嵌入式 Linux 操作系统 第四讲 Linux 环境程序设计

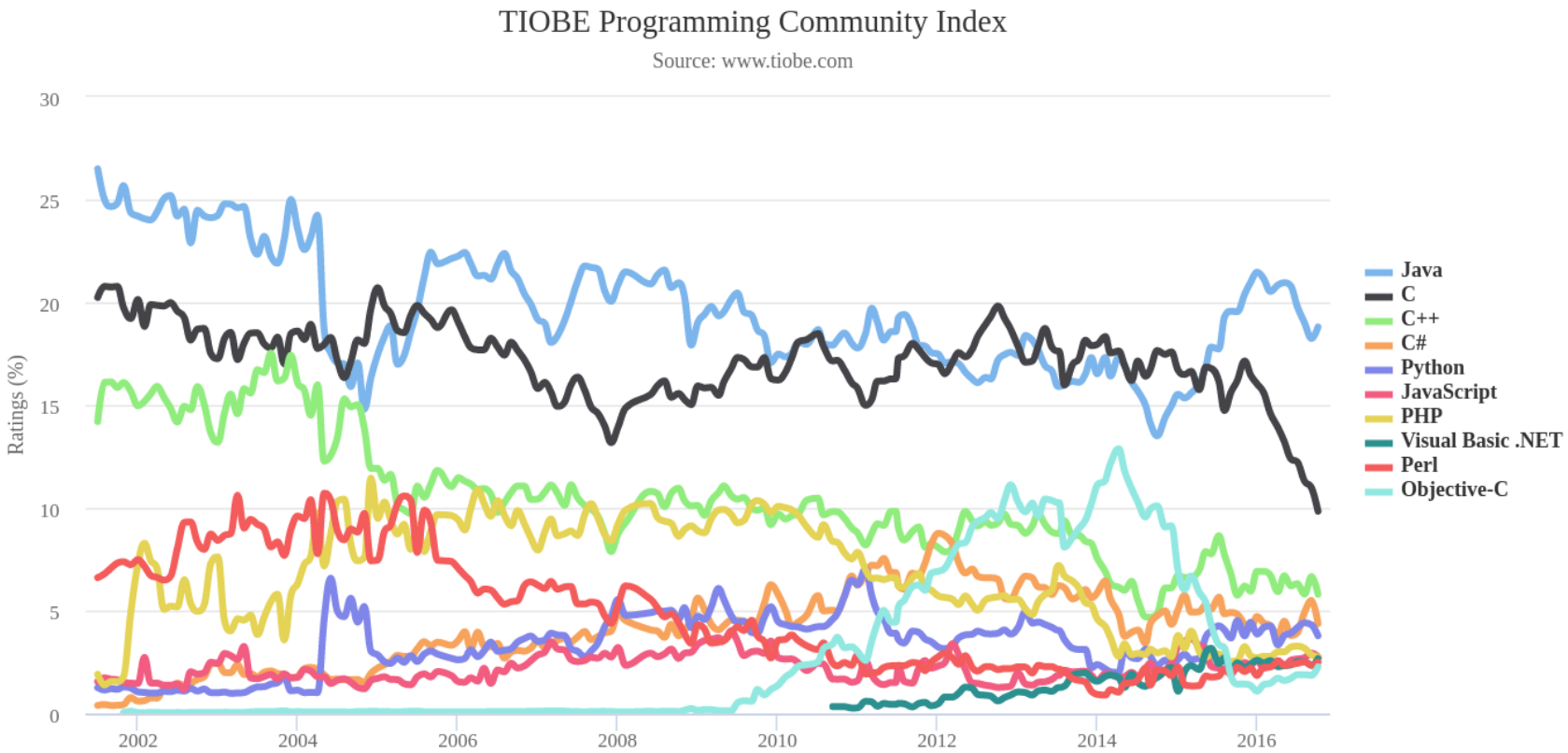
杨延军
yangyj.ee@gmail.com

北京大学

2016 年

主要内容

程序设计语言的使用统计



脚本语言简介

- 不需要对源程序进行编译，而通过解释器运行的程序
- 简单易学，开发快捷，语法灵活
- 适合作为一种“胶水”语言

示例

```
#!/usr/bin/python
myString = 'Hello World!'
print myString
```

Perl 脚本简介

- 最初是 Unix 下的系统管理工具，后来发展成全面的脚本编程语言
- 主要用途
  - 文本数据处理
  - 网络编程
  - 数据库
  - 图像处理等

Python 脚本简介

- 比较“高级”的程序设计语言
- 模块丰富，提供对大型程序的支持
- 采用缩进的方式区分模块
- 支持面向对象的程序设计

Shell 脚本简介

- Shell 有很强的编程能力，可以利用 shell 解释脚本执行较复杂的任务。
- 应用非常广泛，很多 Linux 命令实际就是一个 shell 脚本

例如 gunzip

```
#!/bin/bash
exec gzip -d "$@"
```

- 语法元素可以直接在命令行使用

主要内容

Shell 变量

特定 Shell 变量

- 命令行参数：$0，$1，$2,...，$9, shift 命令
- $# (参数个数)
- $* （ 所有参数)
- $@ (与$* 类似，但如果写成“$@”, 则相当于“$1”,“$2”...)

自定义变量

- 例如：CC=arm-linux-gcc

条件测试

- 两种格式：`test cond`, `[ cond ]`
- 文件状态测试：-d（目录），-f（正规文件），-L（符号链接），-r（可读），-w（可写），-x（可运行），-s（非空）-u（有 suid 位）
- 逻辑与：-a，逻辑或-o
- 字符串测试：=，!=，-z（空），-n（非空）
- 数值测试：-eq（相等），-ne（不等），-gt（大于），-lt（小于），-le（<=），-ge（>=）

示例

```
test -w a.txt    [ -d usr ]
```

条件分支

判断

if ... then ...
elif ... then ...
else ...
fi

- 表达式作为条件：
  `if [ "10" -lt "12" ] then`
- 命令作为条件：
  `if grep 'Dave' data.file > /dev/null 2>&1`

多重分支

格式

```
case 表达式 in
模式1)
    命令
    ;;
模式2)
    ;;
*)
    ;;
esac
```

循环控制

- for 循环：for 变量 in 列表 do 命令 done

示例

```
for loop in `ls`
do
    echo $loop
done
```

- until 循环：until 条件 do 命令 done
- while 循环：while 条件 do 命令 done
- break 和 continue

函数

示例

```
#!/bin/bash
function hello
{
    echo “Hello $1!”
}
hello Alan
hello Bill
```

- 参数使用 $1...$9

脚本中常见的命令

- sed 非交互性文本流编辑器

示例

```
sed -n '/night/'p quote.txt
sed 's/^M//g' dos.txt
```

- awk 从格式化报文中提取数据

示例

```
awk -F@ '{print $1}' proxy.txt
awk '{if($4~/Brown/) print $0}' grade.txt
```

主要内容

Linux 环境 C 语言开发的主要特点

- 一般不使用集成的开发环境
- 使用喜欢的编辑器
- 命令行的编译器和调试器
  - 编译器 Gcc
  - 调试器 gdb
- 操作系统相关部分与 Windows 差别较大

main 函数回顾

```
int main(int argc, char * argv[])
{
}
```

- argc 命令行参数个数
- argv 命令参数数组
- 示例：./a.out arg1 arg2

文件 IO

- 文件描述符
- open 系统调用与 open 函数

```
int open(const char *pathname,
int oflag, ...)
```

- read 函数

```
ssize_t read(int filedes, void *buf,
size_t nbytes);
```

- write 函数 `ssize_t write(int filedes, const void *buf, size_t nbytes);`

文件 IO 示例

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>

main()
{
        int fd,i;

        fd = open("/dev/dleds", O_RDWR);
        if (fd > 0) {
                i = 3;
                if(write(fd, &i, 4)==-1)
                        printf("%s\n",strerror(errno));
                close(fd);
        }
}
```

fork 函数

```
pid = fork();
if(pid == -1){
    fprintf(stderr, "Fail to fork!\n");
    exit(13);
} else if ( pid == 0 ) { // Child
    execlp("telnet", "telent", "ytht.net", NULL);
} else {
    wpid = wait(&status);
    if ( wpid == -1 ) {
        printf("error!\n");
        return 1;
    } else if ( wpid != pid)
        abort();
    else
        if ( WIFEXITED(status) ) printf("Exited!\n");
}
```

select 函数

```
tv.tv_sec = 0;
tv.tv_usec = 100;
do{
    FD_ZERO(&rset);
    FD_SET(0, &rset);
    FD_SET(fd, &rset);
    if (select (fd +1, &rset, NULL, NULL, &tv) < 0) {
        printf("select error!\n");
        return -1;
    }
    else {
        if (FD_ISSET(fd, &rset)) {
            i = read(fd, buf, 1023);
            if (i ==0) return 0;
            buf[i] = 0;
            printf("%s", buf);
        }
    }
}while(1);
```

gcc 编译器简介

- 基本用法：gcc -o myprog myprog.c
- -c 只编译不链接
- -g 包含调试信息
- -Idir 添加包含文件搜索路径
- -Ldir 添加库文件搜索路径
- -lfoo 把 libfoo 链接到目标文件
- -O 优化参数，常用 O2，O3，Os
- -Wall 打开所有的警告信息
- -static 只用静态库链接

gcc 编译过程

- 预处理 gcc -E hello.c -o hello.i
- 编译 gcc -S hello.i -o hello.s
- 汇编 gcc -c hello.s -o hello.o
- 链接 gcc hello.o -o hello

编程中常见错误消息

- undefined reference
  - gcc 参数-l 有关，例如-lqt 代表连接 libqt.so
  - 查看库中是否有某个函数的定义用 nm 命令
- cannot find -lts，gcc 的-L 参数制定库位置
- error while loading shared libraries: ...，LD_LIBRARY_PATH 路径设置的问题

Makefile 简介

- 是由 GNU make 来解释的一个脚本
- 可以智能地对包含多个文件的项目进行编译，处理好依赖关系。

示例

```
all: myprog
myprog: init.o main.o
<TAB>gcc -o myprog init.o main.o
main.o: main.c myprog.h
<TAB>gcc -Wall -c -o main.o main.c
init.o: init.c myprog.h
<TAB>gcc -Wall -c -o init.o init.c
clean:
<TAB>rm myprog *.o -f
```

Makefile 常见例子

```
CC=gcc
PROGRAM=gothello
CFLAGS=-O2 -Wall -ansi -pedantic

SRC := $(wildcard *.c)
OBJS := $(patsubst %.c,%.o, $(SRC))
HEADERS := $(wildcard *.h)

othello: $(OBJS)
    $(CC) $(CFLAGS) $(OBJS) -o $(PROGRAM)

%.o : %.c $(HEADERS)
    $(CC) $(CFLAGS) -c $< -o $@

clean:
    rm -f $(PROGRAM) $(OBJS)
```

用 gdb 调试程序

- 程序编译必须使用-g 选项
- 三种运行方式
  - 以可执行文件为参数运行
    gdb a.out
  - Attach 到特定的 PID。
    (gdb) attach 43193
  - 打开 core dump 文件查看
    gdb -c core
    shell 控制 core 文件生成: ulimit -c 10000

gdb 常用命令

- 断点：break 行号/函数名
- 条件断点：if cond
- 取消断点用 clear
- 显示变量和表达式：print
- 显示源文件：list
- 单步调试：step，next

用 vi 开发 C 程序

- ctags * -R 生成 tags 文件
  支持从标号跳转到定义处 Ctrl+], Ctrl+t
- cscope 功能比较强大，数据保存在 cscope.out 文件。可以用来查看 Linux 内核代码
- 与 C 语言编写相关的 vim 命令：帮助和补全

自动生成 Makefile

- 编写 Makefile.am 文件
- 用 autoscan 工具生成 configure.in 的原型，生成 configure.scan 文件，需要手工修改以适应实际需求
- 用 aclocal 工具生成 aclocal.m4，aclocal.m4 是 automake 需要的宏的定义文件，它根据 configure.in 生成
- 用 autoconf 工具生成 configure 执行脚本
- automake -a 生成 Makefile.in

标准 GNU 程序的编译过程

- 执行 configure 脚本生成 Makefile
- `./configure --help` 可以得到 configure 脚本所支持的参数
- Makefile 的目标：
  make install
  make clean
  make dist-bzip2

GNU 编译工具优点

- 适合源代码比较多的情况
- 自动进行配置，生成 Makefile，而不必让用户修改 Makefile
- 自动生成测试包
- 自动生成发行包
- 自动生成动态库
- 跨平台性好

GNU 程序发布

- 程序包的常见格式：rpm，tar.gz, tar.bz2, tgz，deb
- 源码包的使用方法：参考 README、INSTALL 文件
- 哪里去找源码：Google，sourceforge，freshmeat ...

GNU 编码风格

- 命令行参数：version, help 等
- 图形界面风格
- 编码风格、命名规则、注释等
- 文档风格、帮助等
- 软件配置方式等
- http://www.gnu.org/prep/standards/standards.html

主要内容

交叉编译原理

- A 机编译，B 机运行
- 提高编译速度，减少开发时间
  - 同平台：速度快的机器给速度慢的机器编译
  - 不同 OS：Linux 平台编译 Win32 平台的程序
  - 不同体系：X86 平台编译 ARM 平台的程序

交叉编译环境的建立

- 采用预编译的交叉编译器
  下载-> 解压-> 放到正确的目录
- 从源码进行编译
  - 主体工具：glibc、 binutils、 gcc
  - 可以对交叉编译环境进行“微调”
  - 需要对开源代码和编译器有深入的了解
- 使用自动编译脚本 (crosstools-ng)

arm 平台的交叉编译器

- arm-linux-gcc
  arm-linux-gcc hello.c –o hello_arm
- arm-elf-gcc
- arm-none-linux-gnueabi-gcc
- 交叉编译 GNU 发行程序
  configure 脚本的 host 和 target 参数设置
- 其它交叉编译的方法

主要内容

版本控制系统

- CVS(Concurrent Versions System) 多人协作的程序开发方式
- 记录软件开发过程的不同阶段
- 阶段性的版本标记
- 比较不同版本的区别
- 回复到早期的版本
- Subversion, Git, Source Safe, Rational

集中式的版本管理

- 每个开发人员都可以获得最新代码
- 修改代码后提交到服务器
- 如果出现多人修改到同一行代码的时候会出现冲突，需要后提交的人进行修改
- 版本库操作必须联网进行

分布式的版本控制

- 每个人都获得全部的版本信息，内容与服务器相同，可以互相替换
- 不必连接服务器就可以提交修改
- 提交的修改仍然需要在连接的时候同步到服务器
- 可以方便的进行分支与融合
- 更加高效和灵活的工作方式

Git 简介

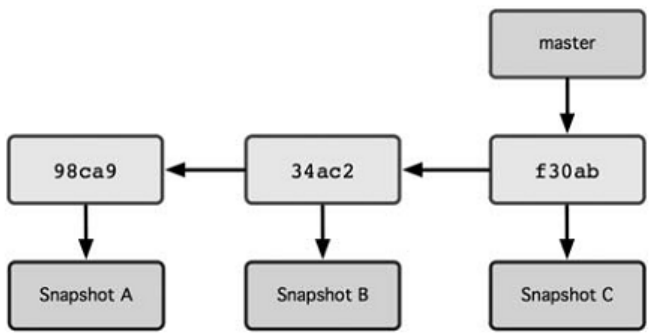
- 最初为 Linux 内核开发所设计的版本控制系统
- 执行效率高，提高开发人员的工作效率
- 分布式系统，可以适合非常大的开源项目
- 非常多的开源软件选择 git 作为版本控制系统

Git 初步

- 初始化代码库：git init，git clone
- 添加新文件或者修改内容：git add
- 提交到版本控制系统：git commit
- 查看提交历史：git log
- 与服务器交互：git push, git pull

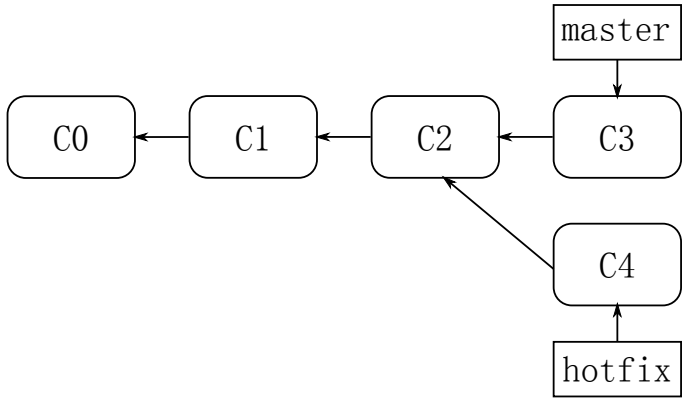
版本信息示意

- master 分支
- 使用 SHA 记录版本
- 每个版本包含全部的文件或链接
- 版本之间记录继承关系



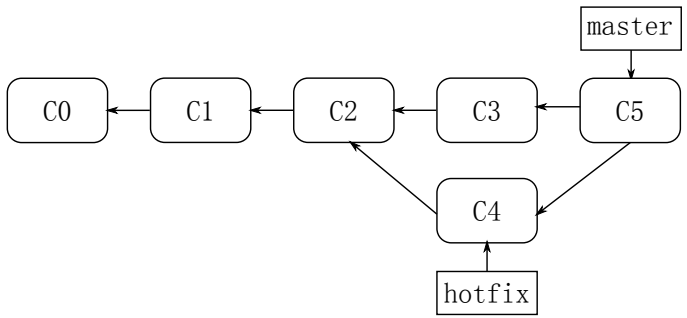
建立分支

- 建立新的分支
- HEAD 表示当前的分支



分支融合

- 在分支上更新代码
- 在 master 分支融合
- 可以删除原分支



如果发生冲突

- 修改冲突的文件，重新提交

```
<<<<<<< HEAD: hello.c
   printf( “Hello world!” ); // Print a greatin
=======
    // Print hello world
    printf( “Hello world!” );
>>>>>>> iss53: hello.c
```

向社区提交补丁

- 生成补丁文件
  git format-patch -M origin/master
- 发送补丁文件到邮件列表
  git send-email *.patch