**一、Linux简介**

**1.1 基本概念**

Linux其实指的是内核，一个内核不是一套完整的操作系统。

通常我们使用的 linux 系统是一个集 linux 内核、工具集、各种库、桌面管理器、应用程序等一体的一个发布包 (发行版)

例如常见的发行版

Debian GNU/Linux 、Red Hat Linux 、Fedora Core、Ubuntu Linux

**1.2 应用领域**

Linux的应用领域非常广泛，主要可以分三个方向

服务器

嵌入式系统（各种单板计算机）

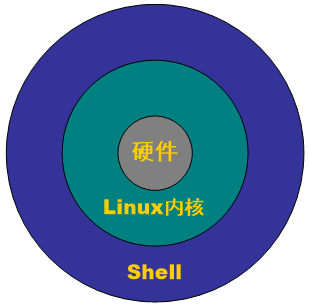
桌面系统

**二、Linux核心命令**

Linux是一个高可靠、高性能的系统，而很多时候只有直接使用Linux命令才能充分体现Linux系统的优越性。

**2.1 Shell**

Shell（外壳）是用户与操作系统内核之间的​​交互接口​​，本质上是一个​​命令行解释器​​。它接收用户输入的指令（命令或脚本），将其转换为操作系统能理解的内核指令，并将执行结果反馈给用户。简单来说，Shell 是用户操作计算机的“桥梁”，尤其在 Linux/Unix 系统中，它是系统管理和自动化任务的核心工具。



**2.1.1 作用**

​​命令解释与执行​​

用户输入的命令（如 ls、cd、cp）由 Shell 解析后，调用操作系统内核的功能完成操作（如文件管理、进程控制）。

​​脚本编程​​

支持将多条命令组合成脚本（.sh 文件），实现自动化任务（如批量文件处理、系统监控）。

​​环境管理​​

管理环境变量（如 PATH、HOME）、用户配置（如别名 alias）、函数定义等，定制个性化操作环境。

**2.1.2 类型**

不同 Shell 实现了不同的特性，常见的有



**2.1.3 命令格式**

通常一条命令包含 3 个要素：命令名称、选项、参数。命令名称是必须的，选项和参数都是可选项。

Command [-Options] Argument1 Argument2…

指令    选项       参数1     参数2

//示例：

ls -l /

**2.2 基本命令**

**2.2.1 文件管理类**

基本命令有pwd、ls、cd、touch、mkdir、rm、cp、mv、cat

文件类型：file

查找类：find、whereis

权限类：chmod、chown

示例

ls -l

ls -a

cd 目录 //目录可以是相对路径 也可以是绝对路径

cd ~ //回到家目录

cd - //回到之前的目录

file 1.c

file a.out

//查找类

whereis command //查找命令

find [搜索路径] [匹配条件] [操作] //搜索

find ./ -name "\*.c"

find ./ -type d

//权限

chmod [选项] [模式] 文件

chmod 支持两种模式：1.符号(+x) 2.数字(664)

chmod 777 share //修改文件 权限

chmod 644 file

chmod +x 1.sh //修改文件 权限 添加执行权限

chmod u+x 1.sh

chmod -R 777 share //递归修改

chown [选项] 新所有者[:新所属组] 文件/目录

chown hqyj t.c //修改文件 所有者

chown yan:yan ser.c //修改文件 所有者 所属组

chown -R hqyj pro //递归修改

//注意：修改权限，需要root权限

**2.2.2 系统管理类**

常用命令：sudo、man、su、passwd、reboot、shutdown

示例

man 1 ls //命令

man 2 close //系统调用

man 3 printf //库函数

su root //切换到root

su hqyj //切换到hqyj

//建议通过exit返回，(切换的本质是创建新会话)

passwd //修改密码

passwd hqyj //修改hqyj账号的密码

shutdown -h now //立即关机

reboot //重启

**2.2.3 归档与压缩**

tar、 zip

示例

tar -cvf ser.tar ser/\* //打包（不压缩）

tar -czvf ser.tar.gz ser/\* //打包（压缩gzip)

tar -cjvf ser.tar.bz ser/\* //打包（压缩bzip2)

tar -xvf ser.tar //解压

zip file.zip \*.txt //将txt文件压缩

zip pro.zip pro -r //将pro文件压缩

zip code.zip \* -r //将当前目录下所有内容压缩

unzip file.zip //解压

**2.2.3 进程管理**

常用命令 ps top kill

ps -ef //查看系统下的进程信息

kill [选项] [信号] [进程id] //给进程发信号

kill -11 pid //给进程发送 11号信号

kill -l //查看信号列表

kill pid //结束该进程

top //动态查看系统进程 ，输入q退出

**2.2.3 软件管理**

Linux系统主要支持RPM和Deb两种软件包管理工具，由于ubunut采用Deb软件包管理工具，因此只介绍Deb软件包管理工具。

apt、dpkg

示例

apt工具

apt update //跟新软件源

apt install 软件名

//安装

apt remove 软件名 //卸载

apt clean //清缓存

//常用的软件： ssh mosquitto、 tftpd-hpa、apache2

dpkg工具

dpkg -i xxx.deb

dpkg -r 软件名

**2.2.3 网络相关**

ifconfig、 ping 、nc、netstat、ufw、service、nmcli

ifconfig //查看网络情况

ping 是 TCP/IP 协议栈中的一个​​网络诊断工具

ping www.baidu.com

ping 192.168.43.124 // ctrl + c

nc命令的核心能力是通过网络在两个端点之间建立双向通信

-l 监听

-t tcp

-u udp

-p 端口

nc -u -l 2222 //创建udp服务器

nc -u 192.168.2.101 2222 //访问udp服务器

nc -t -l 6666 //创建tcp服务器

nc -t 192.168.2.101 6666 //访问tcp服务器

netstat网络状态查看工具

netstat [选项] [参数]

netstat -ltn

netstat -lun

参数说明

-l:监听 -u:udp -t:tcp -n:禁用域名解析和端口服务名转换

service是服务器管理工具

service [选项] [服务] [动作]

sudo service mosquitto status //查看mqtt服务器 状态

sudo service ssh restart //重启ssh服务器

sudo service ssh stop //停止ssh服务器

ufw是一个简化的防火墙工具

ufw [选项] [规则]

sudo ufw status //产看状态

sudo ufw enable //启动

sudo ufw disable //关闭

sudo ufw allow 22 //允许22端口

sudo ufw allow ssh //

sudo ufw deny 22 //组织22端口

nmcli 是网络管理器的命令行工具

nmcli device status

//用于解决ubuntu20.04 网络图标消失

sudo nmcli network off

sudo nmcli network on

**三、Shell特殊符号**

在 Shell 命令中，特殊符号是实现高级操作的核心工具。它们赋予 Shell 强大的灵活性和表达能力。以下是常见的特殊符号

**3.1 重定向符号**

用于控制命令的​​输入/输出流向​​，将标准输出（stdout）、标准错误（stderr）或输入（stdin）重定向到文件或其他命令

|  |  |  |
| --- | --- | --- |
| 符号 | 作用 | 示例 |
| > | ​  **​覆盖重定向​**  ​：将命令的标准输出覆盖写入文件（若文件不存在则创建）。 | ls > files.txt（将当前目录文件列表写入files.txt，覆盖原有内容） |
| >> | ​  **​追加重定向​**  ​：将命令的标准输出追加到文件末尾（不覆盖原有内容）。 | echo "new line" >> log.txt （向log.txt末尾添加一行） |
| < | ​  **​输入重定向​**  ​：将文件内容作为命令的标准输入。 | sort < unsorted.txt  （对unsorted.txt文件内容排序） |
| 2> | ​  **​错误重定向​**  ​：将命令的标准错误（stderr）覆盖写入文件。 | ls non\_existent\_file 2> error.log  （将“文件不存在”错误写入error.log） |
| &> | ​  **​混合重定向​**  ​：将标准输出和标准错误同时重定向到文件。 | command &> all\_output.log  （所有输出（正常+错误）写入all\_output.log） |

**3.2 管道**

符号 | 用于连接两个命令，将​​前一个命令的标准输出​​作为​​后一个命令的标准输入​​，实现数据流的传递。

ps -ef | grep "a.out"

cat 1.txt | wc -l

**3.3 通配符**

用于匹配文件名或路径的模式，简化批量操作。

|  |  |  |
| --- | --- | --- |
| 符号 | 作用 | 示例 |
| \* | 匹配任意字符（包括空字符），0 次或多次。 | ls \*.txt  （匹配所有以.txt 结尾的文件） |
| ? | 匹配任意单个字符（不能是空）。 | ls file?.txt  （匹配file1.txt、fileA.txt等） |
| [abc] | 匹配方括号内的任意一个字符。 | ls file[0-9].txt  （匹配file0.txt到file9.txt） |
| [a-z] | 匹配指定范围内的任意字符（如小写字母）。 | ls file[a-z].txt  （匹配filea.txt到filez.txt） |
| [^abc] | 匹配​  **​非​**  ​方括号内的任意字符（取反）。 | ls file[^0-9].txt  （匹配非数字结尾的file.txt） |

**3.4 命令替换**

` 符号 用于将​​命令的执行结果​​替换到当前命令中（相当于将命令的输出作为参数）。

rm `find ./ -name "\*.txt"` //将查询的结果 作为 rm的参数

**3.5 后台运行符**

符号 & 用于将命令放到​​后台运行​​，用户可继续输入其他命令（不阻塞当前终端）

./ser & //后台执行 ser程序

**3.6 变量替换**

$ 最核心的用途是​​获取变量的值​​。Shell 中的变量存储了字符串或数值，通过 $变量名 可以读取其内容。

name="Alice"

echo $name # 输出：Alice（直接通过 $ 获取变量值）

echo $PATH # 输出 环境变量PATH

**四、Make工具**

**4.1 基本概念**

Make 工具是软件开发中​​自动化构建（Build Automation）​​的核心工具之一，其核心功能是通过解析描述构建规则的文件（通常是 Makefile），自动判断项目中哪些文件需要重新编译或处理，并执行相应的命令，最终生成目标产物。

**4.1.1工作原理**

Make 工具通过读取 Makefile 中的规则（target: prerequisites），构建项目文件的依赖关系图。

Make 工具通过比较文件的最后修改时间（时间戳）判断是否需要重新构建目标。

对于需要更新的目标，Make 工具会按顺序执行 Makefile 中定义的命令,命令可以是任意 Shell 操作（如编译、链接、复制文件、运行测试等）。

**4.1.2 Makefile**

由于Make工具是根据Makefile来执行的，因此Makefile的编写便是构建自动化编译的主要内容。

**4.2 Makefile编写**

**4.2.1 基本结构**

target1: prerequisites ...

[tab] command

command

target2: prerequisites ...

[tab] command

command

target:目标，可以是伪目标

prerequisites：依赖

command：命令

注意：

命令必须以Tab键缩进，不能用空格

伪目标用.PHONY来声明

**4.2.2 执行方式**

make 默认执行第一个目标，如果其依赖更新了则执行

make target2 执行指定目标

**4.2.3 变量**

Makefile 支持变量定义，避免重复代码。变量通过 $(变量名) 引用。

CC = gcc # 编译器变量

CFLAGS = -Wall -O2 # 编译选项（-Wall 开启警告，-O2 优化）

LDLIBS = -lm -lpthread #库

Target = app #目标

Object = xx.o #依赖

除了自定义变量，还有预定义变量，常用的预定义变量有

$@：当前目标名。

$<：当前目标的第一个依赖。

$^：当前目标的所有依赖（用空格分隔）。

$?：所有比目标新的依赖（仅更新需要的依赖）。

**4.2.4 模式规则**

用 % 作为通配符，匹配一类文件（如所有 .c 文件编译为 .o 文件）。

示例

%.o: %.c # 所有 .o 文件依赖 同名.c文件

$(CC) $(CFLAGS) -c $< -o $@

**4.2.5 自动推导规则**

Make 内置了一些默认规则（如 .c → .o，.o → 可执行文件），可以省略部分显式规则，甚至能识别变量

**4.2.6 函数**

Makefile 提供了丰富的函数，用于文件列表处理、字符串操作等。常用函数：

wildcard：获取符合模式的文件列表（如 $(wildcard \*.c) 获取所有 .c 文件）。

patsubst：替换文件扩展名（如 $(patsubst %.c, %.o, $(wildcard \*.c)) 将所有 .c 转为 .o）。

dir/notdir：提取文件路径或文件名。

示例

SRCS = $(wildcard \*.c) # 所有 .c 文件

OBJS = $(patsubst %.c, %.o, $(SRCS)) # 所有 .o 文件

app: $(OBJS)

$(CC) $^ -o $@

makefile 示例1

app:app.c

    gcc app.c -o app

test:

    ./app

makefile 示例2

app:main.o queue.o tree.o

    gcc main.o queue.o tree.o -o app

main.o:main.c

    gcc -c main.c -o main.o

queue.o:queue.c

    gcc -c queue.c -o queue.o

tree.o:tree.c

    gcc -c tree.c -o tree.o

makefile 示例3

app:main.o queue.o tree.o

    gcc $^ -o $@

%.o:%.c

    gcc -c $< -o $@

clean:

    rm \*.o app -rf

makefile 示例4

app:main.o queue.o tree.o

    gcc $^ -o $@

clean:

    rm \*.o app -rf

makefile 示例5

.PHONY:clean

CFLAGS = -std=c99

CC = gcc

LDLIBS = -lm

Target = app

Object = main.o queue.o tree.o

$(Target):$(Object)

$(CC) $^ -o $@ $(LDLIBS)

clean:

rm \*.o app -rf

makefile 示例6

.PHONY:clean

CFLAGS = -std=c99 -I ./include #编译选项

CC = gcc #编译器

LDLIBS = -lm #指定库名

Target = app

SRCS = $(wildcard ./src/\*.c) # 所有 .c 文件

Object = $(patsubst %.c, %.o, $(SRCS)) # 得到 .o 文件

$(Target):$(Object)

$(CC) $^ -o $@ $(LDLIBS)

clean:

rm ./src/\*.o $(Target) -rf

**作业**

1. 练习 linux命令

2. 编写一个 Makefile