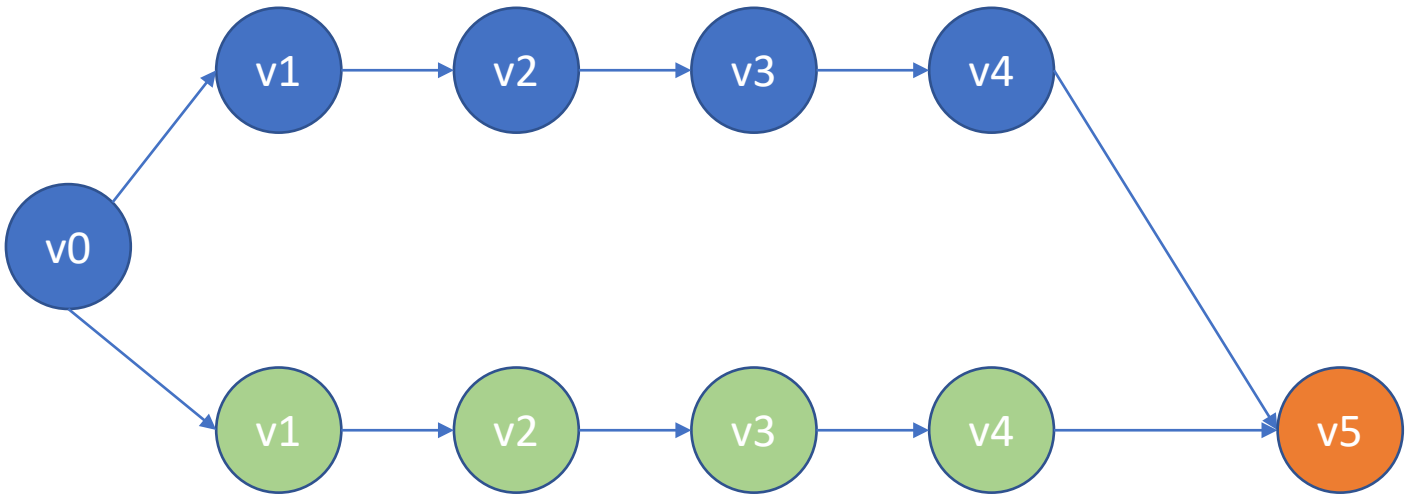

```
commit 8738b7b32e71a78f5b73376dc664780dd16800eb (HEAD -> pa1)
Author: tracer-ics2020 <tracer@njuics.org>
Date:   Thu Aug 19 18:27:15 2021 +0800
```

Git

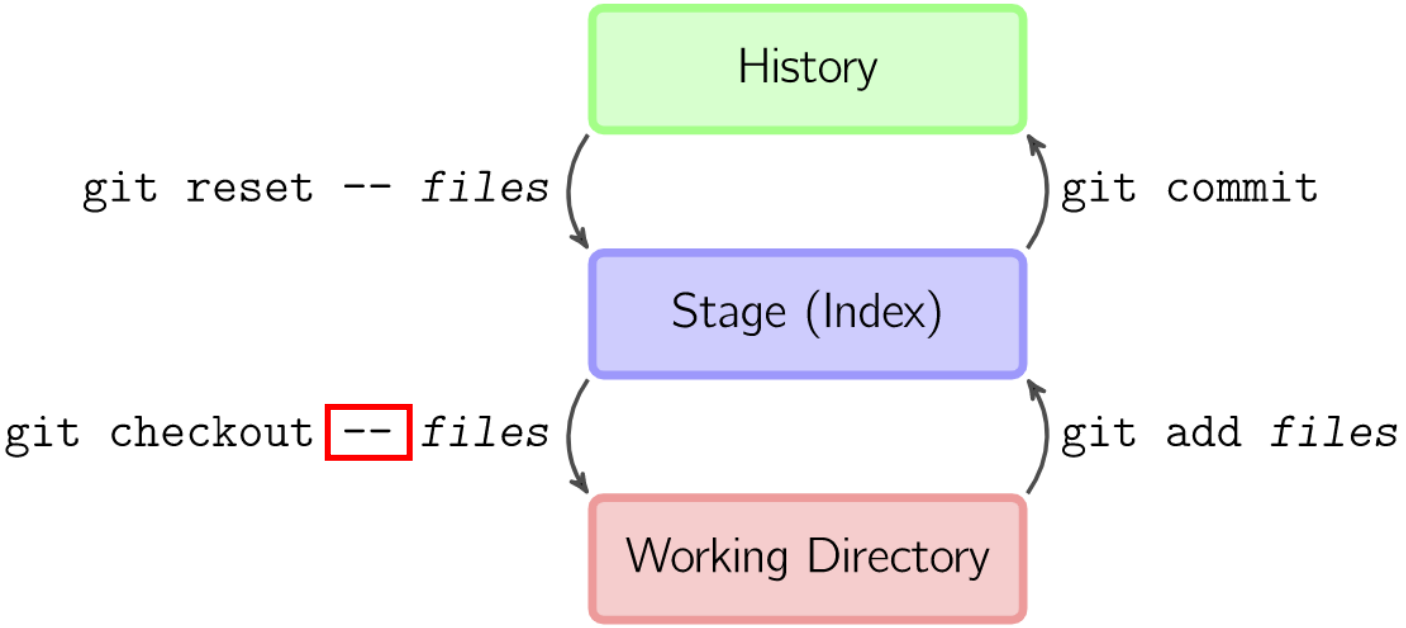

History
git reset -- files
git commit
Stage (Index)
git checkout -- files
git add files
Working Directory


v1
v2
v3
v4
v5
a.c
b.c
.....
a.c
b.c
.....
git commit
临时快照
stage
git add
当前目录
working directory

Git：分布式版本控制系统



A Visual Git Reference (cont'd)

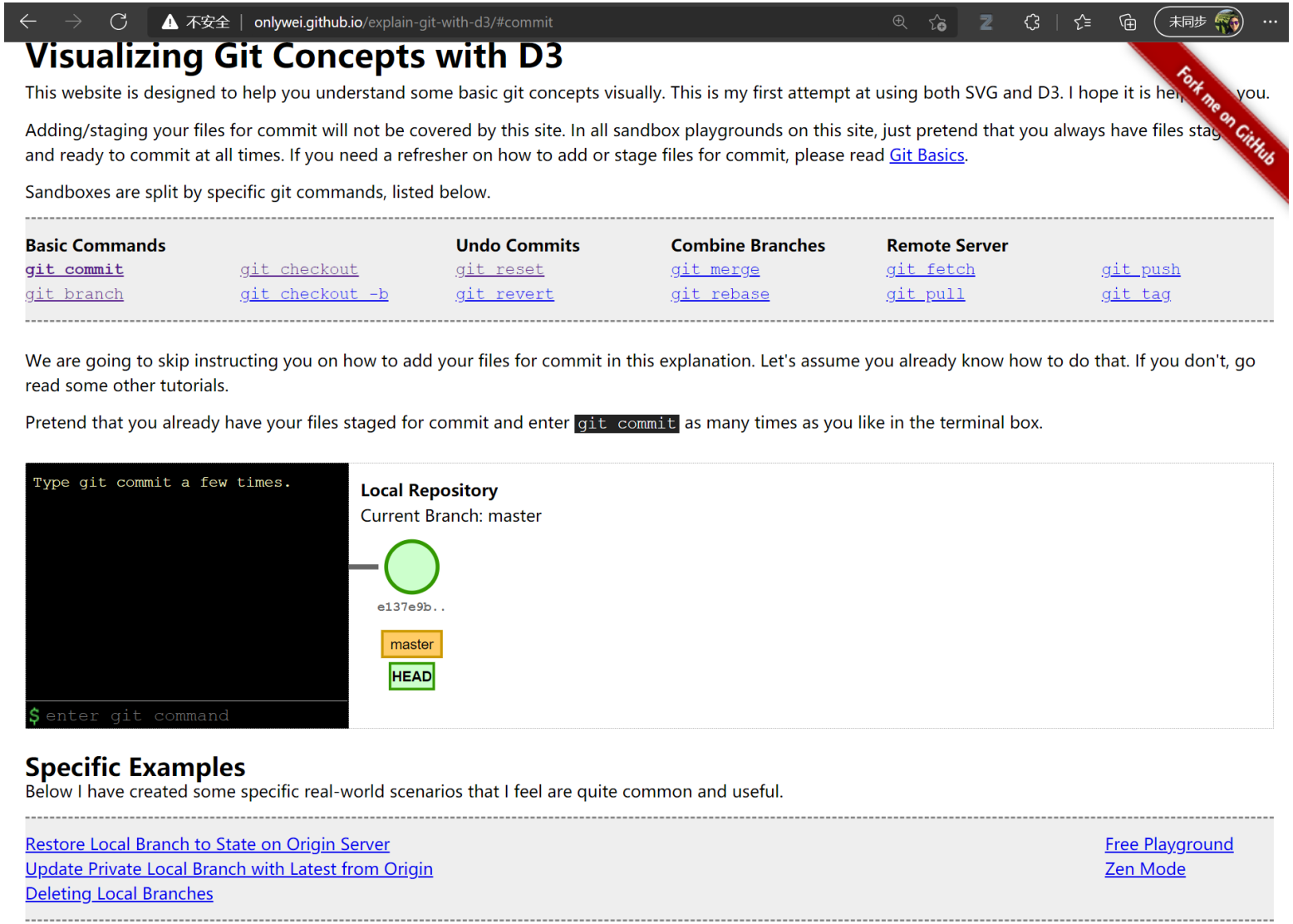



```
$ ls -l
total 0
-rw-r--r-- 1 why why 0 Aug 25 18:10 -rf
$ rm -- -rf
$ ls -l
total 0
```

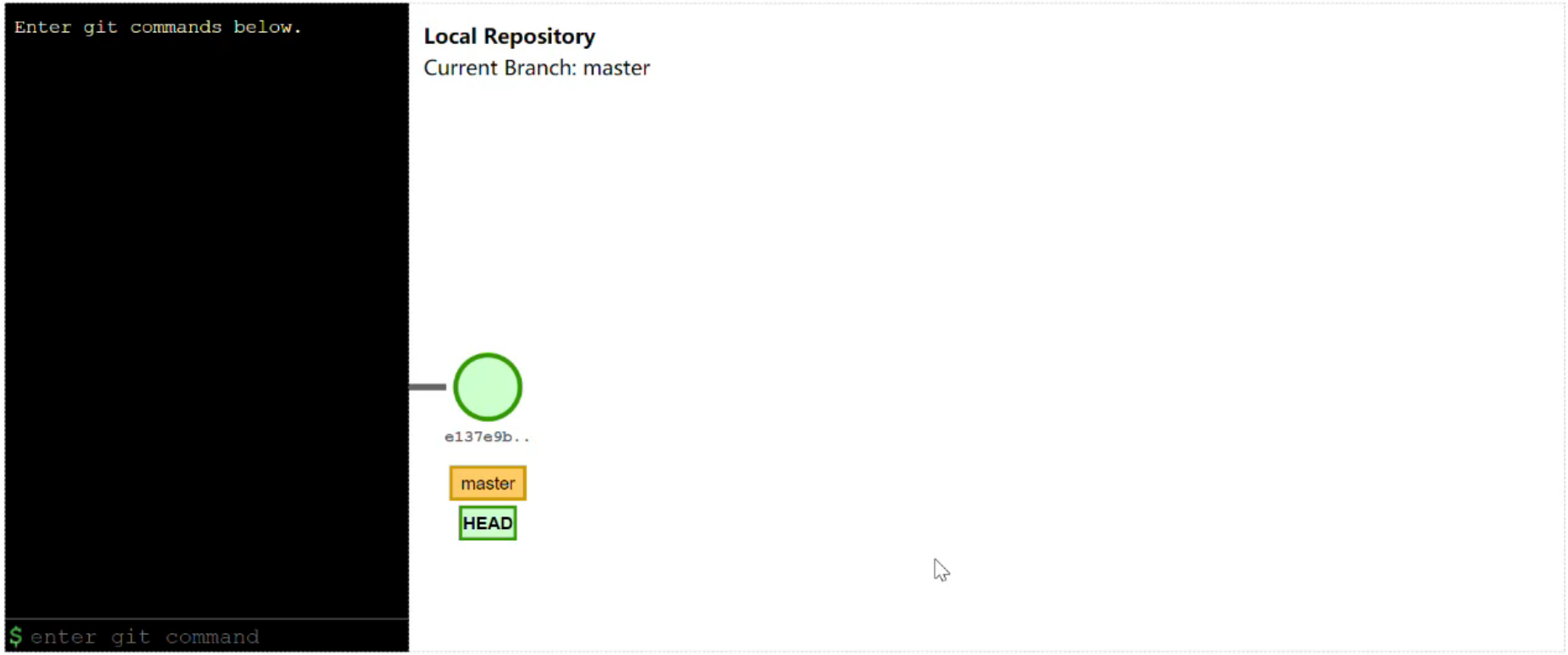
一些Comments

- 体验 Git
  - 创建一个新的 repo，自由探索
    - 为什么 “预计完成时间 XX 小时” 是骗人的?
      - 预计完成时间是假设你在大一开始就用 Git 的
  - Visualizing Git Concepts with D3

Visualizing Git Concepts with D3



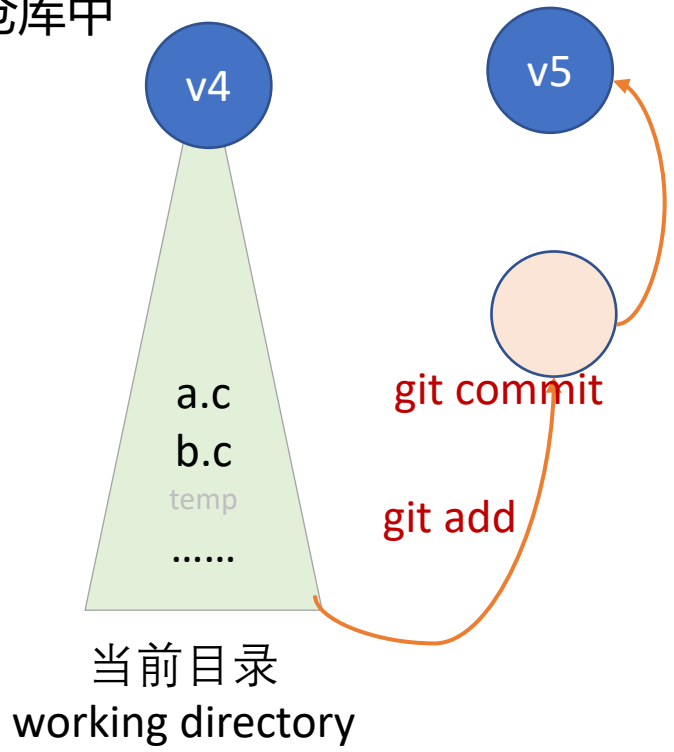
Visualizing Git Concepts with D3



一些Comments (cont'd)

- 我们使用了“白名单” .gitignore文件
  - 只在Git repo里管理.c, .h和Makefile
    - 基本原则：一切生成的文件都不放在Git仓库中

```
*           # 忽略一切文件
!*/         # 除了目录
!*.c        # .c
!*.h        # ...
!Makefile*
!.gitignore
```



- 为什么ls看不到这个文件?
  - 怎么还有一个.git

```
git clone -b 2021 https://github.com/NJU-ProjectN/ics-pa ics2021
```

获取PA代码

```
git clone -b 2022 git@github.com:NJU-ProjectN/ics-pa.git ics2022
```

- 获取来自github的一整个历史

提交脚本

- make submit会下载执行http://jyywiki.cn/static/submit.sh
  - Makefile

```
submit:
   git gc
   STUID=$(STUID) STUNAME=$(STUNAME) bash -c "$$(curl -s
http://why.ink:8080/static/submit.sh)"
```

```
bash -c "$(curl -s http://why.ink:8080/static/submit.sh)"
```

- 我们做了什么?

```
why@why-VirtualBox:~/Documents/ICS2022/ics2022$ curl -s http://why.ink:8080/static/submit.sh
COURSE=ICS2022
MODULE=$(git rev-parse --abbrev-ref HEAD | tr '[a-z]' '[A-Z]')
FILE=/tmp/upload.tar.bz2
tar caf "$FILE" ".git" $(find . -maxdepth 2 -name "*.pdf") && \
  curl -F "token=$TOKEN" \
       -F "course=$COURSE" \
       -F "module=$MODULE" \
       -F "file=@$FILE" \
       http://why.ink:8080/upload
```

提交脚本

- make submit会下载执行http://why.ink:8080/static/submit.sh

```
bash -c "$(curl -s http://why.ink:8080/static/submit.sh)"
```

```
$ git branch
  master
* pa0
  pa1
$ git rev-parse --abbrev-ref HEAD
pa0
$ git checkout pa1
Switched to branch 'pa1'
$ git rev-parse --abbrev-ref HEAD
pa1
```

PA1?PA2?

```
# submit.sh (服务器)
COURSE=ICS2023
MODULE=$(git rev-parse --abbrev-ref HEAD | tr '[a-z]' '[A-Z]’)
FILE=/tmp/upload.tar.bz2

tar caf "$FILE" "$NAME/.git" $(find $NAME -maxdepth 2 -name "*.pdf") && \
    curl -F “token=$TOKEN"
         -F "course=$COURSE"
         -F "module=$MODULE" \
         -F "file=@$FILE" http://why.ink:8080/upload
```

OJ得到的是.git

- .git包含了OJ想知道的一切

```
./.git
./.git/objects
./.git/objects/info
./.git/objects/pack
./.git/hooks
./.git/info
./.git/info/exclude
./.git/HEAD
./.git/branches
./.git/description
./.git/refs
./.git/refs/heads
./.git/refs/tags

./.git/config # 配置
```

Git追踪

- 另一段神秘代码 (来自 Makefile.lab)

```
git:
	@git add $(shell find . -name "*.c") \
	    $(shell find . -name "*.h") -A --ignore-errors
	@while (test -e .git/index.lock); do sleep 0.1; done
	@(hostnamectl && uptime) | \
	    git commit -F - -q --author=tracer-nju <tracer@nju.edu.cn>
--no-verify --allow-empty   @sync
```

- 在每次 make 执行时
  - git 目标都会被执行
  - 将 .c 和 .h 添加、强制提交到 Git repo

```
$ hostnamectl
   Static hostname: why-VirtualBox
         Icon name: computer-vm
           Chassis: vm
        Machine ID: 22e7c7921a0a437bb1817612c8ab2cce
           Boot ID: 51bcab304005479b8dcec150413ee8bf
    Virtualization: oracle
  Operating System: Ubuntu 21.04
            Kernel: Linux 5.11.0-34-generic
      Architecture: x86-64
$
$ uptime
 17:25:29 up  1:45,  1 user,  load average: 0.11, 0.09, 0.02
```

Git追

```
6  GITFLAGS = -q --author='tracer-ics2021 <tracer@njuics.org>' --no-verify --allow-empty
7
8  # prototype: git_commit(msg)
9  define git_commit
10   -@git add $(NEMU_HOME)/.. -A --ignore-errors
11   -@while (test -e .git/index.lock); do sleep 0.1; done
12   -@(echo "> $(1)" && echo $(STUID) && hostnamectl && uptime) | git commit -F - $(GITFLAGS)
13   -@sync
14 endef
```

```
commit 2662595b8712e2247fd1a92b5cb8b9103c9fc01b
Author: tracer-ics2021 <tracer@njuics.org>
Date:   Thu Sep 16 12:07:21 2021 +0800

    >  compile
    ky2107911
       Static hostname: why-VirtualBox
             Icon name: computer-vm
               Chassis: vm
            Machine ID: 22e7c7921a0a437bb1817612c8ab2cce
               Boot ID: 6cbe073460244ae0ad90966cdbbe5961
        Virtualization: oracle
      Operating System: Ubuntu 21.04
                Kernel: Linux 5.11.0-34-generic
          Architecture: x86-64
     12:07:21 up  2:40,  2 users,  load average: 0.23, 0.23, 0.19

commit 713801b46b12063c27db811bd61a9b935395cd35
Author: tracer-ics2021 <tracer@njuics.org>
Date:   Thu Sep 16 12:06:39 2021 +0800

    >  run
    ky2107911
       Static hostname: why-VirtualBox
             Icon name: computer-vm
               Chassis: vm
            Machine ID: 22e7c7921a0a437bb1817612c8ab2cce
               Boot ID: 6cbe073460244ae0ad90966cdbbe5961
        Virtualization: oracle
      Operating System: Ubuntu 21.04
                Kernel: Linux 5.11.0-34-generic
          Architecture: x86-64
     12:06:39 up  2:39,  2 users,  load average: 0.46, 0.26, 0.20
```

这是什么？

- 对抄袭代码的一种威慑
  - 如何抓抄袭
    - 旧方法；但至今仍在使用
  - Git 追踪是 “一劳永逸的新方法”
    - 透过它，我们看到南大学子的人生百态

知乎 首发于 软件工程技术研究漫谈

代码抄袭：那些让985学生沉默，211学生流泪的真相

蒋炎岩
计算机科学与技术博士

2,221 人赞同了该文章

> 这是我们在TURC’18 (SIGCSE China)的论文Needle: Detecting Code Plagiarism on Student Submissions [1]的科普版本。

“我们不生产代码，我们只是互联网的搬运工”——佚名

在平均学历是985的知乎，考不上个好学校都没脸出来冒泡。然而，到底是我们上了大学，还是大学上了我们？这篇文章来谈谈一个985学生知道了会沉默，211学生知道了会流泪的真相：广泛存在的抄作业问题。我想大部分读者朋友们看到这里的感受都是：中枪了。

你还记得上大学的时候，把“大腿”的作业拿来抄抄改改，然后去打游戏的日子吗？或者如果你就是同学们心目中的“大腿”，有没有慷慨相助同学们的经历？也许这类事情的确没有发生在你身上，但在计算机学科里，我们严肃的研究发现：

- 某985大学的一次实验大作业中，若不对代码抄袭控制，有超过82%的同学涉嫌拷贝代码(抄袭或被抄袭)，并且抄袭者会对代码做出不同程度的修改。大家也都知道，计算机专业课程光说不练课可就白上了，也就是说，在一届同学中有超过4/5这门课都白上了。考虑到还有同学选择不交作业，这个比例还会再高一些……
- 在一所985大学的一门课程中，即便使用了抄袭检测工具、制定严格的惩罚措施、给有困难的同学提供“诚信分”，一门课程整个学期下来，依然有约20%的同学涉嫌拷贝代码。甚至有同学被逮到以后还会再抄、再被逮到以后再抄代码，铤而走险，屡教不改。

(以上结论仅针对我们调研的编程实验有效，而且导致同学们抄代码的原因是多种多样的，整体的代码抄袭情况我们尚未完全统计。)

这是什么?

- 对抄袭代码的一种威慑
  - 如何抓抄袭
    - 旧方法；但至今仍在使用
  - Git 追踪是 “一劳永逸的新方法”
    - 透过它，我们看到南大学子的人生百态
- 记得 academic integrity
  - 我们留了足够多的分数给努力尝试过的同学通过
- 未来可能有终极加强版

小结：现代化的项目管理方式

- Git: 代码快照管理工具
  - 是一种 “可持久化数据结构”
  - 拓展阅读：Pro Git
- 框架代码中的两处非常规 Git 使用
  - 提交脚本
    - 仅上传 .git；在服务器执行 git reset
    - 减少提交大小 (仅源文件)
  - Git 追踪
    - 编译时强制提交，获取同学编码的过程
- 思考题：如何管理自己的代码快照?
  - 提示：分支/HEAD/... 只是指向快照的指针 (references)

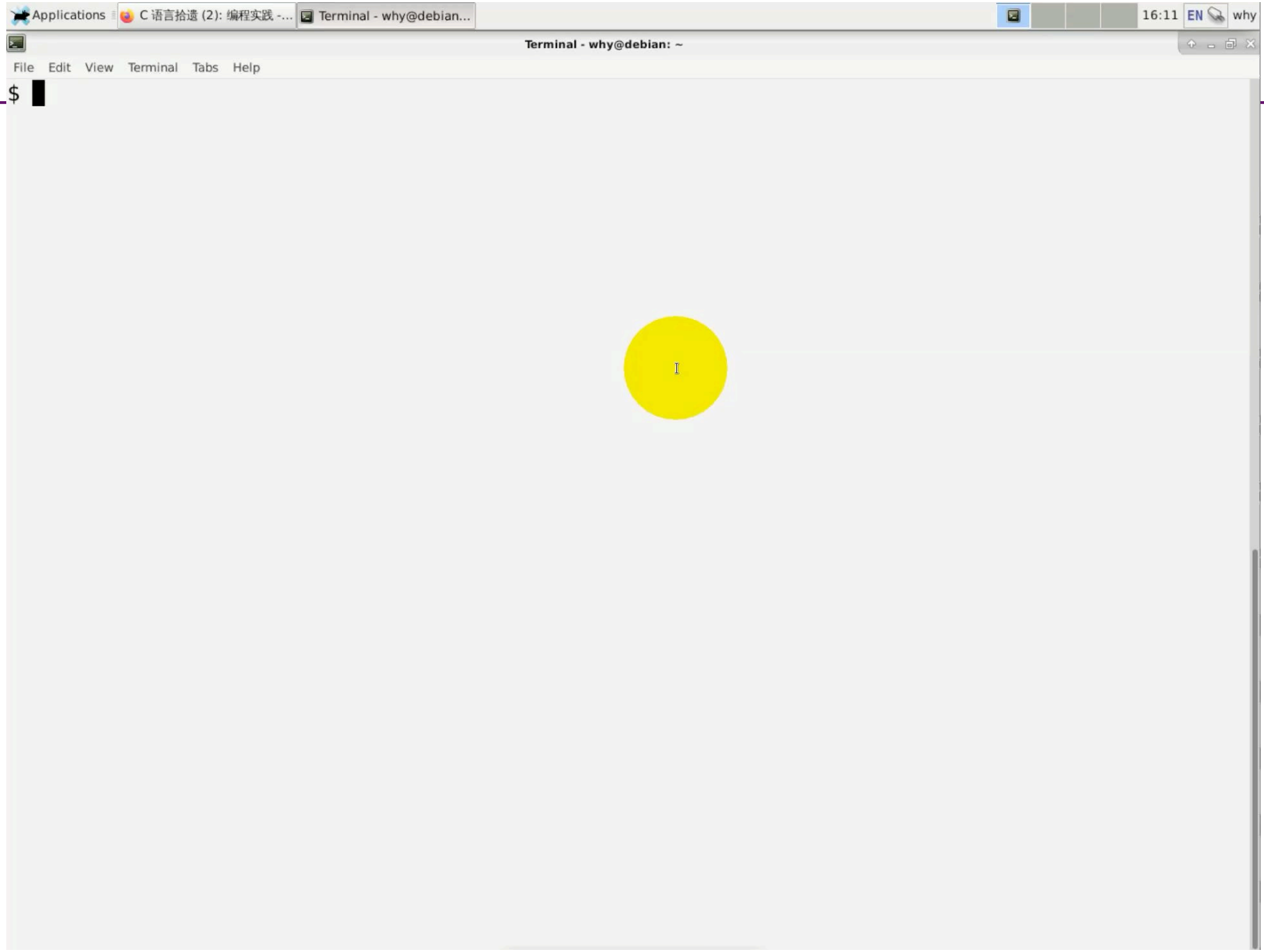
项目构建

Make工具

- 回顾：YEMU 模拟器

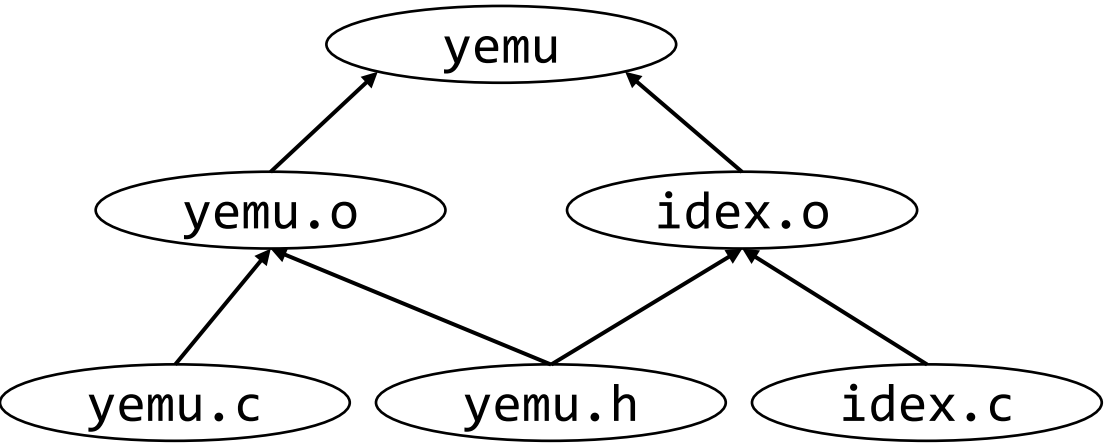
- Makefile 是一段 “declarative” 的代码
  - 描述了构建目标之间的依赖关系和更新方法

```
.PHONY: run clean test

CFLAGS = -Wall -Werror -std=c11 -O2
CC = gcc
LD = gcc

yemu: yemu.o idex.o
    $(LD) $(LDFLAGS) -o yemu yemu.o idex.o

yemu.o: yemu.c yemu.h
    $(CC) $(CFLAGS) -c -o yemu.o yemu.c

idex.o: idex.c yemu.h
    $(CC) $(CFLAGS) -c -o idex.o idex.c

run: yemu
    @./yemu

clean:
    rm -f test yemu *.o

test: yemu
    $(CC) $(CFLAGS) -o test idex.o test.c && ./test
```


Applications
C 语言拾遗 (2): 编程实践 -...
Terminal - why@debian...
16:11
EN
why
Terminal - why@debian: ~
File Edit View Terminal Tabs Help
$

Makefile

- make
  - gcc -Wall -Werror -std=c11 -O2 -c -o yemu.o yemu.c
  - gcc -Wall -Werror -std=c11 -O2 -c -o idex.o idex.c
  - gcc -o yemu yemu.o idex.o



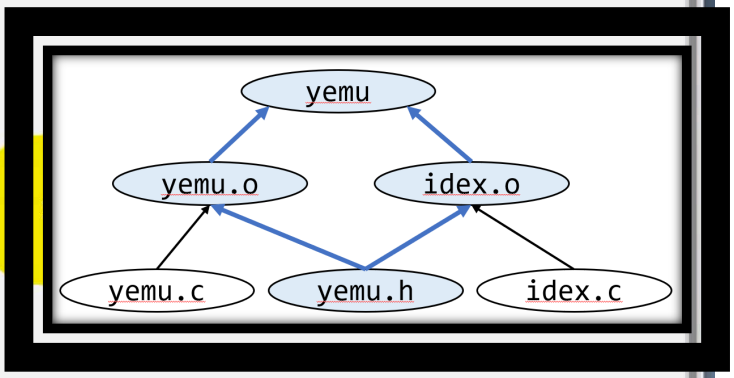

```
1  .PHONY: run clean test
2
3  CFLAGS = -Wall -Werror -std=c11 -O2
4  CC = gcc
5  LD = gcc
6
7  yemu: yemu.o idex.o
8      $(LD) $(LDFLAGS) -o yemu yemu.o idex.o
9
10 yemu.o: yemu.c yemu.h
11     $(CC) $(CFLAGS) -c -o yemu.o yemu.c
12
13 idex.o: idex.c yemu.h
14     $(CC) $(CFLAGS) -c -o idex.o idex.c
15
16 run: yemu
17     @./yemu
18
19 clean:
20     rm -f test yemu *.o
21
22 test: yemu
23     $(CC) $(CFLAGS) -o test idex.o test.c && ./test
```

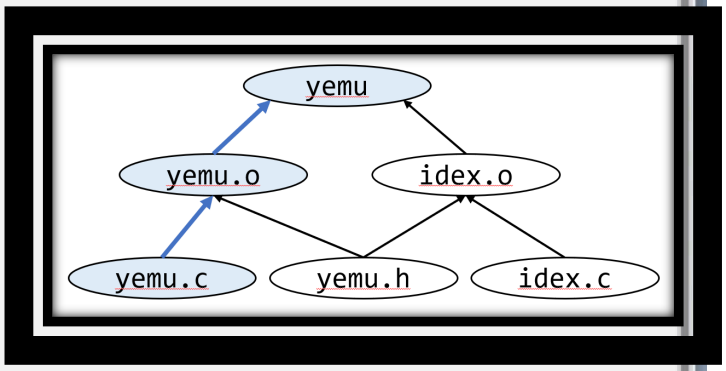
Applications | Terminal - why@debian... | 16:30 EN why

Terminal - why@debian: ~

File Edit View Terminal Tabs Help

```
$ cd ICS_teach/
$ ls
yemu  yemu.tar.gz
$ cd yemu/
$ ls
idex.c  idex.o  Makefile  test.c  yemu  yemu.c  yemu.h  yemu.o
$ make clean
rm -f test yemu *.o
$
```





```
$ make run
Hit GOOD trap @ pc = 5.
M[00] = 0xe7 (231)
M[01] = 0x04 (4)
M[02] = 0xe6 (230)
M[03] = 0x11 (17)
M[04] = 0xf8 (248)
M[05] = 0x00 (0)
M[06] = 0x10 (16)
M[07] = 0x21 (33)
M[08] = 0x31 (49)
M[09] = 0x00 (0)
M[10] = 0x00 (0)
M[11] = 0x00 (0)
M[12] = 0x00 (0)
M[13] = 0x00 (0)
M[14] = 0x00 (0)
M[15] = 0x00 (0)
$ vim Makefile
$ make test
gcc -Wall -Werror -std=c11 -O2 -o test idex.o test.c && ./test
Test #1 (0b11100111): PASS
Test #2 (0b00000100): PASS
Test #3 (0b11100101): PASS
Test #4 (0b00010001): PASS
Test #5 (0b11110111): PASS
```

```
1  .PHONY: run clean test
2
3  CFLAGS = -Wall -Werror -std=c11 -O2
4  CC = gcc
5  LD = gcc
6
7  yemu: yemu.o idex.o
8      $(LD) $(LDFLAGS) -o yemu yemu.o idex.o
9
10 yemu.o: yemu.c yemu.h
11     $(CC) $(CFLAGS) -c -o yemu.o yemu.c
12
13 idex.o: idex.c yemu.h
14     $(CC) $(CFLAGS) -c -o idex.o idex.c
15
16 run: yemu
17     @./yemu
18
19 clean:
20     rm -f test yemu *.o
21
22 test: yemu
23     $(CC) $(CFLAGS) -o test idex.o test.c && ./test
```

make test

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "yemu.h"

u8 R[NREG], M[NMEM], oldR[NREG], oldM[NMEM], *newM = M, *newR = R;

void random_rm() {
  for (int i = 0; i < NREG; i++)
    R[i] = rand() & 0xff;
  for (int i = 0; i < NMEM; i++)
    M[i] = rand() & 0xff;
}

#define ASSERT_EQ(a, b) ((u8)(a) == (u8)(b))

#define TESTCASES(_) \
  _(1, 0b11100111, random_rm, ASSERT_EQ(newR[0], oldM[7]) ) \
  _(2, 0b00000100, random_rm, ASSERT_EQ(newR[1], oldR[0]) ) \
  _(3, 0b11100101, random_rm, ASSERT_EQ(newR[0], oldM[5]) ) \
  _(4, 0b00010001, random_rm, ASSERT_EQ(newR[0], oldR[0] + oldR[1]) ) \
  _(5, 0b11110111, random_rm, ASSERT_EQ(newM[7], oldR[0]) )

#define MAKETEST(id, inst, precond, postcond) { \
  precond(); \
  memcpy(oldM, M, NMEM); \
  memcpy(oldR, R, NREG); \
  pc = 0; M[pc] = inst; \
  idex(); \
  printf("Test #%d (%s): %s\n", \
    id, #inst, (postcond) ? "PASS" : "FAIL"); \
}

int main() {
  TESTCASES(MAKETEST)
  return 0;
}
```

预编译的解析

- `gcc -E test.c | indent - | vim -`

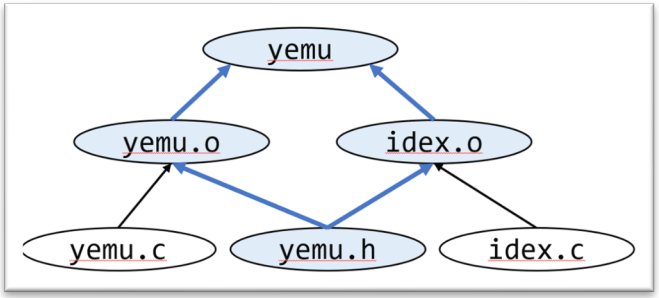
```
#34 "test.c"
int
main ()
{
  {
    random_rm ();
    memcpy (oldM, M, 16);
    memcpy (oldR, R, NREG);
    (R[PC]) = 0;
    M[(R[PC])] = 0 b11100111;
    idex ();
    printf ("Test #%d (%s): %s\n", 1, "0b11100111",
            (((u8) (newR[0]) == (u8) (oldM[7]))) ? "PASS" : "FAIL");
  }
  {
    random_rm ();
    memcpy (oldM, M, 16);
    memcpy (oldR, R, NREG);
    (R[PC]) = 0;
    M[(R[PC])] = 0 b00000100;
    idex ();
    printf ("Test #%d (%s): %s\n", 2, "0b00000100",
            (((u8) (newR[1]) == (u8) (oldR[0]))) ? "PASS" : "FAIL");
  }
  {
    random_rm ();
    memcpy (oldM, M, 16);
    memcpy (oldR, R, NREG);
    (R[PC]) = 0;
    M[(R[PC])] = 0 b11100101;
    idex ();
    printf ("Test #%d (%s): %s\n", 3, "0b11100101",
            (((u8) (newR[0]) == (u8) (oldM[5]))) ? "PASS" : "FAIL");
  }
  {
```

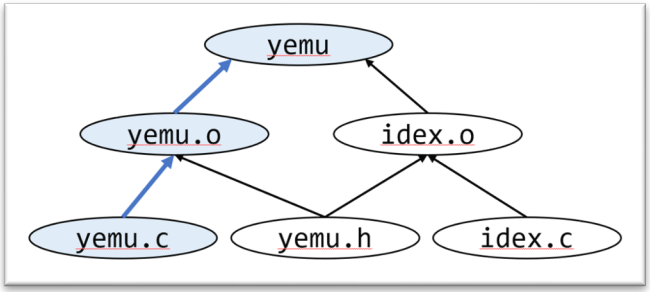
Make工具

- 回顾：YEMU 模拟器

- Makefile 是一段 “declarative” 的代码
  - 描述了构建目标之间的依赖关系和更新方法

```
$ make
gcc -Wall -Werror -std=c11 -O2 -c -o yemu.o yemu.c
gcc -Wall -Werror -std=c11 -O2 -c -o idex.o idex.c
gcc  -o yemu yemu.o idex.o
$ make
make: 'yemu' is up to date.
```





- `make -j8`

Make工具

- 回顾：YEMU 模拟器

- Makefile 是一段 “declarative” 的代码
  - 描述了构建目标之间的依赖关系和更新方法
  - 有用的编译选项
    - -nB、-j
  - 同时也是和 Shell 结合紧密的编程语言
    - 能够生成各种字符串
    - 支持 “元编程” (#include, #define, …)

Lab代码的构建

- 顶层 (top-level) Makefile:

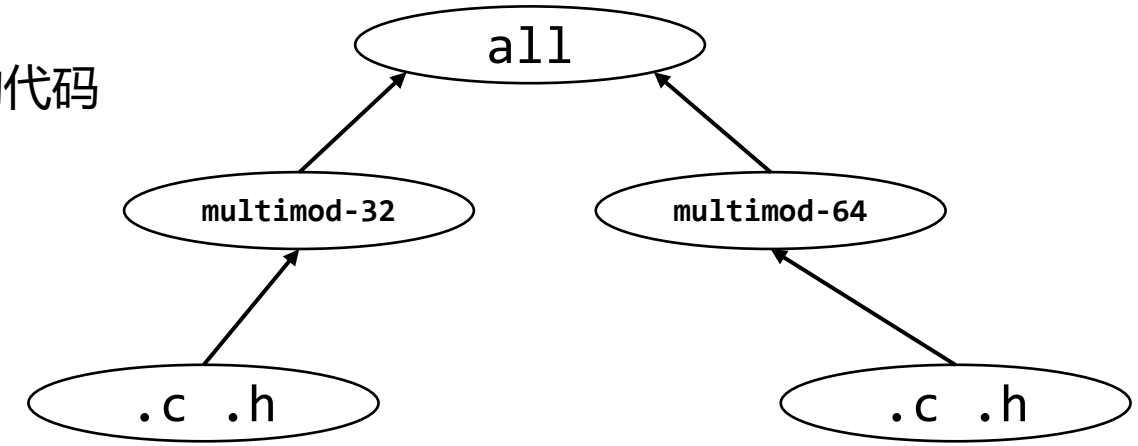
```
# := -> C #define
    NAME := $(shell basename $(PWD))
export MODULE := Lab1

# 变量 -> 字面替换
all: $(NAME)-64 $(NAME)-32

# include -> C #include
include ../Makefile
```

```
1  # **DO NOT MODIFY**
2
3  ifeq ($(NAME),)
4  $(error Should make in each lab's directory)
5  endif
6
7  SRCS    := $(shell find . -maxdepth 1 -name "*.c")
8  DEPS    := $(shell find . -maxdepth 1 -name "*.h") $(SRCS)
9  CFLAGS += -O1 -std=gnu11 -ggdb -Wall -Werror -Wno-unused-result -Wno
   -unused-value -Wno-unused-variable
10
11 .PHONY: all git test clean commit-and-make
12
13 .DEFAULT_GOAL := commit-and-make
14 commit-and-make: git all
15
16 $(NAME)-64: $(DEPS) # 64bit binary
17   gcc -m64 $(CFLAGS) $(SRCS) -o $@ $(LDFLAGS)
18
19 $(NAME)-32: $(DEPS) # 32bit binary
20   gcc -m32 $(CFLAGS) $(SRCS) -o $@ $(LDFLAGS)
21
22 $(NAME)-64.so: $(DEPS) # 64bit shared library
23   gcc -fPIC -shared -m64 $(CFLAGS) $(SRCS) -o $@ $(LDFLAGS)
24
25 $(NAME)-32.so: $(DEPS) # 32bit shared library
26   gcc -fPIC -shared -m32 $(CFLAGS) $(SRCS) -o $@ $(LDFLAGS)
27
28 clean:
29   rm -f $(NAME)-64 $(NAME)-32 $(NAME)-64.so $(NAME)-32.so
30
31 include ../Makefile.lab
```

Applications
Lab1: 大整数运算 - Mozill...
Terminal - why@debian...
18:11
why
Terminal - why@debian: ~/ICS_teach/ics-workbench/multimod
File Edit View Terminal Tabs Help
$ ls
main.c Makefile multimod-32 multimod-64 multimod.c
$ vim Makefile
$ vim Makefile

Lab代码的构建 (cont'd)

- 构建目标
  - 总目标
    - `.DEFAULT_GOAL := commit-and-make`
    - `commit-and-make: git all` (all 在顶层 Makefile 中定义)
  - 可执行文件
    - `multimod-64: gcc -m64`
    - `multimod-32: gcc -m32`
  - 共享库 (之后的 lab 使用)
    - `multimod-64.so: gcc -fPIC -shared -m64`
    - `multimod-32.so: gcc -fPIC -shared -m32`
  - `clean`
    - 删除构建的代码



NEMU代码构建

- Makefile 真复杂
  - 放弃?

```
$ make
+ CC src/nemu-main.c
+ CC src/isa/riscv32/difftest/dut.c
+ CC src/isa/riscv32/system/intr.c
+ CC src/isa/riscv32/system/mmu.c
+ CC src/isa/riscv32/init.c
+ CC src/isa/riscv32/logo.c
+ CC src/isa/riscv32/reg.c
+ CC src/isa/riscv32/instr/decode.c
+ CC src/device/io/map.c
+ CC src/device/io/mmio.c
+ CC src/device/io/port-io.c
+ CC src/engine/interpreter/hostcall.c
+ CC src/engine/interpreter/init.c
+ CC src/cpu/difftest/dut.c
+ CC src/cpu/cpu-exec.c
+ CC src/monitor/sdb/watchpoint.c
+ CC src/monitor/sdb/sdb.c
+ CC src/monitor/sdb/expr.c
+ CC src/monitor/monitor.c
+ CC src/utils/log.c
+ CC src/utils/rand.c
+ CC src/utils/state.c
+ CC src/utils/timer.c
+ CC src/memory/paddr.c
+ CC src/memory/vaddr.c
+ LD /home/why/Documents/ICS2021/ics2021/nemu/build/riscv32-nemu-interpre
$
```

```
1  # Sanity check
2  ifeq ($(wildcard $(NEMU_HOME)/src/nemu-main.c),)
3    $(error NEMU_HOME=$(NEMU_HOME) is not a NEMU repo)
4  endif
5
6  # Include variables and rules generated by menuconfig
7  -include $(NEMU_HOME)/include/config/auto.conf
8  -include $(NEMU_HOME)/include/config/auto.conf.cmd
9
10 remove_quote = $(patsubst "%",%,$(1))
11
12 # Extract variabls from menuconfig
13 GUEST_ISA ?= $(call remove_quote,$(CONFIG_ISA))
14 ENGINE ?= $(call remove_quote,$(CONFIG_ENGINE))
15 NAME    = $(GUEST_ISA)-nemu-$(ENGINE)
16
17 # Include all filelist.mk to merge file lists
18 FILELIST_MK = $(shell find ./src -name "filelist.mk")
19 include $(FILELIST_MK)
20
21 # Filter out directories and files in blacklist to obtain the final set of source files
22 DIRS-BLACKLIST-y += $(DIRS-BLACKLIST)
23 SRCS-BLACKLIST-y += $(SRCS-BLACKLIST) $(shell find $(DIRS-BLACKLIST-y) -name "*.c")
24 SRCS-y += $(shell find $(DIRS-y) -name "*.c")
25 SRCS = $(filter-out $(SRCS-BLACKLIST-y),$(SRCS-y))
26
27 # Extract compiler and options from menuconfig
28 CC = $(call remove_quote,$(CONFIG_CC))
29 CFLAGS_BUILD += $(call remove_quote,$(CONFIG_CC_OPT))
30 CFLAGS_BUILD += $(if $(CONFIG_CC_LTO),-flto,)
31 CFLAGS_BUILD += $(if $(CONFIG_CC_DEBUG),-ggdb3,)
32 CFLAGS_BUILD += $(if $(CONFIG_CC_ASAN),-fsanitize=address,)
33 CFLAGS  += $(CFLAGS_BUILD) -D__GUEST_ISA__=$(GUEST_ISA)
34 LDFLAGS += $(CFLAGS_BUILD)
35
36 # Include rules for menuconfig
37 include $(NEMU_HOME)/scripts/config.mk
38
39 ifdef CONFIG_TARGET_AM
40 include $(AM_HOME)/Makefile
41 LINKAGE += $(ARCHIVES)
42 else
43 # Include rules to build NEMU
44 include $(NEMU_HOME)/scripts/native.mk
45 endif
~
~
~
~
~
~
~
~/Documents/ICS2021/ics2021/nemu/Makefile[1]                    [make] un
```

总结

关于《计算机系统基础》习题课

- 教会大家 “计算机的正确打开方式”
  - 编程 ≠ 闷头写代码
  - 使用工具也是编程的一部分
    - version-control systems: git, svn, ...
    - build systems: make, cmake (C++), maven (Java), ...
    - shell: bash, zsh, ...

- 基本原则：任何感到不爽的事情都一定有工具能帮你
  - 如果真的没有，自己造一个的就会就来了
    - (不太可能是真的)
    - 但这将会是一份非常好的研究工作

End.

(RTFM & RTFSC)