**ssh：**secure shell, syntax: ssh username@hostname. 为防止客户端和服务器之间有恶意服务器，使用key pair认证更安全，服务器公钥和客户端公钥一致，则连接成功。 SSH uses TCP port 22 by default.

**私钥只能自己读写，公钥自己能读写，其他人能读; known\_hosts 是客户端保存已ssh过的计算机的公钥的地方,自己能读写，其他人能读**

连接上lab后，whoami and uname -a to check who you are logged in as, and where; also try hostname which just prints the machine name

three main authentication factors: something you know (password or PIN), something you have (physical key, digital key, ID card, passport) and something you are (biometrics).

known\_hosts is where SSH stores the public keys of computers you've already connected to

authority\_keys是在服务器中才有用，在客户端无用,know\_hosts在客户端有用，服务端无用

**Vagrant up**启动机器 vagrant ssh 连接虚拟机 vagrant halt 停止机器 Vagrant reload 停止+启动（用于配置更新） Vagrant destroy（删除虚拟机）

VagrantFile用ruby写, .vagrant文件夹用于跟踪虚拟机状态;支持多种虚拟化提供商，如 VirtualBox、VMware;可以通过ssh访问，主机与虚拟机存在共享文件夹

包管理器：apt apt search +软件名可以搜到软件不同版本 apt search --help可以提供更多操作

在 UNIX 和类 UNIX 系统中，**man** 命令用于查看手册页面（manual page）。手册页面按照不同的部分（sections）进行分类，每个部分用一个数字表示，代表了手册页面的内容类型。常见的部分包括：

Section 1: 用户命令（User Commands），通常包含了用户在终端中可以直接调用的命令的手册页面，比如 ls、cd、cp 等。

Section 2: 系统调用和内核接口（System Calls and Kernel Interface），包含了与操作系统内核交互的函数和系统调用的手册页面。

Section 3: 库函数（Library Functions），包含了 C 标准库和其他库的函数的手册页面，比如 printf、malloc 等。

Section 4: 设备和特殊文件（Devices and Special Files），包含了设备驱动程序、文件系统和特殊文件的手册页面。

Section 5: 文件格式和约定（File Formats and Conventions），包含了文件格式、配置文件、协议和约定的手册页面。

Section 6: 游戏和屏保（Games et. al.），包含了游戏、动画和屏幕保护程序的手册页面。

Section 7: 杂项（Miscellaneous），包含了各种杂项信息的手册页面，比如宏、协议、文件系统等。

Section 8: 系统管理命令和守护进程（System Administration tools and Daemons），包含了系统管理、维护和配置的命令和守护进程的手册页面。

Lab到虚拟机的过程：1、主机（lab machine）上运行 SSH 服务器（sshd）并监听默认端口 22。 2、使用网络连接到 Vagrant 虚拟化工具，Vagrant 默认端口为 2222。 3、Vagrant 虚拟机（guest）上也运行 SSH 服务器（sshd）并监听默认端口 22。 4、通过端口转发，可以将主机（lab machine）上的 SSH 服务器（sshd）的端口 22 转发到 Vagrant 虚拟机（guest）上的 SSH 服务器（sshd）的端口 22

dpkg-query -S [command] 用来查找在 Debian 系统中已安装软件包中的文件路径的命令

dpkg-query -S /bin/ls 将返回包含 /bin/ls 的文件的软件包名称，以及相关的版本信息。

**Debian文件系统：**

/bin stands for binaries, that is programs that you can run. Have a look with ls /bin: there will be a lot of commands in here, including ls itself. Indeed you can find out where a program is with which, so which ls will show you /usr/bin/ls for example.

/usr is a historical accident and a bit of a mess. A short history is on [this stackexchange question](https://askubuntu.com/questions/130186/what-is-the-rationale-for-the-usr-directory) but essentially, in the earliest days,

/bin was only for binaries needed to start the system - or at least the most important binaries that needed to live on the faster of several disk drives, like your shell.

/usr/bin was where most binaries lived which were available globally, for example across all machines in an organisation.

/usr/local/bin was for binaries installed by a local administrator, for example for a department within an organisation.

In any case, /usr and its subfolders are for normally read-only data, such as programs and configuration files but not temporary data or log files. It contains subfolders like /usr/bin or /usr/lib that duplicate folders in the root directory. Debian's way of cleaning this mess up is to make its /bin just a link to /usr/bin and putting everything in there, but in some distributions there are real differences between the folders.

see some files are green, but others are blue - this indicates the file type, green is an executable program, blue is a link to another file. Have a look with ls -l /bin: the very first character of each line indicates the file type, the main ones being - for normal file, d for directory and l for a so-called soft link.

/etc stores system-wide configuration files and typically only root (the administrator account) can change things in here. For example, system-wide SSH configuration lives in /etc/ssh.

/lib contains dynamic libraries - windows calls these .dll files, POSIX uses .so. For example, /lib/x86\_64-linux-gnu/libc.so.6 is the C library, which allows C programs to use functions like printf.

/home is the folder containing users' home directories, for example the default user vagrant gets /home/vagrant. The exception is root, the administrator account, who gets /root.

/sbin (system binaries) is another collection of programs, typically ones that only system administrators will use. For example, fdisk creates or deletes partitions on a disk and lots of programs with fs in their name deal with managing file systems. /sbin/halt, run as root (or another user that you have allowed to do this), shuts down the system; there is also /sbin/reboot.

/tmp is a temporary filesystem that may be stored in RAM instead of on disk (but swapped out if necessary), and that does not have to survive rebooting the machine.

/var holds files that vary over time, such as logs or caches.

/dev, /sys and /proc are virtual file systems. One of the UNIX design principles is that almost every interaction with the operating system should look to a program like reading and writing a file, or in short everything is a file. For example, /dev offers an interface to devices such as hard disks (/dev/sda is the first SCSI disk in the system, and /dev/sda1 the first partition on that), memory (/dev/mem), and a number of pseudoterminals or ttys that we will talk about later.

/proc provides access to running processes; /sys provides access to system functions. For example, on some laptop systems, writing to /sys/class/backlight/acpi\_video0/brightness changes the screen brightness.

The /vagrant folder is not part of the FHS, but is our convention for a shared folder with the host on Vagrant virtual machines

**Shell:**

单次tab补全文件名（前缀唯一），两次tab列出前缀相同的文件列表，^R + 命令的部分可以搜索之前执行过的命令. Cp \* dest 如果文件夹为空会报错

cp [-rfi] src dest

-r：递归复制，如果源目录是一个目录，则会递归地复制其内容及其子目录。如果不使用-r选项，当src为目录时，cp命令会报错。

-f：强制复制，如果目标文件或目录已经存在，则强制覆盖它们而不询问用户。

-i：交互模式，如果目标文件或目录已经存在，会提示用户确认是否覆盖。如果用户选择了覆盖，那么-i选项的作用就会被-f选项覆盖。

mv [-nf] src dest -f表强制 -n同名不覆盖.

Find可以查找文件位置

$在一个普通shell #在root下 %在C shell下 >字符串换行连续输入的标志

Linux philosophy:

Each program does one thing well. Programs should cooperate to perform larger tasks. The interface between programs should be text stream.

**Pipe：**

Ls -1｜head -n 5注意是1不是l. Ls -1|grep \* |sort|uniq

Grep ^e word.txt. 寻找txt中以e开头的内容 grep e$ word.txt 寻找txt中以e结尾的内容 grep -i表示忽略大小写

Redirect就是把输出到终端的东西又塞给另一个命令，比如用 > 把内容放到一个txt里 >表示覆盖重写 >>表示追加

$ COMMAND > FILE 2> FILE2：将命令的标准输出重定向到 FILE，将错误输出重定向到 FILE2。

$ COMMAND > FILE 2>&1：将命令的标准输出和错误输出都重定向到同一个文件 FILE。

$ COMMAND 2>&1 > FILE：这种形式是错误的，应该使用 $ COMMAND > FILE 2>&1 来正确地将标准输出和错误输出都重定向到同一个文件。

$ COMMAND > /dev/null：将命令的输出重定向到 /dev/null，这将忽略命令的输出，不会显示在终端上。

Program < infile standard input. Program > outfile standard output. program 2> file standard error

Command2 < (COMMAND1)会使command1的的输出变成一个临时文件，然后文件名作为参数喂给command2

**有时program不想输出到文件，可以用-代替然后做其他事情：pdftotext test.pdf -|grep software会把这个pdf的文本输出然后给grep执行其他操作**

文件名不要用-开头，不然执行命令时就要加./： cat ./-filename

Ls |tee filename。 Tee会将ls输出到终端的内容同时输入到filename文件里面

echo $(echo Hi | sed -e s/Hi/Hello/) $会将后面这一串的输出作为第一个echo的参数，所以会输出hello，或者‘(echo Hi | sed -e s/Hi/Hello/)’，没有美元符号会输出 (echo Hi | sed -e s/Hi/Hello/)

shell variable练习有设置变量的内容

command < filename: 这个命令将文件 filename 的内容作为标准输入提供给 command 命令。

command <(filename): 将文件内容以文件名为参数的形式喂给command。这个命令使用了进程替换（process substitution）的语法。换句话说，它会创建一个临时的命名管道，将 filename 的内容通过管道传递给 command。

command2 <(command1)：redirect the output of command1 to be used by command2

Command $(something) something执行完后将输出以string的形式作为参数传给command

grep -i忽略大小写，-v反向匹配

**正则表达式：**

? 匹配文件名中的 0 个或 1 个字符， \* 匹配零个或多个字符，+匹配一个或多个

^[a-zA-Z0-9\_-]{3,15}$

^ 表示匹配字符串的开头。 [a-zA-Z0-9\_-] 表示字符集，包含小写字母、大写字母、数字、下划线和连接字符 -。{3,15} 表示前面的字符集最少出现 3 次，最多出现 15 次，从而限制了用户名的长度在 3 到 15 个字符之间。  表示匹配字符串的结尾

正则表达式元字符和特性

字符匹配

普通字符：普通字符按照字面意义进行匹配，例如匹配字母 "a" 将匹配到文本中的 "a" 字符。

元字符：元字符具有特殊的含义，例如 \d 匹配任意数字字符，\w 匹配任意字母数字字符，. 匹配任意字符（除了换行符）等。

量词

\*：匹配前面的模式零次或多次。

+：匹配前面的模式一次或多次。

?：匹配前面的模式零次或一次。

{n}：匹配前面的模式恰好 n 次。

{n,}：匹配前面的模式至少 n 次。

{n,m}：匹配前面的模式至少 n 次且不超过 m 次。

字符类

[ ]：匹配括号内的任意一个字符。例如，[abc] 匹配字符 "a"、"b" 或 "c"。

[^ ]：匹配除了括号内的字符以外的任意一个字符。例如，[^abc] 匹配除了字符 "a"、"b" 或 "c" 以外的任意字符。

**\**d匹配任意一个阿拉伯数字（0 到 9）。等价于 [0-9]

\w匹配字母、数字、下划线。等价于 [A-Za-z0-9\_]

[\s\S] 匹配所有。

\s 是匹配所有空白符，包括换行，\S 非空白符，不包括换行。

\S \W \D则表示非空格，非字符，非数字

边界匹配

^：匹配字符串的开头。

$：匹配字符串的结尾。

\b：匹配单词边界。

\B：匹配非单词边界。

runoo+b，可以匹配 **runoob、runooob、runoooooob** 等，+ 号代表前面的字符必须至少出现一次（1次或多次）

runoo\*b，可以匹配 **runob、runoob、runoooooob** 等，\* 号代表前面的字符可以不出现，也可以出现一次或者多次（0次、或1次、或多次）

colou?r 可以匹配 **color** 或者 **colour**，? 问号代表前面的字符最多只可以出现一次（0次或1次

所谓特殊字符，就是一些有特殊含义的字符，如上面说的 runoo\*b 中的 \*，简单的说就是表示任何字符串的意思。如果要查找字符串中的 \* 符号，则需要对 \* 进行转义，即在其前加一个 \，runo\\*ob 匹配字符串 **runo\*ob**。

以下正则表达式匹配一个正整数，[1-9]设置第一个数字不是 0，[0-9]\* 表示任意多个数字：/[1-9][0-9]\*/

设置 0~99 的两位数，可以使用下面的表达式来至少指定一位但至多两位数字/[0-9]{1,2}/

\* 和 + 限定符都是贪婪的，因为它们会尽可能多的匹配文字，只有在它们的后面加上一个 ? 就可以实现非贪婪或最小匹配。

例如，您可能搜索 HTML 文档，以查找在 h1 标签内的内容。HTML 代码如下：<h1>RUNOOB-菜鸟教程</h1>

贪婪：下面的表达式匹配从开始小于符号 (<) 到关闭 h1 标记的大于符号 (>) 之间的所有内容。/<.\*>/

非贪婪：如果您只需要匹配开始和结束 h1 标签，下面的非贪婪表达式只匹配 <h1>。/<.\*?>/

也可以使用以下正则表达式来匹配 h1 标签，表达式则是：/<\w+?>/

指明两种匹配模式之间的一个选择，用｜。要匹配 |，请使用 \|

匹配单词边界稍有不同，但向正则表达式添加了很重要的能力。单词边界是单词和空格之间的位置。非单词边界是任何其他位置。下面的表达式匹配单词 Chapter 的开头三个字符，因为这三个字符出现在单词边界后面：/\bCha/

\b 字符的位置是非常重要的。如果它位于要匹配的字符串的开始，它在单词的开始处查找匹配项。如果它位于字符串的结尾，它在单词的结尾处查找匹配项。例如，下面的表达式匹配单词 Chapter 中的字符串 ter，因为它出现在单词边界的前面：/ter\b/

下面的表达式匹配 Chapter 中的字符串 apt，但不匹配 aptitude 中的字符串 apt：/\Bapt/

字符串 apt 出现在单词 Chapter 中的非单词边界处，但出现在单词 aptitude 中的单词边界处。对于 \B 非单词边界运算符，不可以匹配单词的开头或结尾，如果是下面的表达式，就不匹配 Chapter 中的 Cha：\BCha

**GIT 。gitignore可以隐藏不想提交的文件，git status就看不出来了**

Git init 初始化 git add …添加文件到暂存区，如果想将暂存区恢复到git add以前，可以使用git reset 或者git reset –mixed

如果在add+commit后回退，可以使用git reset --hard HEAD^ or git reset –soft HEAD^, hard会回退到add以前（且该条commit记录直接消失）。soft会回退到commit以前，如果此时git status，会提示你要commit

Git fetch会把远程仓库新的commit更新到本地仓库，但是没有合并，需要用git pull拉下来

Git clean:新创建的文件还没有add，如果想一次性删除，就git clean -dfx（感觉手动删除也可以啊）

Git checkout filename文件修改后还没有add，用这个命令可以让文件回到未修改的状态

Git reset <filename>用于取消将指定文件添加到暂存区的操作，将文件恢复到最后一次提交或者最后一次 git add 之前的状态

Git reset –hard HEAD + git clean -dfx就会完全删干净上次commit的东西？但是我这边不行git reset --hard HEAD^这个可以

Git checkout HEAD --filename假如文件修改后已经add但没有commit，就使用这个命令让文件回到修改前的状态，并且git status会显示树干净

Git checkout HEAD~1可以让你切换到最近一次commit前的版本，然后使用 git checkout main切回最新

在commit后，用git revert HEAD可以回退到上一个版本，但是会再commit一次，留下记录，而reset就直接删除上一次commit记录

git revert是用于“反做”某一个版本，以达到撤销该版本的修改的目的。比如，我们commit了三个版本（版本一、版本二、 版本三），突然发现版本二不行（如：有bug），想要撤销版本二，但又不想影响撤销版本三的提交，就可以用 git revert 命令来反做版本二，生成新的版本四，这个版本四里会保留版本三的东西，但撤销了版本二的东西。

Git tag就是给当前已经commit到内容打上一个标签，如果打上后修改文件再次git tag，相当于之前的commit有两个tag，所以必须是commit之后在tag

用法：git tag mak，创建名为mak的标签，git show mak查看该标签内容，git tag -d mak删除标签

Git clone src dest 如果不写dest，文件夹名就是src，如果写了就是dest

关于两人仓库合并的方式(bob,alice)：

--Git format-patch origin/main –to=alice的email 备注信息 然后git send\_email发给alice。Alice用git -am ···去commit，现在一致了，但是bob和alice的commit哈希号是不一样的

--或者Alice add bob’s repo as a remote, git remote add bob ~bob/coursework,拉下来查看后没有问题就 git pull bob main

当两人就同一个仓库先后push，第二个push的人就会报错，如果是不同文件或者不同行，解决方法是rebase或者merge，如果是同文件同行，就手动改冲突的那个文件，然后重新add commit

总体来说：

git remote and git clone to work with other people git fetch or git pull or patch files to get other peoples work

git merge or git rebase to integrate changes git push to send work back to a forge

新建分枝并切换：git checkout -b newbranch == git branch newbranch + git checkout newbranch,分支合并后可以用git branch -d newbranch删除分支

Git rebase -i可以合并已提交的commit

Git rebase 和git merge都可以合并分支，假设现在在mak分支，要合并到main

Merge: git checkout main + git merge mak

Rebase: git rebase main

git merge也可以写成git merge --ff，其中参数--ff意为fast-forward。该命令指的是把HEAD指针指向要合并分支的头，完成一次合并。git merge --no-ff中的--no-ff意为强行关掉fast-forward，所以在使用这种方式后，分支合并后会生成一个新的commit，这样，在使用git log从提交历史上就可以看到分支信息

Git cherry-pick可以将其他分支的特定commit合并到当前分支：

用git log确定想要从其他分支中复制的提交的提交 ID，然后git cherry-pick <commit\_id>**解决冲突**：如果在 cherry-pick 过程中出现冲突，Git 会停止 cherry-pick 进程并提示你解决冲突。你需要手动解决这些冲突，并使用 git add 将解决后的文件标记为已解决。**完成 cherry-pick**：当你解决了所有冲突后，使用 git cherry-pick --continue 命令继续 cherry-pick 进程。**终止 cherry-pick**：如果你想取消 cherry-pick 进程并且不应用任何提交，可以使用 git cherry-pick --abort 命令。

**FILE PERMISSION**

图示

描述已自动生成

S表示普通用户可以用root权限去执行特定文件，t表示这个目录或文件只能追加内容，通常用于日志

用户密码在/etc/shadow文件中， passwd可以改用户密码

更改文件所属用户或者组：

chown [选项] <新所有者>[:<新所属组>] <文件/目录> PS：冒号后面是新组，前面是新所有者，加参数-R表示递归更改该目录下所有文件和目录

更改文件权限：Chmod 744 filename：r4w2x1

修改某一类或某些类，分别三列分别属主u，属组g，其他o，所有a

chmod u=rw 1.txt

chmod uo=rw 1.txt

chmod a=r 1.txt 使用a可以改三类用户权限，使其都只有读r权限

chmod g=r,o=--- 1.txt

只操作某类用户的某些位权限

chmod u+w 1.txt 属主加上写w权限

chmod a-x 1.txt

chmod +x 1.txt 所有用户均增加执行权限x，a可以省略

chgrp只修改文件或目录所属组，而不修改所有者

Linux实际上可以用多个超级用户，用namespace可以划分不同root用户的权限

在 Linux 中，capabilities（能力）是一种机制，允许对于传统的 UNIX 权限模型进行细粒度的扩展。传统的 UNIX 权限模型仅包含了用户和组的权限，而 capabilities 则允许程序在不必完全拥有超级用户权限的情况下执行一些特定的系统操作。capabilities 允许系统管理员将某些特权分配给普通用户或程序，而不必将它们提升为超级用户（root）。这样可以提高系统的安全性，因为程序只能访问其实际需要的权限，而不是整个系统。

/etc/passwd 文件包含了系统中每个用户的基本信息，包括用户名、用户 ID、主目录、登录 shell 等。

/etc/group 文件包含了系统中每个用户组的信息，包括组名、组 ID、组成员等。

sudo adduser NAME sudo addgroup USERNAME GROUPNAME

Make sure your terminal is running as brian and try a sudo ls. You will see a general message, you will be asked for your password, and then you will get the error brian is not in the sudoers file. This incident will be reported. (This means that an entry has been logged in /var/log/messages.)

**Shell Script**

开头：

For portable POSlX shellscripts: #! /bin/sh For less portable BASH scripts: #! /usr/bin/env bash

脚本通常都有：Include a #! Always use env. $? contains the exit code [ $? -eq 0 ]检测程序退出是否为0，等同于 test $? -eq 0

Sh file.sh 这样执行脚本不需要chmod,不然就chmod +x file.sh + ./file.sh

变量命名a=”mak”注意等号左右无空格，使用变量就是”${a}”，删除变量就是unset a

If you want your variable to exist in the programs you start as an environment variable: export a

如果想引用某个不存在的变量时报错，就要设置set -o nounset

一些特殊变量：ps输入参数时最好带上双引号，避免参数有空格引歧义

${0} : shellscript name ${1}:第一个脚本变量 ${#}:变量数量 ${@}将每个参数视为独立的字符串。即使参数中含有空格，也会被视为单独的字符串

手机屏幕截图

描述已自动生成文本

描述已自动生成${\*} 将所有参数视为单个字符串，参数之间用第一个字符分隔符（默认为空格）连接起来。参数中的空格不会被特殊对待，会保留在连接的字符串中。

If elif else

文本

描述已自动生成

“${var##\*/}”可以提取出带目录的文件中的文件名，比如：/usr/bin/file.txt 就会提取出file.txt

“$(basename “${var}”)”能达到同样效果，basename,dirname用法：

$ basename /path/to/file.txt file.txt basename path [suffix]suffix表示要删除的文件名后缀，可选

$ dirname /path/to/file.txt /path/to

In the beginning /bin was reserved for just system programs

/usr/bin for admin installed programs /usr/local/bin for locally installed programs /opt/bin for optional installed programs

/opt/local/bin for optional locally installed programs ~/.local/bin for a users programs

env 是一个用于在指定环境下执行命令或显示当前环境变量的命令。在 Linux 系统中，每个进程都有自己的环境变量列表，env 命令可以帮助你管理和修改这些环境变量 What env does is look through the PATH and tries to find the program specified and runs it.

脚本常用语法：

A;B run A then run B A |B run A and feed its output as the input to B. A && B run A and ifsuccessful run B A || B run A and if not successful run B

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

管道中的<<<用法：$ echo <<< "Hello, World!"

就会输出：Hello, World! 。<<< 可以在一些需要从字符串中读取数据的命令中非常有用，比如 read 命令。这样可以方便地将字符串作为输入，而不必通过临时文件或者管道

**Set -euo pipefail**

set -e makes the whole script exit if any command fails. This way, if you want to run a list of commands, you can just put them in a script with set -e at the top, and as long as all the commands succeed (return 0), the shell will carry on; it will stop running any further if any command returns nonzero. It is like putting || exit $? on the end of every command.

set -u means referencing an undefined variable is an error. This is good practice for lots of reasons.

set -o pipefail changes how pipes work: normally, the return value of a pipe is that of the last command in the pipe. With the pipefail option, if any command in the pipeline fails (non-zero return) then the pipeline returns that command's exit code.

**Building tool**

Makefile 中，phony target 是指不代表实际文件名的目标。它们通常用于执行一些命令或者依赖于其他目标的情况下。phony target 主要有两个作用：

执行命令：它们可以用来执行一些命令，比如清理临时文件、构建项目等，而不会生成对应的文件。

依赖关系：它们可以用来表示其他目标的依赖关系，确保在执行其他目标之前先执行它们

All: typically first rule in a file (or marked .default): depends on everything you'd like to build

clean: deletes all generated files install: installs the program"

如果编译过程中提示找不到源文件或者.o文件，apt-file search <name of file> to find out which package provides the file you're missing and install it

用sudo安装的软件通常在/usr上，而用户自己安装的则在根目录/.local

文本

描述已自动生成

$@是目标文件，$<是源文件，%.o:%.c表示生成与源文件同名的目标文件，因为makefile懂C语言，所以不写hello至zip之间的两行也可以

上面这张图只生成flowchart.pdf,下面这个可以生成批量，将.dot全部转成.pdf,figure会生成所有pdf的名字

表格

中度可信度描述已自动生成文本

描述已自动生成

Makefile的缺点：无法跟踪版本，不知道如何获取依赖项

**关于python：**

pip freeze | tee requirements.txt will list all the packages your using and what version they are and save them in a file called requirements.txt.

pip install -r requirements.txt will install them again!

**关于java：**

A Java Runtime Environment (JRE) contains only the java command, which is all you need to run java applications if you don't want to do any development. Many operating systems allow you to double-click jar files (at least ones containing a special file called a manifest) to run them in a JRE.

A Java Development Kit (JDK) contains the javac and jar tools as well as a JRE. This is what you need to develop in java.

**Maven**

mvn compile compiles the project.

mvn clean will remove all compiled files so the next compile will rebuild everything.

mvn exec:java run the compiled project

mvn compile test exec:java to recompile, and run your tests

mvn test runs the tests in src/test/java

mvn package creates a jar file of your project in the target/ folder.

mvn install install the JAR into your local JAR packages

update-alternatives是debian的软件版本管理工具，如果同个软件多个版本安装在debian中，用update-alternatives --config <name>可以设置软件的默认版本

**Debug**

Gdb;

B,run,n,p,

Strace trace what systemcalls a program uses, strace可以用正则获取你想要的系统调用，strace参数：

strace -tt -T -v -f -e trace=file -o /data/log/strace.log -s 1024 -p 23489

-tt：在每行输出的前面，显示毫秒级别的时间

-T：显示每次系统调用所花费的时间

-v：对于某些相关调用，把完整的环境变量，文件 stat 结构等打出来。

-f：跟踪目标进程，以及目标进程创建的所有子进程

-e：控制要跟踪的事件和跟踪行为，比如指定要跟踪的系统调用名称

-o：把 strace 的输出单独写到指定的文件

-s：当系统调用的某个参数是字符串时，最多输出指定长度的内容，默认是 32 个字节

-p：指定要跟踪的进程 pid，要同时跟踪多个 pid，重复多次 -p 选项即可。

Ltrace跟踪库函数调用

"strace, Ltrace, valgrind and gdb will help deal with most of the bugs you encounter But so will good defensive programming strategies

Always check the return code of functions

Always check assumptions

Always fix your compiler warnings"

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

**database**

SQL语句主要分为四类：DDL: 数据定义语言，用来定义数据库对象(数据库，表，字段) DML: 数据操作语言，用来对数据库表中的数据进行增删改DQL: 数据查询语言，用来查询数据库中表的记录DCL: 数据控制语言，用来创建数据库用户、控制数据库的访问权限.

**DDL**表操作：查询：show create table 表名; 添加列：ALTER TABLE 表名 ADD 字段名 类型 (长度) [ COMMENT 1 注释 ] [ 约束 ]; 修改数据类型：ALTER TABLE 表名 MODIFY 字段名 新数据类型 (长度); 修改字段名及数据类型：ALTER TABLE 表名 CHANGE 旧字段名 新字段名 类型 (长度) [ COMMENT 1 注释 ] [ 约束 ]; 删除字段: ALTER TABLE… DROP …; 修改表名：ALTER TABLE … RENAME TO … ; 删除表：DROP TABLE IF EXISTS …

添加主键：ALTER TABLE emp ADD CONSTRAINT 主键名PRIMARY KEY (id); 添加唯一约束：ALTER TABLE emp ADD CONSTRAINT uk\_emp\_id UNIQUE (id);删除： ALTER TABLE emp drop CONSTRAINT uk\_emp\_id;

ALTER TABLE 表名 ADD CONSTRAINT 外键名称 FOREIGN KEY (外键字段名) REFERENCES 主表 (主表列名) ;

**DML 添加数据：INSERT INTO 表名 (字段名1, 字段名2, ...) VALUES 1 (值1, 值2, ...); ps：数字不用加单引号，其他都要加，给全部字段添加数据可以直接INSERT INTO 表名VALUES 1 (值1, 值2, ...);**

**修改数据：UPDATE 表名 SET 字段名1 = 值1 , 字段名2 = 值2 , .... 1 [ WHERE 条件 ] ; 删除数据：DELETE FROM 表名 [ WHERE 条件 ] ;**

DQL：SELECT 字段列表 4

FROM 表名列表。 1

WHERE条件列表 2

GROUP BY分组字段列表。 3

HAVING分组后条件列表。 5

ORDER BY排序字段列表 6

LIMIT分页参数。 7

条件列表：<> 或 != 不等于 BETWEEN ... AND ... 在某个范围之内(含最小、最大值)IN(...) 在in之后的列表中的值，多选一LIKE 占位符模糊匹配(\_匹配单个字符, %匹配任意个字符) IS NULL 是NULL

执行顺序：where group by 聚合函数 having 分组之后，一般查询分组字段和聚合函数，查询其他字段无意义

SELECT 字段列表 FROM 表名 LIMIT 起始索引, 查询记录数 ;起始索引 = （查询页码 - 1）\* 每页显示记录数 查询第2页员工数据, 每页展示10条记录: select \* from emp limit 10,10;

CREATE TABLE student(

id INTEGER PRIMARY KEY auto\_increment,

name VARCHAR(10) UNIQUE NOT NULL,

age TINYINT UNSIGNED check(age>0&&age<=120),

status char(1) DEFAULT '1',

gender char(1)

);

添加外键：[CONSTRAINT] [外键名称] FOREIGN KEY (外键字段名) REFERENCES 主表 (主表列名) 删除外键：ALTER TABLE 表名 DROP FOREIGN KEY 外键名称;

删除、更新外键：ALTER TABLE 表名 ADD CONSTRAINT 外键名称 FOREIGN KEY (外键字段) REFERENCES 主表名 (主表字段名) ON UPDATE CASCADE ON DELETE CASCADE;

隐式内连接：SELECT 字段列表 FROM 表1 , 表12 **WHERE** 条件 ... ; 显式内连接SELECT 字段列表 FROM 表1 [ INNER ] JOIN 表2 **ON** 连接条件 ... ;

左外连接：SELECT 字段列表 FROM 表1 LEFT [ OUTER ] JOIN 1 表2 ON 条件 ... ; 右外连接：SELECT 字段列表 FROM 表1 RIGHT [ OUTER ] JOIN 表2 ON 条件 ... ;

自连接：SELECT 字段列表 FROM 表A 别名A JOIN 表A 别1 名B ON 条件 ... ;

第一范式：每个字段的值为不可再次拆分的最小数据单元。

第二范式：在满足第一范式的基础上，还要满足数据表里的每一条数据记录，都是可唯一标识的。而且所有非主键字段，都必须完全依赖主键，不能只依赖主键的一部分。如果存在不完全依赖，那么这个属性和主关键字的这一部分应该分离出来形成一个新的实体。

第三范式：在第二范式的基础上，确保数据表中的每一个非主键字段都和主键字段直接相关，也就是说，要求数据表中的所有非主键字段不能依赖于其他非主键字段。非主键字段之间不能有因果关系。

巴斯范式(BCNF) 例如有关系模式C(Cno, Cname, Pcno)，Cno, Cname, Pcno依次表示课程号、课程名、先修课。可知关系C只有一个码Cno，且没有任何属性对Cno部分函数依赖或传递函数依赖，所以关系C属于第三范式，同时Cno是C中的唯一决定因素，所以C也属于BC范式。如果在关系R中，U为主键，A属性是主键的一个属性，若存在A->Y，Y为主属性，则该关系不属于BCNF

第四范式：比如属性集合有学生id,学生所选科目id,学生所谈女朋友id.显然一个学生可以有多个所选科目，他也可以有多个女朋友，所以最好就是拆成俩表：学生-科目、学生-女朋友

SQL注入：输入密码时输入：' OR '1'='1 最终就会变成： select name, password from uses where username = ‘mak’ and password = ‘’ OR '1'='1’. 由于or后的条件永远成立，导致查询到所有密码

所以要用prepared statement

Sudo mysql

ALTER USER 'root'@'localhost' IDENTIFIED VIA mysql\_native\_password USING PASSWORD('root');