|  |
| --- |
| Elvonzion.WX |
| Tools |
|  |
|  |
| **elvonzion@gmail.com** |
| **2012/8/12** |

|  |
| --- |
|  |

**目 录**

[1 Design 1](#_Toc391559557)

[1.1 UML 1](#_Toc391559558)

[1.2 Pattern 2](#_Toc391559559)

[1.2.1 分工相似性 2](#_Toc391559560)

[2 Algorithm 3](#_Toc391559561)

[2.1 线性表结构 3](#_Toc391559562)

[2.1.1 快速排序 3](#_Toc391559563)

[2.2 遍历方式 4](#_Toc391559564)

[2.2.1 二叉树的遍历方式 4](#_Toc391559565)

[2.2.2 树的遍历方式 4](#_Toc391559566)

[2.2.3 森林的遍历方式 4](#_Toc391559567)

[2.3 递归算法 6](#_Toc391559568)

[2.4 二叉排序树 7](#_Toc391559569)

[2.4.1 构造方式 7](#_Toc391559570)

[2.5 图结构 8](#_Toc391559571)

[3 General Cmd 9](#_Toc391559572)

[3.1 unix cmd 9](#_Toc391559573)

[3.1.1 用户管理 9](#_Toc391559574)

[3.1.2 文件管理 11](#_Toc391559575)

[3.1.3 文本处理 14](#_Toc391559576)

[3.1.4 磁盘管理 15](#_Toc391559577)

[3.1.5 文件比较 15](#_Toc391559578)

[3.1.6 IO命令 17](#_Toc391559579)

[3.1.7 安装包管理 19](#_Toc391559580)

[3.1.8 sed 21](#_Toc391559581)

[3.1.9 xargs 23](#_Toc391559582)

[3.1.10 find 24](#_Toc391559583)

[3.1.11 grep 25](#_Toc391559584)

[3.1.12 awk 25](#_Toc391559585)

[3.1.13 ps 26](#_Toc391559586)

[3.1.14 top 27](#_Toc391559587)

[3.1.15 系统状态 30](#_Toc391559588)

[3.1.16 SELinux 31](#_Toc391559589)

[3.1.17 数学计算 32](#_Toc391559590)

[3.1.18 网络管理 34](#_Toc391559591)

[3.1.19 wget 34](#_Toc391559592)

[3.1.20 smba服务器 35](#_Toc391559593)

[3.1.21 ssh配置 35](#_Toc391559594)

[3.1.22 grub配置 36](#_Toc391559595)

[3.1.23 启动方式配置 37](#_Toc391559596)

[3.1.24 修改root密码 37](#_Toc391559597)

[3.2 coding 38](#_Toc391559598)

[3.2.1 ctag 38](#_Toc391559599)

[3.2.2 cscope 38](#_Toc391559600)

[3.2.3 taglist 38](#_Toc391559601)

[3.2.4 SouceInsight 38](#_Toc391559602)

[3.2.5 eclipse 38](#_Toc391559603)

[3.2.6 代码调用关系 38](#_Toc391559604)

[3.2.7 在线编辑 38](#_Toc391559605)

[3.2.8 Doxgen 38](#_Toc391559606)

[3.3 VI 40](#_Toc391559607)

[3.3.1 查看VI信息 40](#_Toc391559608)

[3.3.2 跳转和移动 40](#_Toc391559609)

[3.3.3 查找 41](#_Toc391559610)

[3.3.4 替换 41](#_Toc391559611)

[3.3.5 字符匹配模式 41](#_Toc391559612)

[3.3.6 排版效果 42](#_Toc391559613)

[3.3.7 显示效果 43](#_Toc391559614)

[3.3.8 选取 44](#_Toc391559615)

[3.3.9 复制粘贴 44](#_Toc391559616)

[3.4 cmd 46](#_Toc391559617)

[3.4.1 命令调用 46](#_Toc391559618)

[3.5 vim 47](#_Toc391559619)

[3.5.1 命令定义 47](#_Toc391559620)

[3.5.2 变量定义 47](#_Toc391559621)

[3.5.3 HTML转换 47](#_Toc391559622)

[3.5.4 脚本教程 48](#_Toc391559623)

[3.6 Hacking 49](#_Toc391559624)

[3.6.1 Rootkit 49](#_Toc391559625)

[4 Programing 51](#_Toc391559626)

[4.1 GCC 51](#_Toc391559627)

[4.1.1 符号命名规则 51](#_Toc391559628)

[4.1.2 GCC内置函数 51](#_Toc391559629)

[4.1.3 GAS伪指令 52](#_Toc391559630)

[4.1.4 符号属性 54](#_Toc391559631)

[4.1.5 汇编语法风格 57](#_Toc391559632)

[4.1.6 内嵌汇编 58](#_Toc391559633)

[4.1.7 不透明类型 61](#_Toc391559634)

[4.1.8 编译选项 61](#_Toc391559635)

[4.3 Makefile 63](#_Toc391559636)

[4.3.1 Make参数 63](#_Toc391559637)

[4.3.2 变量语句 64](#_Toc391559638)

[4.3.3 函数语句 65](#_Toc391559639)

[4.3.4 Include与sinclude 66](#_Toc391559640)

[4.3.5 目标定义 66](#_Toc391559641)

[4.3.6 特殊目标 67](#_Toc391559642)

[4.3.7 隐含规则 67](#_Toc391559643)

[4.3.8 文本替换 68](#_Toc391559644)

[4.3.9 文本查找 68](#_Toc391559645)

[4.3.10 文本过滤 69](#_Toc391559646)

[4.3.11 运行控制 69](#_Toc391559647)

[4.3.12 文件名操作 70](#_Toc391559648)

[4.3.13 变量处理 70](#_Toc391559649)

[4.4 CPP 72](#_Toc391559650)

[4.4.1 运算符优先级 72](#_Toc391559651)

[4.4.2 数据对齐 73](#_Toc391559652)

[4.5 verilog 75](#_Toc391559653)

[4.5.1 设计流程 75](#_Toc391559654)

[4.5.2 分支逻辑 75](#_Toc391559655)

[4.6 Shell 76](#_Toc391559656)

[4.6.1 函数定义 76](#_Toc391559657)

[4.6.2 函数参数 76](#_Toc391559658)

[4.6.3 运行环境 77](#_Toc391559659)

[4.6.4 命令控制 77](#_Toc391559660)

[4.6.5 变量定义 77](#_Toc391559661)

[4.6.6 变量引用 79](#_Toc391559662)

[4.6.7 数组变量 80](#_Toc391559663)

[4.6.8 字符串变量 80](#_Toc391559664)

[4.6.9 循环语句 81](#_Toc391559665)

[4.6.10 条件语句 82](#_Toc391559666)

[4.6.11 分支语句 84](#_Toc391559667)

[4.6.12 内置函数 85](#_Toc391559668)

[4.7 Python 86](#_Toc391559669)

[4.7.1 super限制 86](#_Toc391559670)

[4.7.2 全局变量 86](#_Toc391559671)

[4.9 TCL 87](#_Toc391559672)

[4.9.1 特点 87](#_Toc391559673)

[5 Debug 88](#_Toc391559674)

[5.1 ICE 88](#_Toc391559675)

[5.1.1 OpenOCD 88](#_Toc391559676)

[5.2 memory check 97](#_Toc391559677)

[5.2.1 valgrind 97](#_Toc391559678)

[5.3 gdb 100](#_Toc391559679)

[5.3.1 常用命令 100](#_Toc391559680)

[5.4 Terminal 102](#_Toc391559681)

[5.4.1 终端服务器 102](#_Toc391559682)

[5.4.2 SecureCRT 102](#_Toc391559683)

[6 Office 104](#_Toc391559684)

[6.1 PSPad 104](#_Toc391559685)

[6.1.1 hex显示格式 104](#_Toc391559686)

[6.2 Word 105](#_Toc391559687)

[6.2.1 大小写转换 105](#_Toc391559688)

[6.2.2 自动编辑 105](#_Toc391559689)

[6.2.3 域显示 105](#_Toc391559690)

[6.2.4 目录页码编号 105](#_Toc391559691)

[6.2.5 目录项页码格式 105](#_Toc391559692)

[6.2.6 目录项小圆点格式 106](#_Toc391559693)

[6.2.7 文档比较 106](#_Toc391559694)

[6.2.8 图片合成 106](#_Toc391559695)

[6.2.9 粘贴代码 106](#_Toc391559696)

[6.2.10 快捷键 106](#_Toc391559697)

[6.3 Excel 108](#_Toc391559698)

[6.3.1 Filter 108](#_Toc391559699)

[6.3.2 输入运算符 108](#_Toc391559700)

[6.3.3 百分比 108](#_Toc391559701)

[6.3.4 文本转换 108](#_Toc391559702)

[6.3.5 排序 108](#_Toc391559703)

[7 Manage 109](#_Toc391559704)

[7.1 maints 109](#_Toc391559705)

[7.1.1 操作步骤 109](#_Toc391559706)

[7.1.2 状态更新 110](#_Toc391559707)

[7.1.3 状态转换 110](#_Toc391559708)

[7.2 PDM 112](#_Toc391559709)

[7.2.1 状态转换 112](#_Toc391559710)

[7.3 CQ 114](#_Toc391559711)

[7.4 ERP 114](#_Toc391559712)

[7.4.1 过程资产积累 114](#_Toc391559713)

[7.5 SVN 115](#_Toc391559714)

[7.5.1 常见错误 115](#_Toc391559715)

[7.6 Git 116](#_Toc391559716)

[7.6.1 创建本地仓库 116](#_Toc391559717)

[7.6.2 添加和删除 116](#_Toc391559718)

[7.6.3 提取和提交 116](#_Toc391559719)

[7.6.4 常用信息查看 116](#_Toc391559720)

[7.6.5 配置比较工具 116](#_Toc391559721)

[7.6.6 网络代理配置 117](#_Toc391559722)

[7.6.7 patch功能 117](#_Toc391559723)

[7.7 English 118](#_Toc391559724)

[7.7.1 字符读法 118](#_Toc391559725)

[7.7.2 计时 121](#_Toc391559726)

[7.7.3 计数 121](#_Toc391559727)

# Design

## UML

## Pattern

### 分工相似性

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 关注领域 | software | hardware | IC | PM | Life |
| 逻辑 | logic code | board scheme | VHDL | 预算 | 立志 |
| 物理 | bsp | singal characteristic | physic |  |  |
| 集成 | obj link | pcb layout | ic layout | 沟通 |  |
| 运行 | runtime |  | timing |  |  |

# Algorithm

## 线性表结构

### 快速排序

|  |  |
| --- | --- |
| void **QKSort**(RTYPE data[], int low, int high ) | 全部交换 |
| if(**low < high**) {  pos = **QKPass**(data, pos, high); | 一次交换 |
| **QKSort**(data,low,pos-1);  **QKSort**(data,pos+1,high); } | 递归交换 低位置区域 递归交换 高位置区域 |
| int **QKPass**(RTYPE data[], int left, int right ) | 一次交换 |
| low = left; high = right; |  |
| ref = data[low]; while(**low < high**) { | 基准使用最低位置 |
| while(**low < high&&**data[high].key **>=** ref.key)  high--; | 向高的位置找小的 (不大于基准) |
| if(**low < high**) {  data[low] = data[high];  low++;  } | 将高位置的小数移动到 低位置 |
| while(**low < high** **&&**data[low].key **<** ref.key)  low++ | 向低的位置找大的 (不小于基准) |
| if(**low < high**) {  data[high] = data[low];  high--;  } | 将低位置的大数移动到 高位置 |
| } data[low] = ref; | 不大不小的基准数移动到 不高不低的分割点 |
| return low; |  |

## 遍历方式

“遍历”的含义是对结构中的每个数据元素都访问一次且仅仅访问一次。因此进行遍历应该确定一条搜索路径，使得结构中的每个数据元素都出现在这条搜索路径上，才能确保每个数据元素都被访问到。

|  |  |  |
| --- | --- | --- |
| **二叉树** | **树** | **森林** |
|  |  |  |
| 先序**ABCDEFGHK** | 先根**ABEHIJCDFGK** | 先序**BEHIJRCPADFGK** |
| 中序**BDCAEHGKF** | 后根**HIJEBCFKGDA** | 中序**HIJEBPCRFKGDA** |
| 后序**DCBHKGFEA** |  | |

### 二叉树的遍历方式

1. **先序**

**根结点🡪先序**遍历左子树**🡪先序**遍历右子树。

1. **中序**

**中序**遍历左子树**🡪根结点🡪中序**遍历右子树。

1. **后序**

**后序**遍历右子树**🡪后序**遍历右子树**🡪根结点**。

### 树的遍历方式

1. **先根**

**根结点🡪先根**遍历根的各棵子树(从左到右)。

1. **后根**

**后根**遍历根的各棵子树(从左到右)**🡪根结点**。

### 森林的遍历方式

1. **先序**

**第一棵树的根结点🡪先序**遍历第一棵树中的森林**🡪先序**遍历剩余的树

1. **中序**

**中序**遍历第一棵树中的森林**🡪第一棵树的根结点🡪中序**遍历剩余的树

## 递归算法

有限扩展的分支结构必然存在叶子结点和分支结点，递归算法的主要原则就是，将遍历分支结构的逻辑采用递归流程实现，将叶子结点采用不可递归流程实现。

在递归算法的接口设计中，首先要找出这种分支结构的逻辑并将其作为算法主接口，然后将叶子结点逻辑作为不可递归逻辑并作为结束一次递归的条件。

## 二叉排序树

### 构造方式

## 图结构

# General Cmd

## unix cmd

### 用户管理

* 权限配置

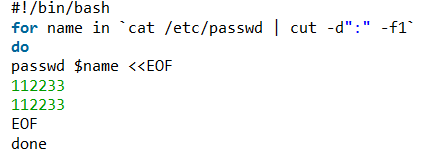
1. 配置文件

|  |  |  |
| --- | --- | --- |
| 用户类型 | 配置文件 | 密码文件 |
| 用户（user） | /etc/passwd | /etc/shadow |
| 用户组（group） | /etc/group | etc/gshadow |

1. 查看用户

awk -F':' '{print $1}' /etc/passwd

1. 密码重置



1. 增加ssh/root/sudo权限.

$ sudo adduser username ssh

$ sudo adduser username root

$ sudo adduser username sudo

1. 增加sudoer

键入visudo即可进入sudo配置，或修改 /etc/sudoers

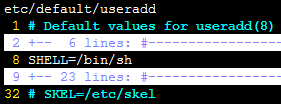
* 用户管理工具

|  |  |
| --- | --- |
| 命令 | 功能 |
| useradd | 添加用户，与adduser相同 |
| passwd | 为用户设置密码 |
| usermod | 修改登录名、用户的家目录等等 |
| pwcov | 同步用户从/etc/passwd 到/etc/shadow |
| pwck | 校验/etc/passwd 和/etc/shadow 文件内容是否合法或完整 |
| pwunconv | 从/etc/shadow和 /etc/passwd 创建/etc/passwd ，然后删除 /etc/shadow |
| finger | 查看用户信息 |
| id | 查看用户的UID、GID及所归属的用户组 |
| chfn | 更改用户信息 |
| su | 用户切换 |
| sudo | 使用另一个用户来执行命令 |
| groupadd | 添加用户组 |
| groupdel | 删除用户组 |
| groupmod | 修改用户组信息 |
| groups | 显示用户所属的用户组 |
| grpck | 校验/etc/group和/etc/gshadow文件内容是否合法或完整 |
| grpconv | 从/etc/group和/etc/gshadow内容来同步或创建/etc/gshadow ，如果/etc/gshadow 不存在则创建 |
| grpunconv | 从etc/group 和/etc/gshadow同步或创建/etc/group ，然后删除gshadow |

* 添加用户规则

1. 增加用户通用配置

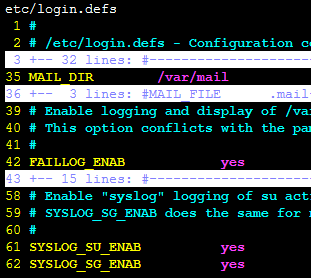
通过useradd 命令添加用户时将使用/etc/default/useradd中的规则对用户进行配置。



其中etc/skel 目录下的文件由系统自动复制到新添加用户（user）的家目录下；但如果通过修改 /etc/passwd 来添加用户时，则需要自己创建用户的家目录，然后把/etc/skel 下的文件复制到用户的家目录下，然后要用chown 来改变新用户家目录的属主；

1. 用户登录配置

/etc/login.defs 文件是当创建用户时的一些规划，比如创建用户时，是否需要家目录，UID和GID的范围；用户的期限等等，这个文件是可以通过root定义的；



### 文件管理

* 权限更改

|  |  |  |
| --- | --- | --- |
| 修改类型 | 命令 | 功能 |
| 增加 | chmod o+x <文件名> | 对other增加可执行权限 |
| 去掉 | chmod g-w <文件名> | 对group取消可写权限 |
| 多属性 | chmod ug+rwX <文件名> | 对user，group增加rwX权限 |
| 更换owner | chown <文件名> | 更换文件的owner |

A combination of the letters ugoa controls which users' access to the file will be changed: the user who owns it (u), other users in the file's group (g),other users not in the file's group (o), or all users (a). If none of these are given, the effect is as if a were given, but bits that are set in the umask are not affected.

The operator + causes the selected file mode bits to be added to the existing file mode bits of each file; - causes them to be removed; and = causes them to be added and causes unmentioned bits to be removed except that a directory's unmentioned set user and group ID bits are not affected.

The letters rwxXst select file mode bits for the affected users: read (r), write (w), execute (or search for directories) (x), execute/search only if the file is a directory or already has execute permission for some user (X), set user or group ID on execution (s), restricted deletion flag or sticky bit (t).

Instead of one or more of these letters, you can specify exactly one of the letters ugo: the permissions granted to the user who owns the file (u), the per‐ missions granted to other users who are members of the file's group (g), and the permissions granted to users that are in neither of the two preceding cate‐ gories (o).

A numeric mode is from one to four octal digits (0-7), derived by adding up the bits with values 4, 2, and 1. Omitted digits are assumed to be leading. The first digit selects the set user ID (4) and set group ID (2) and restricted deletion or sticky (1) attributes. The second digit selects permis‐sions for the user who owns the file: read (4), write (2), and execute (1); the third selects permissions for other users in the file's group, with the same values; and the fourth for other users not in the file's group, with the same values.

* 文件拷贝

1. dd <选项>

if：用于读的输入文件

of：用于写的输出文件

count：个数

bs：块大小

比如：dd if=/dev/sda of=/dev/sdb 将sda内容拷贝到sdb.

1. cpio

利用cpio 可将文件或目录从文件库获取出来或将散列文件拷贝到文件库。一般利用输入/输出重定向或管道的原理，达到真正复制的功能。

cpio 共有三种基本模式，-o即copy-out 模式，将一组文件copy到一个文件库，-i 即copy-in 模式，读取文件库，并将其展开在当前目录。-p 能从某个目录读取所有文件（包括子目录到另一个目录），且不以archive(归档)的方式存放。

cpio 的指令格式：

cpio –i[bcdmrtuv] [patterns]

cpio –o [abcv]

cpio –p [adlmuv][directory]

* 查看文件列表

ls –l <路径>

显示结果格式如下：

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 字段 | attr | | | | dirs**/**links | owner info | | size | create time | | | name |
| T | u | g | o | user | group | month | day | time |
| 示例 | -rw-r--r-- | | | | 2 | root | root | 1584 | 11月 | 28 | 18:18 | file0 |
| lrw-r--r-- | | | | 1 | 0 | 0 | 11 | 2月 | 12 | 15:09 | link0-->file0 |
| drwx----- | | | | 8 | root | root | 4096 | 12月 | 24 | 15:17 | dir0 |
|  | -rwsr-sr-x | | | | 1 | root | root | 9208 | 12月 | 11 | 15:05 | file1 |

各字段意义：

|  |  |  |
| --- | --- | --- |
| 字段 | | 含义 |
| attr | T | 文件类型 |
| u/g/o | user/group/other，权限位适用的用户 |
| r/w/x/s | Read/Write/eXecute/Sticky，对应用户的权限。  Sticky表示当文件执行时，把该文件的UID 或 GID 赋给执行进程的UID 或 GID，且只能由其owner更改/删除。 |
| owner info | | 当owner已在系统删除或者使用-n选项，则owne info中会显示其该owner对应的id号，而不是名称 |
| size | | 指该文件的内容占用物理空间大小，目录可以理解为一个描述子目录名称的文件 |
| name | | dirs./links表示该文件的硬链接符号个数或者子目录个数 |

* 编码转换

1. dos下的文本转换为unix格式

dos2unix <文件路径>

* 设备挂载 mount

命令格式： mount [-t <vfstype>] [-o options] <device> <mount\_point>

1. 指定文件系统类型- t <vfstype>

|  |  |
| --- | --- |
| vfstype | 类型 |
| iso9660 | 光盘或光盘镜像 |
| msdos | DOS fat16 |
| vfat | Windows 9x fat32 |
| ntfs | Windows NT ntfs |
| smbfs | Mount Windows网络共享 |
| nfs | UNIX(LINUX)网络共享 |

1. 描述设备的挂接方式-o <options> 主要用来，多个选项用“,”分隔。

|  |  |
| --- | --- |
| options | 功能 |
| loop | 将一个文件作为硬盘分区挂接上系统，比如挂载文件系统镜像 |
| ro | 采用只读方式挂接设备 |
| rw | 采用读写方式挂接设备 |
| iocharset | 指定访问文件系统所用字符集 |
| remount | 重新加载挂载点，比如remount, rw表示重新以rw方式挂载 |

* 查看信息

|  |  |
| --- | --- |
| 信息类型 | 命令 |
| 文件信息 | file <文件> |
| 目录结构信息 | tree <路径> -aps |
| 占用空间信息 | du <路径> -sh |
| 挂载盘信息 | df -Th |
| 命令路径 | whereis <命令名称> |

* 文件打包

1. 解包

|  |  |
| --- | --- |
| 解压命令 | 解压文件 |
| tar xvf | xxx.tar |
| tar xvjf | xxx.tar.bz或者xxx.tar.bz2 |
| tar xvzf | xxx.tar.gz 或 .tgz |
| gunzip | xxx. gz |
| tar xvJf | xxx.tar.xz (tar 版本 >= 1.22) |
| xz -d | xxx.tar.xz |

1. 打包

tar cvf xxx.tar 要打包的文件(夹)

1. 查看包文件列表

tar -tvf xxx.tar

* 文件管理示例

1. 删除某文件之外的所有文件

ls | grep -v 不删除的文件名 | xargs rm

1. 按文件名查找

find “查找路径” -name “文件名匹配模式”

1. 按内容查找

grep -r “文件包含内容的匹配模式” “查找路径”

### 文本处理

* 文本输出

|  |  |  |
| --- | --- | --- |
| 显示格式 | | 命令 |
| 正行序输出 | | cat |
| 逆行序输出 | | tac |
| 行截取 | 前行 | head –n<number> |
| 前行之后 | tail –n<+number> |
| 末行 | tail –n<number> |
| 末行之前 | head –n-<number> |
| 两行之间 | sed -n'<line1>,<line2>p ' |
| 列截取 | | cut -d' ' -f<number> |
| awk -F' ' '{print $<number>}' |

* 显示控制

1. 下翻页more
2. 上/下翻页less

* 行排序

Usage: sort [OPTION]... [FILE]...

or: sort [OPTION]... --files0-from=F

Write sorted concatenation of all FILE(s) to standard output.

Mandatory arguments to long options are mandatory for short options too.

Ordering options:

-r, --reverse reverse the result of comparisons

-k, --key=POS1[,POS2] start a key at POS1 (origin 1), end it at POS2

(default end of line). See POS syntax below

### 磁盘管理

|  |  |
| --- | --- |
| 命令名称 | 功能 |
| fdisk | 磁盘分区 |
| blkid | UUID获取 |
| df | 分区状态 |

### 文件比较

* 文件比较命令

diff　[选项]　<源文件> <目标文件>

|  |  |  |
| --- | --- | --- |
| 选项 | 功能 | 含义 |
| r | 递归 | 将双方目录以及子目录中的所有文件进行比较 |
| N | 对新文件加入比较 | 如果一方存在另一方缺失的文件，则将该存的文件作为新文件 |
| u | unified的行数 | 在输出两个文件的内容差异时，同时会在差异描述区域前后增加一段该区域附近无差异的内容(unified)，u选项即用于控制该无差异内容的行数。比如：-u1表示前后各增加1行附近的无差异内容。  如果没有使用-u选项，则默认使用3行。 |
| E | 忽略空白字符 | -E, -b, -w, -B, --strip-trailing-cr忽略各种空白 |

* 补丁命令

patch [选项]　<补丁文件>

|  |  |  |
| --- | --- | --- |
| 选项 | 功能 | 含义 |
| p | 路径深度 | 当前目录向下多少层为<补丁文件>中所描述的文件路径参考目录，比如p0表示patch中文件参考目录即当前目录，p2表示当前目录向下2个级别是参考目录。 |
| E | 删除空文件 | 同步后如果整个文件的内容被全删除则将其文件也删除。 |
| R | 取消已存在的修改 | 如果发现patch中描述的修改在新文件已经存在，则恢复(revert)该对应的修改。 |

* 补丁操作

常用的补丁操作：

|  |  |  |
| --- | --- | --- |
| 操作类型 | 命令 | 功能 |
| 创建 | diff -uNr src dst > xxx.patch | 比较src和dst不同，生成补丁xxx.patch |
| 升级 | patch -p0 < xxx.patch | 将补丁同步到当前目录的源码中。应文件  如果应用补丁时有冲突则会在冲突文件对应的目录生成后缀为rej的同名文件。 |
| 回退 | patch –R -p0 < xxx.patch | 将补丁从当前目录的源码中移除 |

* 补丁格式说明

1. 补丁头

用于描述修改点用于哪个文件，“---”后的路径表示diff命令中的<源文件>，“+++”后的路径表示<目标文件>，一个补丁文件中可能包含多个补丁头，即可能包含多个文件的修改。

1. 修改点位置描述

补丁头紧靠的区域，且以@@开始的行。用于描述修改点在文件中的位置，并以@@结束。每两个位置描述之间的内容即一个修改点描述。

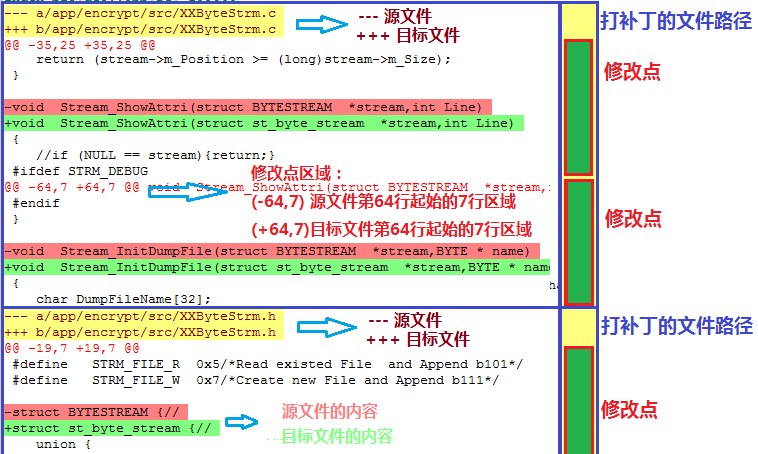
1. 修改点描述

描述差异内容的对比结果，对比结果位于unified内容之间的区域。以‘-’号开始的行表示<源文件>的内容，以‘+’号开始的行表示<目标文件>的内容。

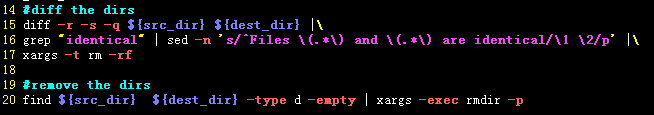
如果修改点描述中没有同时出现‘+’和‘-’的区域，则表示一方对应区域是不存在该内容，即修改点为增加或删除的内容。

* 补丁文件示例

如下是一个补丁文件的部分内容，蓝色方框内的区域是一个文件的所有修改点，绿色方块对应的区域为一个具体的修改点描述。



* 删除相同文件



### IO命令

* 管道命令

command\_a | command\_b：将命令的标准输出作为下个命令的标准输入

* 文件编号描述符

文件引用时可使用其对应的编号，该编号称为文件描述符，系统默认的描述符为0,1,2，其他的描述符可自由分配，默认的描述符如下：

0: 标准输入stdin;

1: 标准输出stdout;

2: 标准错误输出stderr;

* 文件引用方式

当作为输出(目标)文件时，引用格式为 &n，或者直接使用文件名；

当作为输入(源)文件时，引用格式为n。

其中n表示文件编号。

* 重定向

功能：

每个进程在调用系统IO接口(即输出内容或者获取输入内容)时，其默认读取stdin(编号为0) 文件所为系统输入；默认的打印存储到stdout(编号为1)或者stderr(编号为2) 文件。重定向用于修改默认的系统输入输出文件。

格式：

输出重定向格式： **输出端**（命令输出或文件） > 接收端(文件)

输入重定向格式： 接收端(命令) < **输入端**（文件）

|  |  |
| --- | --- |
| 文件重定向 | 功能 |
| output > recv | 将output覆盖到接收端 |
| output >> recv | 将output追加到接收端 |
| recv < input | 将input全部传递给接收端 |
| recv << delitmiter | 将stdin传递给接收端，遇到delimiter符号则停止传递 |
| recv\_a < input > recv\_b | 将input传给recv\_a，再将recv\_a的输出覆盖到recv\_b |

格式说明：

|  |  |
| --- | --- |
| 命令IO重定向 | 功能 |
| cmd 2>&1 | 将命令输出到stderr的内容输出到stdout |
| cmd **>/>>** &n | 把标准输出**覆盖/追加**写入文件编号为n的文件 |
| cmd **>/>>** file | 把标准输出**覆盖/追加**写入文件名为file的文件 |
| cmd **>/>>** file 2>&1 | 把stdout和stdin一起**覆盖/追加**写入文件file中 |
| cmd 2**>/>>** file | 将输出到stderr/stdout的内容**覆盖/追加**写入file文件 |
| cmd < file | 把file内容作为命令的标准输入 |
| cmd << delimiter | 命令从标准输入读数据时当遇到delimiter符则结束 |
| cmd < &m | 文件描述符m作为标准输入 |
| cmd < &- | 关闭标准输入 |
| cmd < file1 > file2 | file1 作为命令标准输入，并将标准输出定向到file2 |

示例：

grep search-word < testfile #搜索testfile中的search-word。

for line in $(<b); do cat a | grep $line; done #搜索a文件中包含b内容的行。

* 分配文件编号 [file\_no] **<>**filename
* 释放文件编号 file\_no >&-

将文件编号file\_no分配给filename。如果没有file\_no，则文件编号为0（即默认重定向到stdin）。

示例：

exec 3<> testfilename #将文件编号 3分配给testfilename .

echo “testdata” >&3 # 写数据到文件编号.3，即写入testfilename。

exec 3>&- # 解除文件编号号3绑定的文件名。

* 双向重定向

tee [-a] file 将标准输入复制到指定文件，并显示到标准输出，即重定向数据到目标文件的同时，还要保证数据能够正常处理。-a 往文件尾添加内容。

示例：

make 2>&1 | tee build.log #将make的输出重定向，将其作为命令tee的标准输入并保存到 build.log文件。

last | tee last\_backup | cut -d " " -f 1 #tee相当于对last的结果备份了一次。

### 安装包管理

* dpkg

功能：管理deb类型本地安装包，如:ubuntun,debian平台的安装包。

语法：dpkg [-选项] <文件名.deb>

注意：dpkg命令无法自动解决依赖关系。如果安装的deb包存在依赖包，则应避免使用此命令，或者按照依赖关系顺序安装依赖包。

|  |  |  |
| --- | --- | --- |
| 选项 | | 功能 |
| –I | --info | 查看软件包的详细信息，包括软件名称、版本以及大小等 |
| –c | --contents | 查看软件包中的文件结构 |
| –i | --install | 安装软件包 |
| –l | --list | 查看软件包的信息 |
| –L | --listfiles | 查看软件包安装的所有文件 |
| –s | --status | 查看状态 |
| –r | --remove | 移除式卸载,删掉数据和可执行文件 |
| -P | --purge | 清除式卸载, 在-r的功能上，另外删除所有的配制文件 |
| -S |  | 查询系统中某个文件属于哪个软件包 |

* apt-get

高级包装工具（英语：Advanced Packaging Tools,简称：APT）是Debian及其衍生发行版（如：ubuntu）的软件包管理器。APT可以自动下载，配置，安装二进制或者源代码格式的软 件包，因此简化了 Unix系统上管理软件的过程,apt-get命令一般需要root权限执行，所以一般跟着sudo命令。

功能： 通过“安装源”管理安装包，可以使网络的，也可以是本地的。

语法： apt-get [选项] 命令

apt-get [选项] install|remove| source pkg1 [pkg2 ...]

|  |  |  |
| --- | --- | --- |
| 选项 | 功能 | |
| -y | 假定对所有的询问选是，不提示 | |
| -d | 仅下载 - 不安装或解压归档文件 | |
| -s | 模拟执行命令，不实际安装。 | |
| -f | 尝试修正系统依赖损坏处 | |
| -m | 如果归档无法定位，尝试继续 | |
| -u | 同时显示更新软件包的列表 | |
| -b | 获取源码包后编译 | |
| -c=? | 阅读此配置文件 | |
| -o=? | 设置自定的配置选项，如 -o dir::cache=/tmp | |
| -q | 输出到日志 - 无进展指示 | |
| -qq | 不输出信息，错误除外 | |
| -V | 显示详细的版本号 | |
| -h | 帮助。 | |
| 命令 | 功能 | |
| install | 默认 | 安装软件包 |
| -f | 修复安装, --fix-missing，只安装缺少的包 |
| --reinstall | 重新安装 |
| remove | 默认 | 移除式卸载，删除包 |
| --purge | 清除式卸载，删除包，包括删除配置文件等 |
| clean | 清理无用的包 | |
| autoclean | 清理无用的包 | |
| build-dep | 安装相关的编译依赖 | |
| clean | 清理无用的包 | |
| source | 下载该包的源代码 | |
| update | 更新源 | |
| upgrade | 更新已安装的包 | |
| dist-upgrade | 更新本地所有安装包 | |
| dselect-upgrade | 使用 dselect 升级 | |
| check | 检查是否有损坏的依赖 | |

* apt-cache

功能：安装包缓存查看。

语法：apt-cache[选项] 命令pkg1 [pkg2 ...]

|  |  |
| --- | --- |
| 选项 | 功能 |
| depends | 查看该包需要依赖哪些包 |
| rdepends | 查看该包被哪些包依赖 |
| search | 搜索包 |
| show | 获取包的相关信息，如说明、大小、版本等 |

* apt-key

sudo apt-key adv –keyserver <serverURL> –recv-keys <keyvalue>

* 源码安装

1. 执行./configure
2. make
3. sudo make install

* 安装源搭建

1. 获取安装包(\*.deb)并统一放到指定文件夹下(比如:/my\_software)；
2. 创建软件安装源信息(需要安装dpkg-dev)

sudo dpkg-scanpackages /my\_software /dev/null | gzip>/software/Packages.gz

Packages.gz存有/my\_software文件下的软件包信息及其依赖关系。

1. 添加下载源列表

在/etc/apt/sources.list中添加，或者在sources.list.d下创建一个xxx.list文件。

deb file:/// my\_software/

1. 更新源信息

通过读取/etc/apt/目录下的source.list文件以及子目录sources.list.d下所有.list文件实现对下载源的同步。

sudo apt-get update

### sed

sed (Stream Editor)是一种在线编辑器，它一次处理一行内容。处理时，把当前处理的行存储在临时缓冲区中，称为“模式空间”（pattern space），接着用sed命令处理缓冲区中的内容，处理完成后，把缓冲区的内容送往标准输出。接着处理下一行，这样不断重复，直到文件末尾。

* 使用方式

1. 命令sed [options] 'command' <待处理文件>
2. 脚本sed [options] -f scriptfile <待处理文件>

* 选项

|  |  |
| --- | --- |
| 选项符号 | 含义 |
| -e command | --expression=command，多个命令依次执行，前面的处理结果为后面的输入 |
| -f script-file | --filer=script-file |
| -n | --quiet, --silent ,一般都需要该选项，否则会打印出待处理文件内容。 |
| -h | --help |

* 命令的格式

[定位] + 处理

* 命令的定位单元

1. 用逗号分隔的两个行数表示以这两行为起止的行的范围（包括行数表示的那两行）。如1，3表示1，2，3行， $表示最后一行。
2. 通过数据，正则表达式的方式确定 。

* 命令的处理单元

|  |  |
| --- | --- |
| 命令 | 含义 |
| a\ | 在当前行后面加入文本，sed '/test/a\\ text' file 🡪在 file含test的行后面插入text |
| i\ | 在当前行上面插入文本。 |
| c\ | 用新的文本改变本行的文本，sed '/test/c\\ text' file 🡪将 file含test的文本替换为text |
| y | 表示把一个字符映射为另外的字符（不适于正则表达式）sed '1,20y/abc/789/' file🡪将file中1`20行中abc分别替换为789，如tr命令类似。 |
| b lable | 跳到脚本<lable>的地方，如果lable不存在则跳到脚本末尾 |
| d | 从模板块（Pattern space）位置删除行 |
| D(大写) | 删除模板块的第一行 |
| h | 拷贝模式空间的内容到缓冲区，即只保留最后一次匹配的模式空间内容。 |
| H(大写) | 追加模板块的内容到内存中的缓冲区，保留所有匹配的模式空间内容。 |
| g | 获得内存缓冲区的内容，并替代当前模板块中的文本 |
| G(大写) | 获得内存缓冲区的内容，并追加到当前模板块文本的后面 |
| l | 列出不能打印字符的清单 |
| n | 读取下一个输入行，sed '/test/{ n; d; }' file🡪对file含test的行的下一行删除处理 |
| N(大写) | 追加下一个输入行到模板块后面并在二者间嵌入一个新行，改变当前行号码 |
| P(大写) | 打印模板块的第一行 |
| q | 退出Sed，sed '10q' file 🡪 打印完file前10行后，退出sed |
| r file | 从file中读行，sed '/test/r input' file🡪file中含test的行后插入input文件的内容 |
| ! | 表示后面的命令对所有没有被选定的行发生作用 |
| s | s/a/b用b替换正则表达式a(s后的第一个字符为分割符，默认为/,中间使用要转义) |
| = | 打印当前行号码 |
| g(替换) | 表示行内全局global替换 |
| p | 表示将结果print到标准输出，sed '/my/p' file🡪输出file中含my的行 |
| x | 表示互换模板块中的文本和缓冲区中的文本 |
| w file | 表示把行写入文件, sed -n '/test/w output' file🡪 file含test的行后写入output |
| W file | 同w，追加模式，将行添加入文件尾部。 |

* 命令中正则表达式的元字符

|  |  |
| --- | --- |
| 元字符 | 功能 |
| ^ | 行首定位符, /^my/ 🡪匹配所有以my开头的行 |
| $ | 行尾定位符, /my$/ 🡪匹配所有以my结尾的行 |
| . | 匹配除换行符以外的单个字符, /m..y/🡪匹配包含字母m，后跟两个任意字符，再跟字母y的行 |
| \* | 匹配零个或多个前导字符, /my\*/ 🡪匹配包含字母m,后跟零个或多个y字母的行 |
| [] | 匹配指定字符组内的任一字符, /[Mm]y/🡪 匹配包含My或my的行 |
| [^] | 不在指定字符组内的任一字符, /[^Mm]y/🡪匹配包含y，但y之前的那个字符不是M或m的行 |
| \(..\) | 分组, 最多可以定义9个标签，从左边开始编号。1,20s/\(you\)self/\1r/🡪将you匹配的文本分组为1，之后可以使用\1来引用它。 |
| & | 查找串引用, s/my/\*&\*/🡪符号&代表查找串。my将被替换为\*my\* |
| \< | 词首定位符,  /\<my/🡪匹配包含以my开头的单词的行 |
| \> | 词尾定位符, /my\>/🡪匹配包含以my结尾的单词的行 |
| x\{m\} | 连续m个x,  /9\{5\}/🡪 匹配包含连续5个9的行 |
| x\{m,\} | 至少m个x, /9\{5,\}/ 🡪匹配包含至少连续5个9的行 |
| x\{m,n\} | 至少m个，但不超过n个x, /9\{5,7\}/🡪匹配包含连续5到7个9的行 |

### xargs

* 用途

1. 构造参数列表并运行命令。

将接收的参数传递给后面的command 命令执行，其原理是把标准输入的数据流分割为一些小的块，然后与一个命令字符串组合，从而构造出一个带参数的命令序列。

1. 将多行输入转换为单行 （特殊功能）

* 使用格式

xargs [-p] [-t]

[-e [EOFString]] [-E EOFString]

[-i [ReplaceString]] [-I ReplaceString]

[-l [Number]] [-L Number]

[-n Number [-x]]

[ -s Size]

[Command [Argument ...]]

注：不要在小写的标志和参数之间放置空格

* 个别选项说明

1. –i,[ReplaceString]，将标准输入的每一行字符串用ReplaceString表示，如果Command 的参数中出现ReplaceString，则扩展为其表示的字符串。ReplaceStrings 不能在超过 5 个自变量中使用，默认为{}。在每个标准输入行开始的空字符被忽略。

例如： 删除文件：ls|xargs -I rm -rf {}

文件改名：ls|xargs -I mv {} {}.old

1. -t，打开调试功能，显示每次所组的命令，在调试中非常有用。

### find

* 使用格式

find path -option [ -print] [ -exec –ok command] {} \;

1. -print 将查找到的文档输出到标准输出;

-exec command {} \; 将查到的文档执行command操作, {}表示find所找到的文件,注意{} 和 \;之间有空格;

1. -ok 和-exec相同，只但是在操作前要询用户;

* option的逻辑运算

1. 逻辑与，在命令中用“-a”。是系统缺省的选项，表示所给的条件都要满足；

例如：find –name ‘tmp’ –xtype c -user ‘inin’

用于寻找三个给定条件都满足的所有文件。

1. 逻辑或，在命令中用“-o”。表示只要所给的条件中有一个满足即可；

例如: find –name ‘tmp’ –o –name ‘mina\*’

用于查询文件名为tmp或是匹配mina\*的所有文件。

1. 逻辑非，在命令中用“!”。表示不满足所给条件。

例如：find ! –name ’tmp’

用于查询文件名不是‘tmp’的所有文件。

* 示例

1. 在查找的文件中继续查找包含有特定字符串的文件

find . -name "\*.c" -exec grep "main" {} \;

1. 查找权限为777的文档

find . -perm -777 -exec ls -l {} \;

1. 查找磁盘中大于3M的文档

find .**/** -size +3000k -exec ls -ld {} ;

1. 查找最近两天内改变过的文档

find .**/** -mtime -2

1. 将查找的文件拷到另一个地方

find \*.c -exec cp {} **/**tmp ;

1. 查找某文件其他的硬链接路径

find <搜索路径> –inum <文件节点号> **//**用ls -i <文件路径> 得到文件节点号

### grep

* PATTERN类型

1. 多表达式

egrep '(cron|syslog)'

### awk

* 使用格式

awk [options] 'script' file1,file2,...  
awk [options] 'PATTERN { action }' file1,file2,...

根据指定的分隔符将读入的文本切片，默认分隔符为空格，引用第一段$1,引用第二段$2,...引用全部字段$0。

* Options可选项

1. 输入分隔符

-F 字符 或者 FS=字符,比如: awk -F : '{print $1}' file 即指定以“：”为分隔符，分割file的文本，输出分割后的第一段。

1. 输出分隔符

OFS=字符，比如: awk -v OFS=: '{print $1,$2}' file即指定以“：”为输出分隔符，输出分割的第一段和第二段。

* PATTERN类型

1. 正则表达式: **/**regular expression**/**
2. 表达式

其值非0或为非空字符时则模式匹配;

1. 匹配范围: pat1,pat2

被pat1匹配到的行开始到被pat2匹配到的行结束;

1. 特殊模式: ' BEGIN**/**END { action }'

仅在awk命令执行前运行一次或结束前运行一次;

1. 空模式: ' Empty { action }'

匹配任意输入行。

### ps

report a snapshot of the current processes.

* 参数含义

|  |  |
| --- | --- |
| 参数 | 含义（参数前有‘-’与没有的含义不相同） |
| -e | 显示所有进程，环境变量 |
| e | 列出程序时，显示每个程序所使用的环境变量 |
| -A | 显示所有进程，与 -e 效果相同 |
| -a | 显示与终端无关的所有进程(与 ‘a’不相同) |
| a | 显示当前终端上的所有进程(包括其它用户的进程) |
| -N | 显示所有的进程，除了执行ps指令终端下的进程之外 |
| -f | 用全格式显示 |
| f | 用ASCII字符显示树状结构，表达进程间的关系 |
| -H | 显示树状结构，表示进程间的关系 |
| -h | 不显示标题 |
| x | 显示没有控制终端的进程 |
| r | 仅显示运行中的进程 |
| w | 避免详细参数被截断 |
| -u | 显示有效使用者(effective user) 相关的进程 |
| u | 按用户名和启动时间的顺序来显示 |
| j | 以工作的格式 (jobs format) 来显示 |
| -l | 长格式输出 |
| -w | 宽格式输出 |
| s | 采用signal的格式显示进程状况 |
| S | 显示的进程包括已中断的子进程信息 |
| -t <tty> | 显示指定终端上运行的进程 |
| -T | 显示线程 |

* 输出格式

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 列名 |  |  | 含义 | | |
| HEADER | ALIAS | CODE |  |  | | |
| USER | user | %U |  | 进程所有者 | | |
| PID | pid | %p |  | 进程ID | | |
| %CPU | pcpu | %C |  | 占用的 CPU 使用率 | | |
| %MEM |  | %mem |  | 占用的内存使用率 | | |
| VSZ | vsz | %z |  | 占用的虚拟内存大小 | | |
| RSS |  |  |  | 占用的内存大小 | | |
| TTY | tty | %y |  | 终端次设备号 (minor device number of tty) | | |
| STAT |  |  |  | 进程状态 | | |
|  |  |  | D | Uninterruptible sleep (usually IO) | |
|  |  |  | R | Running or runnable (on run queue) | |
|  |  |  | S | Interruptible sleep (waiting for an event to complete) | |
|  |  |  | T | Stopped, either by a job control signal or because it is being traced. | |
|  |  |  | W | paging (not valid since the 2.6.xx kernel) | |
|  |  |  | X | dead (should never be seen) | |
|  |  |  | Z | Defunct ("zombie") process, terminated but not reaped by its parent. | |
|  |  |  | BSD 格式所附加的特性，附加显示在进程状态之后 | | |
|  |  |  | < | high-priority (not nice to other users) | |
|  |  |  | N | low-priority (nice to other users) | |
|  |  |  | L | has pages locked into memory (for real-time and custom IO) | |
|  |  |  | s | is a session leader | |
|  |  |  | l | is multi-threaded (using CLONE\_THREAD, like NPTL pthreads do) | |
|  |  |  | + | is in the foreground process group | |
| START |  |  |  | 启动进程的时间 | | |
| TIME | time | %x |  | 进程消耗CPU的时间 | | |
| COM  MAND |  |  |  | 命令的名称和参数，如果shell命令前有‘-’号，表示为登录所在的shell. | | |
| F |  |  |  | flags | | |
|  |  |  | 1 | | forked but didn't exec |
|  |  |  | 4 | | used super-user privileges |
| S |  |  |  | 即STAT | | |
| C |  |  |  | CPU使用的资源百分比 | | |
| PRI |  |  |  | 进程优先级Priority | | |
| NI | nice | %n |  | Nice 值 | | |
| ADDR |  |  |  | 该程序在内存的那个部分 | | |
| SZ |  |  |  | 使用掉的内存大小 | | |

### top

输出实时监控CPU利用率，进程状态和内存利用率。

* 参数含义

|  |  |
| --- | --- |
| 参数 | 含义 |
| d | 指定每两次屏幕信息刷新之间的时间间隔。 |
| p | 监控指定进程的ID |
| q | 使top没有任何延迟的进行刷新。 |
| S | 指定累计模式。 |
| s | 使top命令在安全模式中运行。 |
| i | 使top不显示任何闲置或者僵死进程。 |
| c | 显示整个命令行而不只是显示命令名。 |

* 运行时的交互命令

|  |  |
| --- | --- |
| 交互命令 | 功能 |
| Ctrl+L | 擦除并且重写屏幕。 |
| h/? | 显示帮助画面，给出一些简短的命令总结说明。 |
| k | 终止指定PID的进程并发送指定signal(默认为15, 强制kill可使用9)。 |
| I | 忽略闲置和僵死进程。这是一个开关式命令。 |
| q | 退出程序。 |
| r | 重新安排指定PID的进程优先级别(正值将使优先级降低)。 |
| f/F | 从当前显示中添加或者删除项目。 |
| o/O | 改变显示项目的顺序。 |
| l | 切换显示平均负载和启动时间信息。 |
| m | 切换显示内存信息。 |
| t | 切换显示进程和CPU状态信息。 |
| c | 切换显示命令名称和完整命令行。 |
| M | 根据驻留内存大小进行排序。 |
| Shift+M | 可按内存占用情况进行排序。 |
| P | 根据CPU使用百分比大小进行排序。 |
| T | 根据“时间/累计时间”进行排序。 |
| W | 将当前配置写入~/.toprc文件中。 |

* 输出格式

|  |  |
| --- | --- |
| **<line 1>任务队列信息,等价于uptime命令输出** | |
| 12:38:33 | 当前时间 |
| up 50days | 系统运行时间，格式为时:分 |
| 1 user | 当前登录用户数 |
| load average: 0.06, 0.60, 0.48 | 系统负载，即任务队列的平均长度。 三个数值分别为  1分钟、5分钟、15分钟前到现在的平均值。 |
| **<line 2~3>进程和CPU的信息** | |
| Tasks: 29 total | 进程总数 |
| 1 running | 正在运行的进程数 |
| 28 sleeping | 睡眠的进程数 |
| 0 stopped | 停止的进程数 |
| 0 zombie | 僵尸进程数 |
| Cpu(s): 0.3% us | 用户空间占用CPU百分比 |
| 1.0% sy | 内核空间占用CPU百分比 |
| 0.0% ni | 用户进程空间内改变过优先级的进程占用CPU百分比 |
| 98.7% id | 空闲CPU百分比 |
| 0.0% wa | 等待输入输出的CPU时间百分比 |
| 0.0% hi | cpu处理硬件中断的时间百分比 |
| 0.0% si | cpu处理软中断的时间百分比 |
| **<line 4~5>内存信息** | |
| Mem: 191272k total | 物理内存总量 |
| 173656k used | 使用的物理内存总量 |
| 17616k free | 空闲内存总量 |
| 22052k buffers | 用作内核缓存的内存量 |
| Swap: 192772k total | 交换区总量 |
| 0k used | 使用的交换区总量 |
| 192772k free | 空闲交换区总量 |
| 123988k cached | 缓冲的交换区总量。 内存中的内容被换出到交换区，而后又被换入到内存，但使用过的交换区尚未被覆盖， 该数值即为这些**内容已存在于内存中**的交换区的大小。相应的内存再次被换出时可不必再对交换区写入。 |
| **进程信息** | |
| 列名 | 含义 |
| PID | 进程id |
| PPID | 父进程id |
| RUSER | Real user name |
| UID | 进程所有者的用户id |
| USER | 进程所有者的用户名 |
| GROUP | 进程所有者的组名 |
| TTY | 启动进程的终端名。不是从终端启动的进程则显示为 ? |
| PR | 优先级 |
| NI | nice值。负值表示高优先级，正值表示低优先级 |
| P | 最后使用的CPU，仅在多CPU环境下有意义 |
| %CPU | 上次更新到现在的CPU时间占用百分比 |
| TIME | 进程使用的CPU时间总计，单位秒 |
| TIME+ | 进程使用的CPU时间总计，单位1/100秒 |
| %MEM | 进程使用的物理内存百分比 |
| VIRT | 进程使用的虚拟内存总量，单位kb。VIRT=SWAP+RES |
| SWAP | 进程使用的虚拟内存中，被换出的大小，单位kb。 |
| RES | 进程使用的、未被换出的物理内存大小，单位kb。RES=CODE+DATA |
| CODE | 可执行代码占用的物理内存大小，单位kb |
| DATA | 可执行代码以外的部分(数据段+栈)占用的物理内存大小，单位kb |
| SHR | 共享内存大小，单位kb |
| nFLT | 页面错误次数 |
| nDRT | 最后一次写入到现在，被修改过的页面数。 |
| S | 进程状态。详见ps命令说明 |
| COMMAND | 命令名/命令行 |
| WCHAN | 若该进程在睡眠，则显示睡眠中的系统函数名 |
| Flags | 任务标志，参考 sched.h |

### 系统状态

|  |  |  |  |
| --- | --- | --- | --- |
| 工具 | 功能 | | |
| uname | 获取系统OS版本，等价于cat /proc/version | | |
| lsb\_release | 获取系统发布信息 | | |
| getconf | 获取系统配置 | | |
| LONG\_BIT | | 获取long位数 |
| WORD\_BIT | | 获取word位数 |
| date | 系统时间，格式转换 比如：date “+%Y%M%S” | | |
| 任务状态 | | | |
| ps auxZ | 其中Z表示显示Security的lable，  ps -eo pcpu,pid,user,args | sort -k 1 -r | head -16，输出并排序 | | |
| top | 查看进程活动状态以及一些系统状况 | | |
| uptime | CPU平均负载 | | |
| pidof | pidof [选项] <进程名称>，查找一个运行的程序的PID | | |
| -s | 表示只返回1个 pid | |
| -x | 表示同时返回运行给定程序的 shell 的 pid | |
| -o | 忽略给定的 pid ，可以使用多个 -o | |
| CPU状态 | | | |
| vmstat | 查看系统状态、硬件和系统信息等 | | |
| iostat | 查看CPU 负载，硬盘状况 | | |
| sar | 综合工具，查看系统状况 | | |
| mpstat | 查看多处理器状况(需要安装sysstat软件包) | | |
| 网络状态 | | | |
| netstat | 查看网络状况 | | |
| iptraf | 实时网络状况监测 | | |
| tcpdump | 抓取网络数据包，详细分析 | | |
| mpstat | 查看多处理器状况 | | |
| tcptrace | 数据包分析工具 | | |
| netperf | 网络带宽工具 | | |
| dstat | 综合工具，综合了 vmstat, iostat, ifstat, netstat 等多个信息 | | |

### SELinux

* 查看SELinux信息

1. 使用命令sestatus –v 查看工作状态
2. 使用命令 getenforce 查看工作模式

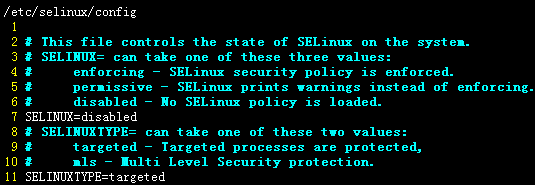
* 配置SELinux的方法

1. 使用命令

setenforce 1 设置工作模式为enforcing；

setenforce 0 设置工作模式为permissive。

1. 修改配置文件



1. 修改启动参数

在lilo或者grub的启动参数中增加：selinux=0则关闭SELinux。

* 应用配置

所有的更改都需要重启机器，只有一个例外：在SELinux启用状态下，设置为permissive模式(即SELinux实际不起作用，仅发出Warning)。

### 数学计算

* 使用let

var=1  
let "var+=1"  
echo $var  
输出结果为2

注意：

1. 方幂运算应使用“\*\*”;
2. 参数在表达式中直接访问，不必加$;算数表达式可以不加双引号，但若表达式中有shell关键字则需加上;
3. 只能进行整数运算。

* 使用(())

var=1  
((var+=1))  
echo $var

输出结果为2

注意：使用方法与let完全相同。

* 使用$[]

var=1

var=$[$var+1]

echo $var

输出结果位2

注意：

1. 中括号内的表达式作为数学运算将先计算出结果后再输出;
2. 在$[]中访问变量时前面需要加$;
3. 支持的运算符与let相同，但也只支持整数运算。

* 使用expr

var=1

var=`expr $var + 1`

echo $var

输出结果为2

注意：

1. 表达式各符号间需用空格隔开;
2. 支持的操作符有： |、&、<、<=、=、!=、>=、>、+、-、\*、**/**、%;
3. 使用时需用\进行转义的有：|、&、<、<=、>=、>、\*;
4. 只支持整数运算。
5. expr也字符串相关操作,如下：

|  |  |
| --- | --- |
| 表达式 | 功能 |
| STRING **:** REGEXP | anchored pattern match of REGEXP in STRING |
| match STRING REGEXP |
| index STRING CHARS | index in STRING where any CHARS is found, or 0 |
| substr STRING POS LENGTH | substring of STRING, POS counted from 1 |
| length STRING | length of STRING |

* 使用bc

var=1

var=`echo "$var+1"|bc`

echo $var

输出结果为2

注意：

1. 支持除位操作运算符之外的所有运算符；
2. 支持浮点数计算，可使用scale进行精度设置，如下：

var=3.14

var=`echo "scale=2;$var\*3"|bc`

echo $var

输出结果为9.42

* 使用awk

var=1

var=`echo "$var 1"|awk '{printf("%g",$1\*$2)}'`

echo $var

输出结果为2

注意：

1. 支持除位操作运算符之外的所有运算符;
2. 内置有log、sqr、cos、sin等函数;
3. 支持浮点数计算。

### 网络管理

* 修改IP信息

ifconfig eth0

* 修改IP地址

ifconfig eth0 192.168.168.64 netmask 255.255.255.0

* 配置网卡地址

ifconfig eth0 hw ether xx：xx：xx：xx：xx：xx

* 禁用网卡

ifconfig eth0 down 或 ifdown eth0

* 启用网卡eth0

ifconfig eth0 up  或 ifup eth0

### wget

* wget的使用格式

wget [OPTION]... <URL>...

* 用wget做站点镜像:

wget -r -p -np –k <URL>  
 或者：wget -m <URL>

* 设置代理的方法

1. 通过shell命令行配置代理环境变量xxx\_proxy

set “http\_proxy=http://[user]:[pass]@host:port/”

1. 通过配置文件${HOME}/.wgetrc

增加：http\_proxy=http://[user]:[pass]@host:port/

1. 通过参数传入代理配置xxx\_proxy  
   wget -e "http\_proxy=http://[user]:[pass]@host:port/" <URL>

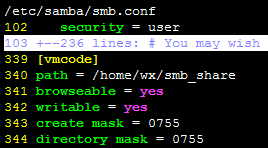
### smba服务器

* 创建smba账号

sudo smbpasswd –a username

* 增加smba访问目录

增加配置如下：



* 启动smba服务

sudo /etc/init.d/smbd restart

* 访问smba共享文件夹

访问路径：[\\192.168.40.190\vmcode](file:///\\192.168.40.190\vmcode)，则可以访问服务器上vmcode对应的path.

### ssh配置

用于与 ssh服务器连接，其相关配置如下：

* ssh密钥创建

分为两个部分，一个存放在服务器的公有密钥和一个存放在客户端的私有密钥。

1. 查看默认的私有密钥$HOME/.ssh/id\_rsa是否存在。
2. 密钥文件创建

ssh-keygen -t rsa -C [your\_email@example.com](mailto:your_email@example.com)

在创建期间，会要求输入用于加密本地私有密钥文件的passphrase，以及生成的密钥文件所存放的路径(默认为：$HOME/.ssh/)。

执行后，生成私有密钥文件id\_rsa，以及公有密钥文件id\_rsa.pub.

1. 配置公有密钥

将id\_rsa.pub文件的内容配置到服务器端。可以使用clip命令将公有密钥文件中的内容拷贝到剪切板，然后方便配置给服务器端。

clip < ~/.ssh/id\_rsa.pub

1. 连接测试

配置完后，在客户端使用ssh命令进行服务器连接测试。

ssh -T git@github.com

* 修改密钥文件位置

如果已经存在私有密钥文件，则使用ssh-add命令将对应路径配置为ssh客户端的默认私有密钥，如果没有配置，则私有密钥路径为$HOME/.ssh/id\_rsa。

ssh-add ~/.my\_other\_ssh/id\_rsa -l

* 创建或修改私有密钥的passphrase

私有密钥使用ssh-keygen命令对私有密钥文件进行加密，或者修改加密密码。

ssh-keygen -p

在创建过程中，会提示密钥文件的本地密码。

* 通过https访问ssh服务器

如果ssh端口被屏蔽，则可以通过https端口进行ssh连接：

ssh -T -p 443 git@ssh.github.com

或者修改配置文件~/.ssh/config：

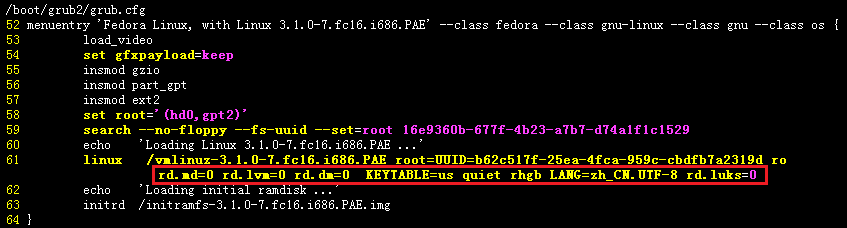
Host github.com

Hostname ssh.github.com

Port 443

### grub配置

参数设置如下如下：



其中：

* set root <存储设备编号> 表示引导的系统安装在哪个存储设备中。
* linux　<内核镜像文件>　用于指定加载的内核镜像文件名称。
* root=<分区UUID>，表示VFS根目录挂载点使用的是哪个分区，ro表示该分区是只读的。
* 红色方框区域为内核启动参数，格式为：参数名＝值，多个参数之间用空格分隔，如：LANG=zh\_CN　表示系统环境变量LANG(表示默认系统语言)的值设置为zh\_CN。

### 启动方式配置

* 通过编辑/etc/inittab文件实现图形界面和文本模式的切换。把其中的id:3:initdefault中的3改为5，则重启后将进入图形界面，反之则进入文本模式。
* 在字符界面输入startx或init 5则进入图形界面；在图形界面打开终端输入输入init 3则进入文本模式 。
* 使用Ctrl+Alt+F1~6切换到字符界面，再用Ctrl+Alt+F7切换到图形界面；如果是虚拟机，由于屏蔽了Ctrl+Alt键，必须使用其他热键：Ctrl+Alt+shift+F1~6切换到字符界面；使用Alt+F7返回到图形界面。

### 修改root密码

通过将bootloader引导kernel的runlevel配置为single模式实现。进入single后可直接用passwd命令重置root密码。

* 对于gurb，在kernel参数行增加字符1(空格分开)，对应runlevel的single模式。
* 对于 Redhat，在kernel参数行增加字符串single。

## coding

### ctag

### cscope

### taglist

### SouceInsight

* 快捷键

|  |  |
| --- | --- |
| 快捷键 | 功能 |
| ctrl + o | open file |
| Shift + F8 | high light selected |
| alt + < | jump to preview window |
| ctrl + = | jump to define |
| alt + L | symble list window in the left |

### eclipse

### 代码调用关系

* CodeViz

生成代码调用关系图的工具关系图。

下载地址：  
<http://www.csn.ul.ie/~mel/projects/codeviz/>

### 在线编辑

* LXR

Linux cross reference以web的方式阅读和查找LINUX内核源代码的工具。

可直接到官方网站直接读代码，<http://lxr.linux.no/linux+v2.6.24/>。

### Doxgen

* 功能

doxgen通过对源码的特定格式的注释进行解析，生成对应的API文档。

* 注释类型

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类型 | | 格式 | | |
| visible comment  (doxgen可见注释) | block | 注释块。以“**/\*!**”或者“**/\*\***”开始，以“**\*/**”结束，始末标示符必须各独占一行，不能有其他字符。 | | |
| single line | 注释后方代码 | | 以“**//!**”或“**///**”开始。 |
| 以“**/\*!**”开始，以或“**\*/**”结束。 |
| 注释前方代码 | | 以“**/\*!<**”开始，以或“**\*/**”结束。 |
| 注释项 | field | 以“@”或“**\**”开始的单词，比如@param，\param等效。 | | |
| version | 版本 | |
| remark | 备注 | |
| file | 文件名 | |
| brief | 简介 | |
| retval | @retval <ret\_val> {description} | |
| param | @param <param\_name> {description} | |
| author | 作者 | |
| date | 日期 | |
| warning | 警告 | |
| exception | 异常情况 | |
| 换行 | 在注释行尾增加“\n”字符，则对应文档内容会换行 | | |
| 模块分组 | header | 定义 | **/\*!**@defgroup <label> “display text”**\*/** | |
| 添加 | **/\*!**@addtogroup <label> “display text” **\*/** | |
| module | 紧跟分组头之后，代码模块以“**/\* @{\*/**”开始，以“/**\* @}\***/”结束。 | | |
| 引用项 | @ref <label> “display text” | | | |
| 排序项 | 用于需要排列的项目，比如对retval进行说明。 | | | |
| ordered | -# {description} | | |
| unordered | - - {description} | | |

* 创建文档

1. wizard🡪project:配置项目名称，源码目录，文档输出目录。
2. wizard🡪mode:选择documented entities only，对应源码类型优化。
3. wizard🡪output:选择plain html和pdf格式。
4. wizard🡪diagrams:选择无。
5. run🡪run doxgen。

## VI

### 查看VI信息

* 查看版本信息

version可以看到vim相关配置信息。

比如配置文件的路径，安装路径等。

* 查找映射信息

verbose map <所要查找的快捷键>

比如 verbose map <tab> 或者 verbose map <C-c>

* 查找特殊符号信息

:help keycodes，可以看到Ctrl，换行等特殊符号的完整列表。

* 查找函数信息

:h function 列出所有函数和它们的参数；

:h {name} 列出{name}命名的函数，以及它的定义；

:h **/**{pattern} 列出名字匹配{pattern}的函数。

* 查找变量

:h variable；列出变量类型

:h v:var 列出vim变量，类似有g:var，l:var，w:var，s:var。

* 文件名显示

1. 在命令行中输入：echo expand(“%:r”) 符号%是代表粘贴板号
2. echo bufname(“%”)

* VIM寄存器查看

reg: <寄存器名称>

### 跳转和移动

可通过如下方式实现跳转：

* 在Normal模式下

1. 直接敲击数字，则跳到对应行；
2. 敲击gg 跳到文件的开始，G 跳到文件的结束；
3. 组合键Ctrl+u , Ctrl+d 跳转到上一页，下一页；
4. HJKL实现左下上右的移动；
5. 敲击10gg 或10G 则跳到第10行。

### 查找

* 在CommandLine模式下

1. 命令行中输入：**/**要查找的字符串，回车，则高亮显示第一个匹配的字符串；
2. 然后敲击n则高亮下一个匹配，敲击N则高亮上一个匹配。

* 在Normal模式下

敲击#或者\*(数字键38的特殊符号)则查找与当前光标所在单词的下**/**上一个。

### 替换

* 命令行输入格式:[range]s**/**{pattern}**/**{string}**/**[flag]

1. 冒号':'是这一类命令的开始；
2. [range]表示命令的作用域，即命令起作用的行的范围(闭区间)，通常是两个整数，表示作用域起始行和结束行(可以用 ‘$’表示最后一行)的行号，中间以一逗号隔开, 如某个数省略不写则代表光标所在行。也可以用一个‘%’表示文档的所有行;
3. ‘s’是替换命令substitute的简写，其后第一个字符(任意字符)将解释为分隔符；
4. {pattern}和{string}分别为待搜索的模式串和所要替换成的替换串，支持正则表达式, 如果{pattern}为空串，则使用上一条替换命令中的 {pattern}作为模式串；
5. ‘**/**’表示{pattern}和{string}的分隔符，如果{pattern}和{string}也出现分隔符要使用反斜杠‘\’进行转义；
6. [flag]表示标志位，用来开启或关闭一些选项。

|  |  |  |
| --- | --- | --- |
| flag | 功能 | 命令格式 |
| ‘g’ | global,全局替换 | %s/要替换的/替换后的/g |
| ‘c’ | confirm, 确认替换 | %s/要替换的/替换后的/c |
| ‘i’ | ignore, 忽略大小写 | %s/要替换的/替换后的/i (需要匹配模式为不忽略大小写) |

* 全字匹配替换

将{pattern}放入尖括号< >则代表全字匹配(需要转义)。

比如：%s**/**\<要替换的\>**/**替换后的**/**g,其中\< \> 即表示全字匹配。

### 字符匹配模式

* 大小写匹配(全局)

在使用查找、替换或者补全命令时被当作默认匹配模式。

1. 在命令行中输入：set ic或者set ignorecase 则忽略大小写；
2. 在命令行中输入：set noic或者set noignorecase 则不忽略大小写。

* 大小写匹配(单个命令)

1. 查找命令后加\c: **/**要查的字符串\c 则忽略大小写；
2. 替换命令的flag增加i选项：%s**/**要替换的**/**替换后的**/**i则忽略大小写。

### 排版效果

* 显示行号set nu 或者 set nonu
* 自动缩进set cindent/nocindent
* 自动换行(默认开启) set wrap/nowrap
* 折叠命令flod

**:**3,9flod第3行与第9行折叠起来。

* 自动折叠set fdm=XXX

foldmethod可在文件中使用注释调用vim命令 /\* vim: set fdm=XXX: \*/

具体查看help fold信息，折叠类型如下：

|  |  |
| --- | --- |
| fdm | 含义 |
| indent | 自动折叠大括号的中间部分 |
| expr | 用表达式来定义折叠 |
| syntax | 用语法高亮来定义折叠 |
| diff | 对没有更改的文本进行折叠 |
| marker | 对文中的标志折叠，默认折叠{{{和}}}之间的内容 |
| manual | 人工模式 |

* 折叠区命令(maker和manual方式)

|  |  |  |
| --- | --- | --- |
| 命令 | 功能 | |
| zf | zf7G | 当前行到全文第7行。 |
| 10zf | 当前行后10行，等价命令10zf+/zf10↓。 |
| 10zf- | 当前行前10行，等价命令zf10↑。 |
| zf% | 在括号处使用，从当前行起到对应匹配括号所在行 |
| zfap | 当前行所在段落 |
| shif＋v | 用光标选中的区域，然后按zf |
| zd | 删除折叠(光标之下)。 | |
| zD | 嵌套删除折叠(光标之上)。 | |
| zE | 除去 (Eliminate) 窗口里“所有”的折叠 | |

* 折叠区扩展命令(manual)

|  |  |
| --- | --- |
| 命令 | 功能 |
| zc | 折叠当前折叠区 |
| zC | 折叠当前行所长范围内所有折叠区 |
| zo | 展开当前折叠点 |
| zO | 展开所在范围内所有折叠点 |
| [z | 移动光标到当前打开的折叠的开始处 |
| ]z | 移动光标到到当前打开的折叠的末尾处 |
| zj | 移动光标到到下一个折叠的开始处，关闭的折叠也被计入。 |
| zk | 移动光标到到前一折叠的结束处，关闭的折叠也被计入。 |

### 显示效果

* 语法高亮

|  |  |  |
| --- | --- | --- |
| 显示类型 | 命令 | 含义 |
| 句法高亮 | set syntax=c | 设置句法高亮类型为c语言(其他如html,makefile) |
| syntax on/off | 打开或关闭句法高亮显示 |
| syntax reset | 对句法高亮的配置进行重置 |
| 搜索高亮 | set hlsearch | 打开高亮搜索 |
| set nohlsearch | 关闭高亮搜索 |
| nohl/noh | 消掉当前的搜索高亮 |

* 文件类型

1. 查看文件类型检测

在命令行中输入：filetype 可以查看文件类型检测是否打开

如果filetype命令没有效果，修改安装目录下的filetype.vim， 找到对应的文件类型，在文件名匹配项中加入你需要的模式，前提是已经有了对应的高亮类型配置文件。

1. 检测当前文件语法类型

在命令行中输入：set filetype 可以查看文件类型检测结果

1. 设置当前文件的语法检测类型

在命令行中输入：set ft=文件类型名称 可设置文件采用哪种类型语法高。

(ft是filetype的缩写)

1. 设置默认的文件类型

修改配置文件，指定文件名与文件类型的联系，如下:



* 设置配色主题

colorscheme 主题名 或 colo (主题名)，常用的主题有 morning evening，可用的主题可以查看色彩配置文件夹colors下的脚本。

* 添加配色主题

对于root用户，直接将配色文件放到vim安装路径的colors目录下，对于普通用户，创建~/.vim/colors，然后将配色文件放进去。修改配色文件后可使用hi命令查看配色效果。

* 颜色设置

如果是在gvim下，则用 gui(图形用户接口)，如果在终端则用cterm(彩色终端)即可，以下以终端为例，如果要配置gvim，把cterm替换为gui即可。

每个颜色可分为三个等级，深(dark)，普通，浅色(light)。

查看所有的已有设置，直接执行highlight命令即可，不带任何参数。

1. 背景颜色

在命令行中输入：set background=light 可选值还有：dark，darkblue等；

1. 行号背景颜色

在命令行中输入：hi LineNr ctermfg=red则改变行号的背景色为红色，hi为highlight的简写；

1. 行号前景颜色

在命令行中输入：hi LineNr ctermbg=white 则改变行号字体颜色为白色。

* 设置无效的问题

在vim的脚本中有时执行一些 scheme的设置会导致设置无效，此时可以注释掉这些代码，手动在命令行模式下输入对应命令，可跟踪调试。

### 选取

* 按v进入可视化模式，移动方向键选中文本，之后的命令则只作用于选中区域；
* 按命令前加入行号范围，比如10,12“命令”，表示对10到12行执行“命令”；

### 复制粘贴

* 在插入模式下

组合键shift+Insert粘贴 “系统粘贴板”的内容到光标处；

* 在Normal模式下

1. p则把 “默认粘贴板”的内容粘贴到光标所在处；
2. dd可以剪切整行到“默认粘贴板”；
3. yy可以拷贝整行到“默认粘贴板”；
4. yG拷贝当前位置到文件结尾到“默认粘贴板”；
5. ygg拷贝当前位置到文件起始到“默认粘贴板”；
6. yw 表示拷贝从当前光标到光标所在单词结尾的内容；
7. y0 表示拷贝从当前光标到光标所在行首的内容；
8. y$ 表示拷贝从当前光标到光标所在行尾的内容；
9. yfa 表示拷贝从当前光标到光标后面的第一个a字符之间的内容。

* 在CommandLine模式下

命令行中输入：1,3y 复制1到3行到“默认粘贴板”。

## cmd

### 命令调用

* cmd /c xxx 是执行完xxx命令后关闭命令窗口。
* cmd /k xxx是执行完dir命令后不关闭命令窗口。
* cmd /c start xxx会打开一个新窗口后执行xxx指令，原窗口会关闭。
* cmd /k start xxx会打开一个新窗口后执行xxx指令，原窗口不会关闭。

## vim

用于vim编辑器的功能描述。

### 命令定义

* 自动命令

au[tocmd] [group] {event} {pat} [nested] {cmd}

### 变量定义

* 局部变量

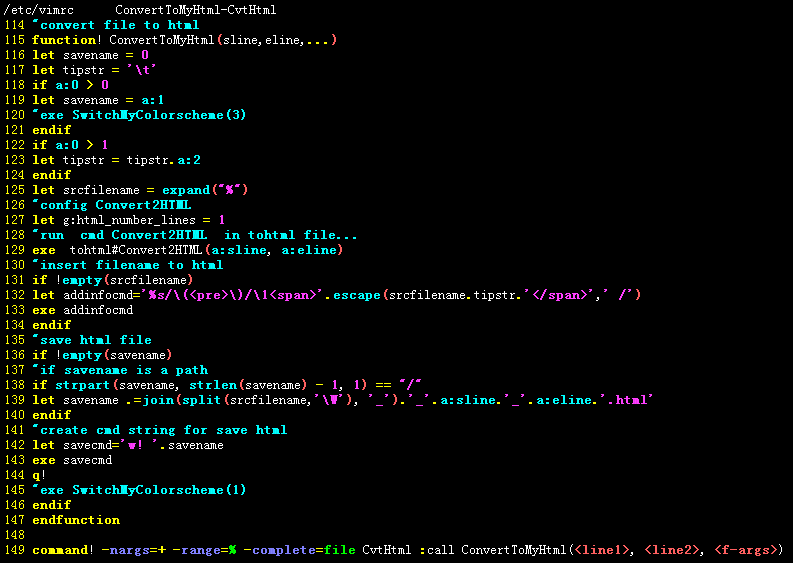
let

### HTML转换

* 使用系统函数

1. 命令行中输入：TOhtml 则把整个文件都转换为html；
2. 命令行中输入：32,40TOhtml 则把32行至40行转为html；
3. 如果有中文字符，则需要先设置:let html\_use\_encoding='gb2312'；

* 使用自定义函数



1. 函数名必须以大写字母开头。
2. 命令行中输入：32,40CvtHtml **/**mypath**/**，则把32行至40行转为html并保存到**/**mypath目录下
3. 代码解释：

118: a:0 参数个数，a:1第一个参数，依次类推；

125: expand 获取寄存器值；

125: % 系统寄存器名称，表示当前编辑的文件名；

129: tohtml#Convert2Html ‘#’表示引用插件tohtml(文件名)的Convert2Html函数;

132: 字符.(dot) 字符串连接；

132: escap内置函数,如果第二个参数中的字符在第一个参数中出现，则用\进行转义;

149: nargs=+ 命令参数个数类型为+(大于1), \*(任意)；

149: range=% 命令可作用范围的行号为%(整个文件)；

149: complete=file 输入参数时候按tab自动补全的类型为文件路径；

149: <line-1>,<line-2>系统保留变量，指命令的作用范围的行号；

149: <f-args>系统保留变量，指命令的参数序列，各参数用‘空格’或‘，’分开，如果参数中需要特殊字符，则需要使用‘\’转义，否则会被作为参数之间的分隔符；类似的还有<q-args>，表示除过<q-args>之前的输入，剩余所有输入作为一个参数，即不会对参数分割，所以也不需转义。

### 脚本教程

[http:**//**www.ibm.com**/**developerworks**/**cn**/**linux**/**l-vim-script-1**/**index.html?ca=drs-cn-0601](http://www.ibm.com/developerworks/cn/linux/l-vim-script-1/index.html?ca=drs-cn-0601)

## Hacking

### Rootkit

简介: 在这篇文章里, 我们将看到各种不同的后门技术，特别是Linux的可装载内核模块(LKM)。 我们将会发现LKM后门比传统的后门程序更加复杂，更加强大，更不易于被发现。知道这些之后，我们可以制造我们自己的基于LKM的Rootkit程序, 主要体现在TCP/IP层, 因为我们相信这是在系统管理员面前最好的隐藏后门的地方。

序言

在一些黑客组织中, Rootkit (或者backdoor) 是一个非常感兴趣的话题。 各种不同的Rootkit被开发并发布在internet上。在这些Rootkit之中, LKM尤其被人关注, 因为它是利用现代操作系统的模块技术。作为内核的一部分运行,这种Rootkit将会越来越比传统技术更加强大更加不易被发觉。一旦被安装运行到目标机器上, 系统就会完全被控制在hacker手中了。甚至系统管理员根本找不到安全隐患的痕迹, 因为他们不能再信任它们的操作系统了。

本文章以及我们开发的一些强大的LKM程序都是基于Linux Kernel 2.2.x版本的。我们的目的是尽可能多的隐藏足迹。

在接下来的一部分, 我们将介绍一下已经存在的后门技术, 然后和LKM技术相比较, 最后讨论我么的LKM程序的设计与实现。

后门程序的目的就是甚至系统管理员企图弥补系统漏洞的时候也可以给hacker系统的访问权限。后门程序使本地用户取得root权限可以这样做: 设置uid程序, 系统木马程序, cron后门。

1. 设置uid程序。 黑客在一些文件系统理放一些设置uid脚本程序。无论何时它们只要执行这个程序它们就会成为root。

2. 系统木马程序。黑客替换一些系统程序, 如"login"程序。因此, 只要满足一定的条件，那些程序就会给黑客最高权限。

3. Cron 后门。黑客在cron增加或修改一些任务, 在某个特定的时间程序运行，他们就可以获得最高权限。

后门程序给远程用户以最高访问权限可以这样做: ".rhost" 文件, ssh认证密钥, bind shell, 木马服务程序。

1. ".rhosts" 文件。一旦 "+ +"被加入某个用户的.rhosts文件里, 任何人在任何地方都可以用这个账号来登陆进来而不需要密码。

2. ssh 认证密钥。黑客把他自己的公共密钥放到目标机器的ssh配置文件"authorized\_keys"里, 他可以用该账号来访问机器而不需要密码。

3. Bind shell。黑客绑定一个shell到一个特定的tcp端口。任何人telnet这个端口都可以获得交互的shell。更多精巧的这种方式的后门可以基于udp,或者未连接的tcp, 甚至icmp协议。

4. Trojaned服务程序。任何打开的服务都可以成为木马来为远程用户提供访问权限。例如, 利用inetd服务在一个特定的端口来创建一个bind shell，或者通过ssh守护进程提供访问途径。

在入侵者植入和运行后门程序之后, 他会找一些方法和系统管理员开一些善意的玩笑。这主要涉及到两个方面问题: 如何来隐藏他的文件且如何来隐藏他的进程。

为了隐藏文件, 入侵者需要做如下事情: 替换一些系统常用命令如"ls", "du", "fsck"。在底层方面, 他们通过把硬盘里的一些区域标记为坏块并把它的文件放在那里。或者如果他足够疯狂，他会把一些文件放入引导块里。

为了隐藏进程, 他可以替换 "ps"程序, 或者通过修改argv[]来使程序看起来象一个合法的服务程序。有趣的是把一个程序改成中断驱动的话，它就不会出现在进程表里了。

# Programing

## GCC

### 符号命名规则

* 对于C语言源代码所有全局量和函数编译后在符号名前加上前缀"\_"。
* 对于C++语言：

1. 所有符号以“\_Z”开头
2. 在命名空间或者类内的符号其标识为：N<命名空间名字><类的名字><符号名字>E，其中<XXX>前都会加上XXX符号长度。
3. 对于函数，最后附加参数的首字母。

例如：N::C::func(int)----------> \_ZN1N1C4funcEi。

1. 使用extern “C”{} 可以使得其中的代码当作C代码处理，则其命名规则与C相同。

### GCC内置函数

* 获取调用地址

void \* \_\_builtin\_return\_address( unsigned int level );

level参数表示返回地址的调用堆栈级别。例如，如果指定 level 为 0，那么就是请求当前函数的返回地址。如果指定 level 为 1，那么就是请求进行调用的函数的返回地址。

* 常量检测

int \_\_builtin\_constant\_p( exp )

用于判断一个值是否是常量，方便编程者对代码优化。

注意：GCC并不能检测出所有常量。

* 分支预测

int \_\_builtin\_expect( exp , val )

使用该函数编译器可以做出符合提供的预测信息的指令选择决策。这使执行的代码尽可能接近实际情况，还可以改进缓存和指令流水线。

例如：

#define likely(x) \_\_builtin\_expect(!!(x), 1)

#define unlikely(x) \_\_builtin\_expect(!!(x), 0)

如果一个条件标上了 “likely”，那么编译器可以把代码的 True 部分直接放在分支指令后面，从而少了一个分支跳转指令。按照这种方式，代码对于最可能(likely)出现的情况是最优的。

* 预取

void \_\_builtin\_prefetch( const void \*addr, int rw, int locality )

参数定义：

addr：数据的地址

rw：数据是用于执行读操作，还是执行写操作

locality：数据使用之后应该留在缓存中，还是应该清除

在需要数据之前，使用这个函数把数据放到缓存中。从而减少缓存缺失和停顿，提高性能。

### GAS伪指令

* 段表示

|  |  |
| --- | --- |
| 段名 | 功能 |
| .data | 数据段 |
| .bss | block started by symbol，符号开始段 |
| .text | 代码段 |

* 数据表示

1. 数据段定义

|  |  |  |
| --- | --- | --- |
| 段名 | 数据定义 | 示例 |
| .byte | 单字节数据 | .byte 1,2,0b01,0x34,072,'s' |
| .short | 双字节数据 | .short 0x1234,60000 |
| .long | 4字节数据 | .long 0x12345678,23876565 |
| .quad | 8字节数据 | .quad 0x1234567890abcd |
| .float | 浮点数 | .float 0f-31415926535897932384626433832795028841971.693993751E-40 |
| .string | 多个字符串 | .string "abcd", "efgh", "hello!" |
| .asciz | 多个字符串 | .asciz "qwer", "sun", "world!" |
| .ascii | 多个字符串 | .ascii "welcome\0" ，注：需要自行添加结尾字符'\0' |
| .space | 连续的数据并填充 | .space |

1. 重复数据定义伪操作

格式如下：

.rept 重复次数

数据定义

.endr @结束重复定义

例如定义3个0x23：

.rept 3

.byte 0x23

.endr

1. 替换

.equ/.set:，类似C代码中的宏定义, 格式如下：

.equ(.set) 变量名,表达式

例如使用abc替换3：

.equ abc 3

* 符号信息

1. .global(或.globl)

定义全局符号，例：.global \_start。

1. .align

用来指定数据的对齐方式,格式如下：

.align [absexpr1, absexpr2]

在未使用的存储区域填充值使地址对齐， 第一个值表示对齐方式(4, 8,16或 32)， 第二个表达式值表示填充的值。

1. .size

设置符号的大小，单位为字节。

1. .lcomm

设置符号的大小，单位为字节，在.bss段中声明未初始化变量的内存空间（适用于全局和静态变量的声明）。

例：.bss

.lcomm buffer, 256

* 符号类型表示

有五种写法

.type <name>, %<type-name>

.type <name>, #<type-name>

.type <name>, @<type-name>

.type <name>, “<type-name>”

.type <name>, %<type-name>

该属性在ELF格式(.out文件)中使用，用于描述符号类型。其中可支持的type-name为:function(函数类型)，object(数据类型)。

例：

.type func, @function

func:

表示func符号为function类型

* 指令所在行的地址

使用小数点(.)表示

* 模块包含

.include 文件

外部文件的调用

例：

.include "outside.s"

* 注释行

#号开头用于单行  
/\* \*/  
用于多行。

### 符号属性

GCC的\_\_attribute\_\_机制用于设置函数、变量和类型的扩展属性。格式如下：

<symbol> \_\_attribute\_\_ ((<attribute-list>)) **;**

* 书写格式

\_\_attribute\_\_前后有两个下划线，后面双括弧里是属性列表。其位于声明语句<symbol>之后，分号(;)之前。

多个属性可以使用逗号(,)隔开或者使用多个\_\_attribute\_\_语句以空白键(空格或者换行等)隔开。

* 函数属性

函数属性可以把一些特性添加到函数声明中，用于编译器错误检查。GNU CC使用 –Wall编译器来击活该功能。函数属性的格式如下：

<return-type function-name(param-list)> \_\_attribute\_\_((<attribute-list>)) **;**

常见的函数属性如下：

1. format(archetype, string-index, first-to-check)

编译器按照printf, scanf, strftime或strfmon的参数列表格式规则对该函数的参数进行检查。

“archetype”指定是哪种风格，默认情况下，编译器只能识别类似printf的风格；

“string-index”指定传入函数的第几个参数是格式化字符串；

“first-to-check”指定从函数的第几个参数开始按上述规则进行检查。

具体使用格式如下：

\_\_attribute\_\_((format(printf,m,n)))

\_\_attribute\_\_((format(scanf,m,n)))

其中参数m与n的含义为：

m：第几个参数为格式化字符串（format string）；

n：参数集合中的第一个，即参数“…”里的第一个参数在函数参数总数排在第几。

比如：

extern void myprint(const char\*format,...) attribute\_\_((format(printf,1,2)));

1. noreturn

表示函数没有返回值，当遇到类似函数需要返回值而却不可能运行到返回值处就已经退出来的情况，该属性可以避免出现错误信息。C库函数中的abort()和exit()的声明格式就采用了这种格式，如下所示：

extern void exit(int) \_\_attribute\_\_((noreturn));

extern void abort(void) \_\_attribute\_\_((noreturn));

1. const

该属性只能用于带有数值类型参数的函数上。当重复调用带有数值参数的函数时，由于返回值是相同的，所以此时编译器可以进行优化处理，除第一次需要运算外，其它只需要返回第一次的结果就可以了，进而可以提高效率。

该属性主要适用于没有静态状态（static state）和副作用的一些函数，并且返回值仅仅依赖输入的参数。

该属性不但影响函数的参数值，同样也影响到了参数指向的数据，它可能会对代码本身产生严重甚至是不可恢复的严重后果，所以不能用在带有指针类型参数的函数中。

1. constructor/destructor

若函数被设定为constructor属性，则该函数会在main()函数执行之前被自动的执行。类似的，若函数被设定为destructor属性，则该函数会在main()函数执行之后或者exit()被调用后被自动的执行。拥有此类属性的函数经常隐式的用在程序的初始化数据方面。

这两个属性还没有在面向对象C中实现。

1. always\_inline

让 GCC 以内联方式处理指定的函数，无论是否启用了优化

1. deprecated

 指出函数已经被废弃，不应该再使用。如果试图使用已经废弃的函数，就会收到警告。还可以对类型和变量应用这个属性。

1. \_\_used\_\_

告诉编译器无论 GCC 是否发现这个函数的调用实例，都要使用这个函数。这对于从汇编代码中调用 C 函数有帮助。

1. warn\_unused\_result

让编译器检查所有调用者是否都检查函数的结果。这确保调用者适当地检验函数结果，从而能够适当地处理错误。

* 变量属性

变量属性的格式如下：

<variable-type variable-name> \_\_attribute\_\_((<attribute-list>)) [=init value]**;**

常见的变量属性如下：

1. aligned (alignment)

指定变量或结构体成员的最小对齐格式，以字节为单位。如果aligned后面没有指定数值，那么编译器将依据目标机器情况使用最大最有益的对齐方式。

1. packed

指定变量或者结构体成员使用最小的对齐方式，即对变量是一字节对齐，对域（field）是位对齐。

* 类型属性

对结构体（struct）或共用体（union）进行属性设置，类型属性的格式如下：

< type define> \_\_attribute\_\_((<attribute-list>)) **;**

与变量属性类似，区别在于对所有成员有效。

### 汇编语法风格

ARM官方汇编语法风格与Intel风格类似但不完全相同， gcc的gas采用了AT&T风格，但对于ARM版本的gas其和官方语法一致，如下表：

* 寄存器名

在AT&T汇编格式中，寄存器名要加上'%'作为前缀；而在intel汇编格式中，寄存器名不需要加前缀。例如：

|  |  |  |
| --- | --- | --- |
|  | AT&T | Intel |
| X86 | pushl %eax | push eax |
| ARM | mov r3 #2 | |

* 立即操作数

在AT&T汇编格式中，用’$’  前缀表示一个立即操作数；而在intel 汇编格式中，立即数的表示不用带任何前缀。例如：

|  |  |  |
| --- | --- | --- |
|  | AT&T | Intel |
| X86 | pushl $3 | push 3 |
| ARM | mov r0 #2 | |

* 操作数位置

在Intel 汇编格式中， 目标操作数在源操作数的左边；而在AT&T 汇编格式中，目标操作数在源操作数的右边。例如：

|  |  |  |
| --- | --- | --- |
|  | AT&T | Intel |
| X86 | add $2, %eax | add eax, 2 |
| ARM | orr dest, r0,#2 | |

* 操作数的字长

在AT&T 汇编格式中，操作数的字长由操作符的最后一个字母决定，后缀’b’、’w’、’l’分别表示操作数为字节（byte，8 比特）、字（word，16 比特）和长字（long，32比特）；而在Intel 汇编格式中，操作数的字长是用”byte ptr” 和”word ptr” 等前缀来表示的。例如：

|  |  |  |
| --- | --- | --- |
|  | AT&T | Intel |
| X86 | movb val, %al | mov al, byte ptr val |
| ARM | ldrb r0 ,[r2]  strb r0, [r2] | |

* 寻址方式

|  |  |  |
| --- | --- | --- |
|  | AT&T | Intel |
| win32 | section**:**disp(base, index, scale) | section**:**[base + index\*scale + disp] |
| linux | disp + base + index \* scal | disp + base + index \* scal |
| X86  (linux) | movl -4(%ebp), %eax  movl array(, %eax, 4), %eax  movw array(%ebx, %eax, 4), %cx  movb $4, %fs:(%eax) | mov eax, [ebp - 4]  mov eax, [eax\*4 + array]  mov cx, [ebx + 4\*eax + array]  mov fs:eax, 4 |
| ARM | ldrb r0 ,[r2]  strb r0, [r2] | |

### 内嵌汇编

在使用高级语言编写程序的时候，可插入汇编程序。格式如下：

asm(<code> **:** <output operand list> **: <**input operand list> **:** <clobber list>)**;**

* 书写格式

asm关键字表示汇编指令的起始，括号括起来的部分即汇编程序，最后以分号结尾，表示汇编指令的结束。

汇编程序部分总共分为4个域，域之间以冒号分隔，从左到右分别为代码，输出操作数列表，输入操作数列表，修改列表。当某个段后续的段全部为空时，则可以省略该段后面的冒号，如下：

1. 只有代码域

asm("mov r0,r0");

1. 只有修改列表域

asm(""**:::**"memory");

1. 只有代码域和输出域

asm("vmrs %[result], fpscr" **:** "=r" (var)); //FPSCR的值保存到变量var

1. 只有代码域和输入域

asm("vmsr fpscr, %0" **: :** "r" (var)); //将变量var的值保存到FPSCR

* 代码域格式

汇编指令放在双引号之中，多条指令可以使用分号“；”隔开，或者使用换行符隔开。

代码域对寄存器的使用只需给出"样板"和约束条件即可，具体寄存器如何与变量结合由编译器负责。

代码域中表示的寄存器的"样板"操作数有两种方式：

1. 老的方式

使用数字，如%0便是第0个操作数，%1表示第一个操作数，以此类推；

1. 新的方式

使用变量名，如%[result]或者$[value]。

注意：

由于操作数的前缀使用了“％”，因此在用到具体的寄存器时就在前面加两个“％”，以免产生混淆，如%%r0表示寄存器r0。

* 操作数列表域格式

1. 操作列表格式

“[=/+]<约束条件>”(<变量>)

其中等号(=)约束说明<变量>是一个 Write-Only的，加号(+)约束说明<变量>是一个Read-Write的，如果没有给出这两个符号中的任何一个，则说明<变量>是Read-Only的。

因为输入操作数列表其<变量>是可写的，所以其约束条件前需要增加等号(=)或加号(+)。

1. 主要的约束字母及其含义

<约束条件>用于规定<变量>如何与样板操作数进行结合，必要时可以包含多个约束，相互之间用逗号分隔。

|  |  |
| --- | --- |
| 字母 | 含义 |
| m, v,o | 内存单元 |
| r | 任何通用寄存器 |
| q | 寄存器eax, ebx, ecx,edx之一 |
| i, h | 直接操作数 |
| E, F | 浮点数 |
| g | 任意 |
| a, b,c ,d | 要求使用寄存器eax/ax/al, ebx/bx/bl, ecx/cx/cl或edx/dx/dl |
| S, D | 要求使用寄存器esi或edi |
| I | 常数（0～31） |

* 修改列表域格式

1. 寄存器限制

如果在<修改列表>域声明了一个寄存器，那么这个寄存器则不能出现在<操作数列表>中，如果<操作数列表>的约束被指定为"r"或"g"，则编译器在选择寄存器时会忽略<修改列表>中的寄存器。

比如：asm ("movl %0, %%ebx" **: :** "a"(\_\_foo) **:** "ax", "bx");中，由于"a"(\_\_foo)已经指定了%eax寄存器，那么在<修改列表>域中指定"ax"就是非法的。编译时会报错。

1. 数据访问方式限制

<修改列表>域存在"memory"，表示在执行该汇编时，如果某个内存的内容之前已经保存到寄存器，则使用时需要重新读取，而不是使用寄存器中的拷贝。

* 示例

#include <stdio.h>

int add(int i, int j)

{

int res = 0;

\_\_asm (

"ADD %[result], %[input\_i], %[input\_j]"

: [result] "=r" (res)

: [input\_i] "r" (i), [input\_j] "r" (j)

);

return res;

}

int main(void)

{

int a = 1;

int b = 2;

int c = 0;

c = add(a,b);

printf("Result of %d + %d = %d\n", a, b, c);

}

### 不透明类型

opaque data structure(不透明数据类型)隐藏了内部格式或结构。可以利用typedef声明这样一个类型，但是无法预知空间的大小，也不希望开发者把它重新转化成其对应的数据类型。

* linux中的不透明类型

pid\_t，atomic\_t，dev\_t，gid\_t和uid\_t等等。

不透明类型还帮助我们隐藏了类型的长度，该类型也并不总是完整的32位。

* 处理不透明类型的原则

1. 不要假设该类型的长度。
2. 不要将该类型转化回其对应的C标准类型使用。
3. 编程时要保证在该类型实际存储空间和格式发生变化时代码不受影响。
4. 个别编译器不支持opaque类型。

### 编译选项

* 选项说明

<http://gcc.gnu.org/onlinedocs/gcc-3.4.4/gcc/Option-Summary.html#Option-Summary>

* Wl

将Wl后带有逗号的参数分割，传递给linker。

* L<path>

-L<path>表示linker将在path中查找so文件，如果path中没有，然后在默认路径找 (in GNU/Linux, the directories can be listed in /etc/ld.so.conf, and the cache updated by running /etc/ldconfig).

-Wl,-rpath=<path>表示loader在run-time时的so路径，用于区分linker的搜索路径，即编译时和运行时的so搜索路径可以分开指定。

* M

-M表示输出源文件的依赖关系。

1. -M <source files>表示输出source files的依赖关系到标准输出，即所有include的文件(包括include的文件中又使用include包含了其他的文件)。

比如：gcc -c –M foo.c表示输出foo.c的依赖关系。即foo.c中所有include的文件。

1. 依赖关系的描述格式

foo.o:foo.c <include 的文件列表(各文件以空格分隔)>

1. -M –MF <outputfile> <source files>表示将依赖关系输出到outputfiles中。
2. –MD <source files>则表示将依赖关系输出到同名的.d文件。
3. 当gcc使用-o指定输出文件名时，则-MD输出的.d后缀的文件名与-o指定的文件名(除去后缀的部分)相同。

## Makefile

### Make参数

|  |  |
| --- | --- |
| 选项 | 功能 |
| -b/-m | 忽略和其它版本make的兼容性 |
| -B | --always-make,所有的目标都需要更新（重编译） |
| -C <dir> | --directory=<dir>,指定读取makefile的目录.(助记：Change dir)  如果有多个“-C”参数，后面的路径以前面的作为相对路径，并以最后的目录作为被指定目录。  如：“make -C ~ /test -C prog”等价于“make -C ~ /test/prog”。 |
| --debug[=<opt>] | 输出make的调试信息，<options>的取值： a：是verbose，在b选项的级别之上。输出的信息包括哪个makefile被解析，不需要被重编译的依赖文件（或是依赖目标）等。 i：即implicit，输出所有的隐含规则。 j：即jobs，输出执行规则中命令的详细信息，如命令的PID、返回码等。 m：即makefile，输出make读取makefile，更新makefile，执行makefile的信息。 |
| -d | 相当于“--debug=a” |
| -e | --environment-overrides，使用环境变量的定义覆盖makefile中变量定义的值 |
| -f=<file> | --file=<file>或--makefile=<file>，指定需要执行的makefile |
| -h | --help 显示帮助信息 |
| -i | --ignore-errors, 执行时忽略所有的错误 |
| -I <dir> | --include-dir=<dir>,指定一个被包含makefile的搜索目标。可以使用多个“-I”参数来指定多个目录 |
| -j [<COUNT>] | --jobs[=<COUNT>],指同时运行命令的个数。  如果没有这个参数，make运行命令时能运行多少就运行多少。如果有一个以上的“-j”参数，那么仅最后一个“-j”才是有效的. |
| -k | --keep-going出错也不停止运行, 如果生成一个目标失败了，那么依赖于其上的目标就不会被执行了 |
| -l <load> | --load-average[=<load]，--max-load[=<load>]，指定make运行命令的负载 |
| --just-print | --dry-run，--recon，仅输出执行过程中的命令序列，但并不执行 |
| -o <file> | --old-file=<file>，--assume-old=<file>不重新生成的指定的<file>，即使这个目标的依赖文件新于它 |
| -p | --print-data-base，输出makefile中的所有数据（规则和变量） |
| -q | --question不运行命令，也不输出。仅仅是检查所指定的目标是否需要更新。如果是0则说明要更新，如果是2则说明有错误发生 |
| -r | --no-builtin-rules，禁止make使用任何隐含规则 |
| -R | --no-builtin-variabes，禁止make使用任何作用于变量上的隐含规则 |
| -s | --silent，--quiet，在命令运行时不输出命令的输出 |
| -S | --no-keep-going，--stop，取消“-k”选项的作用。因为有些时候，make的选项是从环境变量“MAKEFLAGS”中继承下来的。所以你可以在命令行中使用这个参数来让环境变量中的“-k”选项失效。 |
| -t | --touch，相当于UNIX的touch命令，把目标的修改日期变成最新的，也就是阻止生成目标的命令运行 |
| -v | --version，输出make程序的版本、版权等关于make的信息 |
| -w | --print-directory，输出运行makefile之前和之后的信息 |
| --no-print-directory | 禁止“-w”选项 |
| -W <file> | --what-if=<file>，--new-file=<file>，--assume-file=<file>假定目标<file>需要更新，如果和“-n”选项使用，那么这个参数会输出该目标更新时的运行动作。如果没有“-n”那么就像运行UNIX的“touch”命令一样，使得<file>的修改时间为当前时间。 |
| --warn-undefined-variables | 只要make发现有未定义的变量，那么就输出警告信息 |

### 变量语句

* 变量引用(读取)

使用 “$变量名”的形式引用，实际的“$”书写格式为“$$”。

* 变量赋值(写入/定义)

使用“变量名 = 值”的形式赋值。

|  |  |
| --- | --- |
| 赋值符号 | 功能 |
| **?=** | 变量没有定义的情况下进行赋值 |
| **+=** | 增加值 |
| **=** | 延迟赋值(即变量使用的时候才真正赋值) |
| **:=** | 立即赋值，无延迟 |

* 限定目标/模式变量

格式：

TARGET ... **:** VARIABLE-ASSIGNMENT

PATTERN ... **:** VARIABLE-ASSIGNMENT

或者：

TARGET ... **:** override VARIABLE-ASSIGNMENT

PATTERN ... **:** override VARIABLE-ASSIGNMENT功能：

即变量定义只在特定情况(当构造某目标或符合某模式的目标时)下有效，否则该变量相当于没有定义，或者使用默认普通赋值语句给的值。

目标指定变量( Target-speciﬁc Variable )允许对于相同变量根据目标指定不同的值。目标指定的变量值只在指定它的目标的上下文中有效，对于其他的目标没有影响。也就是说目标指定变量是此目标的"局部"变量。

模式指定变量（Pattern-speciﬁc Variable ）是将一个变量值指定到所有符合此模式的目标上，而不像目标指定变量一样只适用于某一个目标。

示例：

foo**:** rm-dir **:**= test\_dir

表示当构造foo目标时，rm-dirs的值为test\_dir。

%**.**o**:** rm-dir **:**= test\_dir

表示当构造符合%.o模式的目标时，rm-dirs的值为test\_dir。

* 系统内置变量

|  |  |
| --- | --- |
| 变量名 | 含义 |
| VPATH | 用于设置依赖文件的搜索路径 |
| CURDIR | 当前目录，表示Make正在该目录解析Makefile |
| MAKECMDGOALS | 目标列表，表示调用Make命令时传入的所有目标 |
| MAKEFILE\_LIST | 所有make使用到的Makefile，包括从命令行指定以及通过include指示符加入的makefile，LIST中文件名顺序即添加顺序 |
| MAKEFILES | 环境变量，存储make所有参数，变量信息，以及上层Makefile中定义的变量，环境变量会自动传递到下层Makefile中 |
| $$$$ | 当前make的进程号 |

* 系统自动变量(只读)

|  |  |
| --- | --- |
| 变量名 | 含义 |
| $@ | 目标名(助记：按键2的上标，即to)  $(@F)目标中的文件，$(@D)目标中的目录 |
| $\* | 目标文件名去除后缀 |
| $% | 目标中的归档成员名(助记：按键5的上标，即five=file)  比如目标为A.so(B.o)，则$@=A.so，$%=B.o |
| $+ | 所有的依赖文件名,未消除重复(助记：简单累加)  文件名顺序按照出现在makefile中的顺序 |
| $^ | 等价于$+,消除重复(助记：^为C的异或符号，即相同的为0) |
| $? | 等价于$+，且比目标文件新(助记：依赖比目标新，必有question问题) |
| $< | 第一个依赖文件名(助记：左边的箭头表示最左边) |

### 函数语句

* 函数定义格式

define 函数名

函数体

endef

* 参数

第一个参数$(1)，第二个$(2)，依次类推。

* 调用方式

$(call, 函数名, 参数列表)

* 变量扩展

当函数体中有对变量进行引用时，需要使用$$符号替换$。

### Include与sinclude

* include在查找文件失败后会报错；
* sinclude 在查找失败后会直接忽略，s指silence；
* sinclude 与 –include 等价。

### 目标定义

* 普通目标

格式：<targets> **: <**normal-prerequisites>

功能：依赖更新后执行构造命令。

* 可重复的目标

格式：<targets> **:: <**prerequisites>

功能：使用“::”(双冒号)代替普通规则的“:”得到的规则，允许在多个规则中为同一个目标指定不同的重建目标的命令。

1. 对于没有依赖的目标，如果定义为可重复类型则构造命令将会被无条件执行，如果定义为普通类型，只要目标存在(无论是否更新)，则不会执行。
2. 多个相同的目标不能以普通类型的方式定义，否则make会报语法错误。
3. 可重复类型的目标只要任何一个目标因为依赖更新执行了构造命令，则其后面重复的目标将不会被执行。

* 静态模式规则

格式：<targets> **:** <target-pattern> **: <**prerequisites-patterns>

功能：在<targets>中查找符合<target-pattern>的目标，如果**<**prerequisites-patterns>  
 对应的依赖有更新则执行构造命令。

1. 目标和依赖的pattern只能出现1次通配符“%”。
2. **<**prerequisites-patterns>的“%”和<target-pattern>中的“%”表示的字符串对应，该字符串替换**<**prerequisites-patterns>的“%”后即目标的依赖条件。

* 有序依赖

格式：<target> **: <**normal-prerequisites> **| <**order-only-prerequisites>

功能：order-only类型的依赖使用“|”开始，其功能如下：

1. 当<target>存在时，更新order-only类型的依赖后不会引起构造命令执行。
2. 当<target>不存在时，order-only类型的依赖等价于normal类型依赖。

* 库依赖

格式：<target> **: -**l <**name**>

功能：表示<target>依赖lib**name**.a文件。

### 特殊目标

* 伪目标

格式：**.**PHONY**: <** targets>

功能：表示**<**targets>总是更新的，必须无条件构造。当有多个伪目标时，则默认make时只构造最早定义的目标。和在.PHONY中声明顺序无关。

* 错误处理

1. 删除构造规则命令错误的目标

格式：**.**DELETE\_ON\_ERROR

功能：如果规则的命令执行错误，将删除已经被修改的目标文件。

1. 忽略错误

格式：**.**IGNORE**: <** targets>

功能：忽略构造**<** targets>所执行命令的错误。

### 隐含规则

* 后缀规则

后缀规则有两种方式：”双后缀”和”单后缀”。

1. 后缀必需为make可识别的。可使用伪目标”.SUFFIXES”来定义或删除一些特定的后缀。make的参数”-r”或”-no-builtin-rules”也会清空默认认的后缀列表。
2. 双后缀规则定义了一对后缀：目标文件的后缀和依赖目标（源文件）的后缀。如”.c.o”相当于”%o : %c”。
3. 单后缀规则只定义一个后缀，也就是源文件的后缀。如”.c”相当于”% : %.c”。
4. 后缀规则不允许任何的依赖文件，如果有依赖文件的话，那就不是后缀规则，那些后缀统统被认为是文件名，如：.c.o: foo.h表示文件”.c.o”依赖于文件”foo.h”，并不表示.o依赖于.c和foo.h。

* 库文件的成员

表示一个库文件目标由多个文件组成。格式如下：

archive(member)

如：foolib(hack.o) : hack.o表示hack是foolib的成员，可用$%来引用member。

### 文本替换

文本指以“空格”、“Tab”或“回车”、“换行”分隔的字符串(单词)。

* $(subst <from>,<to>,<text>)

功能：substitude，把文本<text>中的<from>字符串替换成<to>

示例：

$(subst ee,EE,feet on the street)

将“feet on the street”中的“ee”替换成“EE”。

* $(patsubst <pattern>,<replacement>,<text>)
* $(text**:**<pattern>=<replacement>)

功能：把<text>中符合<pattern>模式的字符串用<replacement>替换。

patten substitude，如果<replacement>中包含通配符“%”(表示任意字符串)，则其与<pattern>中“%”所代表示的字符串对应。如果只是实际的字符“%”，则需要转义表示为“\%”。

注意：

模式匹配是以单词为单位，如果字符串中无分隔符，则视为一个单词。比如<%foo>可以匹配到<abc**foo**>但是不能匹配到<abc**foo**def>。

示例：

objects = foo.o bar.o baz.o

$(patsubst %.o,%.c, $(objects) )   或 $(objects**:**.o=.c)

将\*.o替换为\*.c，返回foo.c bar.c baz.c。

### 文本查找

* word

功能：取字符串中取某个位置的单词。  
示例：$(word 2, foo bar baz)返回“bar”。

* wordlist

功能：从字符串中取从开始到结束位置的单词串。

示例： $(wordlist 2, 3, foo bar baz)返回值是“bar baz”。

* words

功能：统计中字符串中的单词个数。

示例：$(words, foo bar baz)返回值是“3”。

* firstword

功能：取字符串中的第一个单词。

示例：$(firstword foo bar)返回值是“foo”。

备注：这个函数可以用word函数来实现：$(word 1,)。

### 文本过滤

* strip

功能：去掉字串中开头和结尾的空字符。

示例：

$(strip a b c )

把字串“a b c ”去掉开头和结尾的空格，结果是“a b c”。

* filter/filter-out

功能：filter保留符合模式的单词，filter-out去除符合模式的单词，多个模式用  
 空格分开。

示例：

sources := A.c B.c C.s D.h

则$(filter %.c %.s,$(sources))返回 “A.c B.c C.s”。

则$(filter-out %.c %.s,$(sources))返回 “D.h”。

### 运行控制

* $(error text...)

Generates a fatal error where the message is text.

* $(warning text...)

Text is expanded and the resulting message is displayed, but processing of the makefile continues

* $(info text...)

Does nothing more than print its (expanded) argument(s) to standard output.

### 文件名操作

* wildcard

功能：返回所有符合其参数描述的有效文件路径，文件间以空格间隔。

示例：

$(wildcard \*.c)

返回结果是一个所有以 '.c' 结尾的文件的列表。

* dir/notdir

功能：从文件名序列中取出目录/文件部分。

目录部分是指最后一个反斜杠（“/”）之前的部分。如果没有反斜杠，那么返回“./”。

示例：

$(dir src/foo.c hacks)返回 “src/ ./”

$(notdir src/foo.c hacks)返回 “foo.c hacks”

* suffix/ basename

功能：从文件名列表中取出各个文件名的后缀/前缀。

如果文件没有后缀/前缀，则返回空字串。

示例：

$(suffix src/foo.c src-1.0/bar.c hacks)返回“.c .c”。

$(basename src/foo.c src-1.0/bar.c hacks)返回“src/foo src-1.0/bar hacks”。

* addsuffix/ addprefix

功能：把后缀加到中的每个单词后面/把前缀加到中的每个单词前面。

示例：

$(addsuffix .c,foo bar)返回“foo.c bar.c”。

$(addprefix src/,foo bar)返回“src/foo src/bar”。

* join

功能：把两个单词组中的单词分别连接，组成一个新的单词组。

如果某一组的单词个数多，则多出来的单词直接放入新的单词组中。

示例：

$(join aaa bbb , 111 222 333)返回“aaa111 bbb222 333”。

### 变量处理

* origin

功能：获取变量的定义来源。

语法：$(origin <variable>)

|  |  |
| --- | --- |
| 返回值 | 功能 |
| "undefined" | 未定义 |
| "enviroment" | 定义在环境变量 |
| "default" | 默认的变量定义 |
| "file" | 定义在 Makefile 文件中 |
| "command line" | 来自命令行 |

* eval

功能：将text 的内容展开(本身并不解析，只是交给makefile进行解析执行)。

注意：存在变量引用时，需要使用“$($(val))”替换“$(val)”(类似函数定义)。

语法：$( eval <text>)

## CPP

### 运算符优先级

* 优先级表

|  |  |  |  |
| --- | --- | --- | --- |
| Precedence | Operator | Description | Associativity |
| 1 | :: | Scope resolution | Left-to-right |
| 2 | ++ -- | Suffix/postfix increment and decrement |
| () | Function call |
| [] | Array subscripting |
| . | Element selection by reference |
| -> | Element selection through pointer |
| typeid() | Run-time type information (see [typeid](http://en.cppreference.com/w/cpp/language/typeid)) |
| const\_cast | Type cast (see [const\_cast](http://en.cppreference.com/w/cpp/language/const_cast)) |
| dynamic\_cast | Type cast (see [dynamic\_cast](http://en.cppreference.com/w/cpp/language/dynamic_cast)) |
| reinterpret\_cast | Type cast (see [reinterpret\_cast](http://en.cppreference.com/w/cpp/language/reinterpret_cast)) |
| static\_cast | Type cast (see [static\_cast](http://en.cppreference.com/w/cpp/language/static_cast)) |
| 3 | ++ -- | Prefix increment and decrement | Right-to-left |
| + - | Unary plus and minus |
| ! ~ | Logical NOT and bitwise NOT |
| (type) | Type cast |
| \* | Indirection (dereference) |
| & | Address-of |
| sizeof | Size-of |
| new, new[] | Dynamic memory allocation |
| delete, delete[] | Dynamic memory deallocation |
| 4 | .\* ->\* | Pointer to member | Left-to-right |
| 5 | \* / % | Multiplication, division, and remainder/modulus[[1]](http://en.cppreference.com/w/cpp/language/operator_precedence#cite_note-0) |
| 6 | + - | Addition and subtraction |
| 7 | << >> | Bitwise left shift and right shift |
| 8 | < <= | For relational operators < and ≤ respectively |
| > >= | For relational operators > and ≥ respectively |
| 9 | == != | For relational = and ≠ respectively |
| 10 | & | Bitwise AND |
| 11 | ^ | Bitwise XOR (exclusive or) |
| 12 | | | Bitwise OR (inclusive or) |
| 13 | && | Logical AND |
| 14 | || | Logical OR |
| 15 | ?: | Ternary conditional | Right-to-Left |
| 16 | = | Direct assignment (provided by default for C++ classes) |
| += -= | Assignment by sum and difference |
| \*= /= %= | Assignment by product, quotient, and remainder |
| <<= >>= | Assignment by bitwise left shift and right shift |
| &= ^= |= | Assignment by bitwise AND, XOR, and OR |
| 17 | throw | Throw operator (exceptions throwing) |
| 18 | , | Comma | Left-to-right |

* 自增减运算

1. (++i)+(i++)+(++i)

假设i=3，则运算顺序为：i增加2次(5)🡪相加结果为13🡪i增加1次(6)

* 逗号运算符

(a,b)返回b

* 括号运算符

### 数据对齐

1. 大小端

以字节(8bit)为单位的数据排列方式，单个字节的bit无大小端区分。多个字节的大小端用地地址保存的对应数值的高位还是低位决定，如下：

1. 大端模式

数据的高位，保存在内存的低地址中，而数据的低位，保存在内存的高地址中，这样的存储模式有点儿类似于把数据当作字符串顺序处理：地址由小向大增加，而数据从高位往低位放；

1. 小端模式

数据的高位保存在内存的高[地址](http://baike.baidu.com/view/494802.htm)中，而数据的低位保存在内存的低地址中，这种存储模式将地址的高低和[数据](http://baike.baidu.com/view/38752.htm)位权有效地结合起来，高地址部分权值高，低地址部分权值低，和我们的逻辑方法一致。

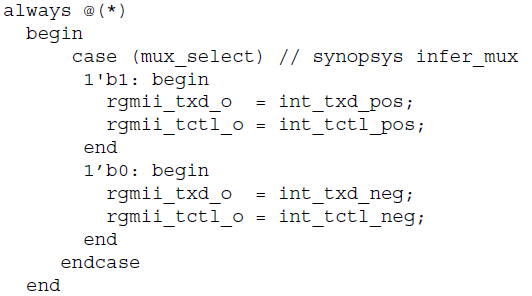
## verilog

### 设计流程

首先设计各个模块，包括寄存器组、控制器、RAM、ROM、IO等等。然后根据不同的情况把他们连接起来，也就是建立各个指令的数据通路，然后接入时钟源。再在ROM里写入程序就可以了。

### 分支逻辑

1. always
2. always@后面内容是敏感变量，always@(\*)里面的敏感变量为\*，意思是说敏感变量由综合器根据always里面的输入变量自动添加，不用自己考虑。
3. 如果没有@，那就是不会满足特定条件才执行，而是执行完一次后立马执行下一次，一直重复执行，比如testbench里面产生50Mhz的时钟就(假设时间尺度是1ns）可以写成 always #20 CLK\_50Mhz = ～CLK\_50Mhz;



## Shell

### 函数定义

* 函数类型

1. 内建命令，shell内部定义的函数，比如cd,type,source,declare,pwd,export等。
2. 外部命令，外部执行文件。
3. 自定义函数，用户自定义。

* 自定义命令语法

[ function ] <funname > [()]

{

action;

[return int;]

}

说明:

1. function 和()关键字可以不写。
2. 参数返回，可以显示加：return 返回，如果不加，将以最后一条命令运行结果，作为返回值。 return后跟数值n(0-255)

* 函数调用

内建函数，外部命令，自定义函数的调用方法相同，如下：

<funname> [<args>]

* 函数调用优先级

1. Aliases (使用内建命令alia定义的命令)
2. shell关键字，例如function，if，for
3. Functions
4. 内建命令(Built-ins)，例如cd，type 。
5. 外部命令(在命令搜索路径PATH下的脚本和可执行文件)。

* 函数返回值，只能通过$?

### 函数参数

* 参数引用

$1是第一个，$2是第二个，依次类推。

* 位置移动

参数位置可以用shift命令左移。比如shift 3表示原来的$4现在变成$1，原来的$5现在变成$2等等，原来的$1、$2、$3丢弃，$0不移动。

不带参数的shift命令相当于shift 1。

* 自动补全

complete -F <参数补全方法> <函数名>

### 运行环境

当脚本直接在命令行所在shell执行时(或者使用sh xxx.sh执行)，则会启动一个子shell来执行命令，子shell的运行环境和命令行所在shell环境不一样。传递环境的方法：

* 变量传递

定义变量的语句和执行脚本的语句写在一行时，则变量定义语句与命令执行环境相同，相当于变量定义传递给脚本。

* 环境传递

使用内建命令source (该命令通常用命令“.”来替代)执行脚本，则执行脚本的内容导入到当前运行的shell环境中，即相当于使用source命令的脚本内容在调用者的脚本中。

* 用户环境传递

当在shell中切换用户时，在su命令后增加“-”可将当前环境传递。

* 环境查看/更新

使用内建命令declare不带参数可以查看当前shell 环境的内容。declare需要在是bash版本2之后使用。在在ksh脚本中则使用typeset。

|  |  |
| --- | --- |
| 参数 | 含义 |
| -r | 只读(与readonly作用相同) |
| -i | 整数 |
| -a | 数组 |
| -f | 函数 |
| -x | export |

### 命令控制

* && 左边的命令返回真（返回值==0），右边的命令才会被执行。
* **||** 左边的命令返回假（返回值!=0），右边的命令才会被执行。

### 变量定义

* 函数内部的变量需要使用local 声明
* 定义方法

|  |  |
| --- | --- |
| 定义方法 | 注意 |
| = | =号两边不能有空格 |
| read | 暂停脚本的执行并等待键盘的输入，当用户输入完毕并且敲下回车之后，将完成赋值操作，脚本继续执行。 |
| 多个数据或变量之间用空格隔开 |
| 若变量个数与数据个数相等时，对应取值 |
| 若变量个数大于数据个数时，没有输入数据的变量取空值 |
|  | 若变量个数小于数据个数时，将多余的数据赋给最后一个变量（也就是把空格当成字符串处理） |

* 特殊变量

|  |  |
| --- | --- |
| 特殊变量 | 含义 |
| $$ | Shell本身的PID（ProcessID） |
| $! | Shell最后运行的后台Process的PID |
| $0 | Shell本身的文件名 |
| $1~$n | 添加到Shell的各参数值。$1是第1参数、$2是第2参数…。 |
| $? | 最后运行的命令的结束代码（返回值） |
| $- | 使用Set命令设定的Flag一览 |
| $\* | 参数列表。用doublequote引用"$\*"，输出"$1 $2 ...$n" |
| $@ | 所有参数。用doublequote引用"$@",输出"$1" "$2" ... "$n" |
| $# | 添加到Shell的参数个数 |

* 环境变量

|  |  |
| --- | --- |
| 环境变量 | 含义 |
| $HOME | 当前用户主目录 |
| $HOSTNAME | 用户所在主机的名称 |
| $LOGNAME | 当前用户的登录名 |
| $PATH | 以冒号分隔的路径,用于shell搜索命令或程序 |
| $PWD | 当前路径 |
| $OLDPWD | 上次cd前的路径 |
| $PS1 | 命令行提示符，root用户是“#”，对于普通用户是“$” |
| $PS2 | 辅助提示符，用来提示后续输入，通常是“>”字符 |
| $IFS | 输入区单词间的分隔符。通常是空格,制表符和换行符，比如：ls命令在输出时使用IFS作为分隔符 |
| $PROMPT\_COMMAND | 执行命令时shell首先自动调用的命令，一般用于日志跟踪 |
| $SECONDS | shell被invoke的时间间隔 |
| $BASH | 当前运行的bash的路径，例如:/bin/bash |
| $BASH\_VERSION | shell的版本，例如:3.2.49(22)-release |
| $BASH\_VERSINFO | 当前使用shell的主版本信息。例如:3 |
| $SHELL | login时所使用的shell路径，一般和BASH一样 |
| $TERM | 当前teminal的类型，例如:cygwin |
| $EDITOR | text editor的路径 |
| $HISTSIZE | 历史记录数 |
| $LANGUGE | 语言相关的环境变量，多语言可以修改此环境变量 |
| $MAIL | 当前用户的邮件存放目录 |
| $LINES | 终端显示的行数 |
| $COLUMNS | 终端显示的列数 |

* 变量操作

1. 查看所有环境变量：env
2. 查看指定变量：echo $变量名
3. 查看/设置本地shell变量：set
4. 删除shell变量：unset变量名
5. 设置变量为只读：readonly变量名

* 修改变量的方法

1. 使用export命令

该方法变量在关闭shell时失效。

比如在PATH中临时加入一个目录，则直接在命令行输入：

export PATH=$PATH:/usr/local/arm/4.3.2/bin

1. 修改默认配置文件

将export命令添加到配置文件。

如果变量用于所有用户，则定义在/etc/profile文件；

如果用于某个用户，则定义在对应用户的$HOME/.bash\_profile文件(On some systems this file is called ".profile".)

1. 导入自定义配置文件

从文件导入变量：source <bashrc文件>。

### 变量引用

* 直接引用

引用变量${var}或者$var

* 间接引用

引用变量的变量，\$$var 或者${!var}

* 函数结果引用

$(fn)

### 数组变量

* 声明

declare –a arr

* 赋值

1. 逐项赋值

arr[0]=BeiJing

arr[1]=GuangZhou

arr[2]=ShenZhen

1. 列表

arr=([0]=BeiJing [1]=GuangZhou [2]=ShenZhen)

1. 连续赋值

arr=(BeiJing GuangZhou ShenZhen)

* 引用

${arr[0]}

* 长度

${#arr[@]}

* 遍历

1. ${arr[\*]}，与$\*格式相同
2. ${arr[@]}，与$@格式相同

### 字符串变量

* 字符串长度

${#VALUE}：计算VALUE字符串的字符数量。

* 字符串匹配

“%”表示从右向左匹配，“#”表示从左向右匹配，“/”表示替换，属于非贪婪匹配，即匹配符合通配符的最短结果。

“%%”、“##”和“//”，属于贪婪匹配，即匹配符合通配符的最长结果。(助记：键盘上“#”和“%”分别位于“$”的左边和右边)

1. ${VALUE%KEY}或${VALUE%%KEY}

删除VALUE字符串中匹配KEY模式的左边字符，保留右边字符，其中KEY字符串可以使用通配符“\*”。

1. ${VALUE#KEY}或${VALUE##KEY}

删除VALUE字符串中匹配 KEY模式的右边字符，保留左边字符，其中KEY字符串可以使用通配符“\*”。

1. ${VALUE/OLD/NEW}或${VALUE//OLD/NEW}

用NEW子串替换VALUE字符串中匹配的OLD子串。

* 字符串截取

截取位置左边第一个字符从“0”开始，右边第一个字符从“0-1”开始。 表示偏移OFFSET个字符开始，LENGTH表示要截取字符的长度。如果没有LENGTH变量，表示偏移OFFSET个字符开始到字符串结束。

1. ${VALUE:OFFSET}或${VALUE:OFFSET:LENGTH}

从VALUE字符串的左边开始中截取子串。

1. ${VALUE:0-OFFSET}或${VALUE:0-OFFSET:LENGTH}

从VALUE字符串的右边开始中截取子串。

### 循环语句

* while

1. 通用条件

min=1

max=100

while [ $min -le $max ]

do

echo $min

min=`expr $min + 1`

done

1. 数学条件

i=1

while(($i<100))

do

if(($i%4==0))

then

echo $i

fi

i=$(($i+1))

done

* for

1. 集合遍历

for <x> in <集合>

do

处理$<x>语句

done

<集合>可以为：

`cmd` 🡪命令的输出结果；

{m..n} 🡪数字段；

x y z 🡪列表；

$(<file)🡪文件以行为单位的内容；

1. 条件遍历

for((i=1;i<100;i++))

do

处理$<i>语句

done

* 循环控制

1. break退出循环

不执行当前循环体内break下面的语句从当前循环退出.

1. continue继续循环

在本循体内忽略下面的语句,从循环头开始执行

### 条件语句

* 语句格式

if 条件

then

处理语句

else

处理语句

fi

* 条件类型

|  |  |  |  |
| --- | --- | --- | --- |
| 条件类型 | 格式 | 真 | 假 |
| 数学表达式/命令返回/函数返回 | xxx | 0 | 非0 |
| 字符串 | “xxx” | 非空字符串 | 空字符串 |

* 条件格式

|  |  |
| --- | --- |
| 条件格式 | 含义 |
| if command/function then | 将命令/函数返回值作为条件，如果为多个命令，则等价于所有返回值 and后的结果，  等价于 command执行后执行if $? |
| if [ expression ] then | 数学表达式或字符串作为条件，中括号必须和表达式之间有空格 |
| if test expression  then | 非真 |
| [ expression ] && command | 如果左边的表达式为真则执行右边的语句 |
| [ expression ] || command | 如果左边的表达式为假则执行右边的语句 |

* 复合条件

逻辑非 ! if [ ! 表达式 ]

逻辑与 -a if [ 表达式1 –a 表达式2 ]

逻辑或 -o if [ 表达式1 –o 表达式2 ]

* 条件表达式

|  |  |
| --- | --- |
| 字符串比较 | 含义 |
| $a = $b ] | 等于（=号两边必须有空格，否则为赋值语句） |
| $string1 != $string2 b | 不等于 |
| -n $stringb | 是否非空 |
| -z $string b | 是否为空 |
| $sting b | 是否非空 |
| 文件表达式 | 含义 |
| -e file | file存在 |
| -d file | file为目录 |
| -f file | file为常规文件 |
| -L file | file为符号链接 |
| -r file | file可读 |
| -w file | file可写 |
| -x file | file可执行 |
| -s file | 如果文件长度不为0 |
| -h file | 如果文件是软链接 |
| file1 -nt file2 | file1比 file2新 |
| file1 -ot file2 | file1比 file2旧 |
| 整数表达式 | 含义 |
| a -eq b b | a等于b |
| a -ne b | 不等于 |
| a -ge b | >= |
| a -gt b | > |
| a -le b | <= |
| a -lt b | < |

* 条件变量

1. ${VALUE:-WORD}

当变量未定义或者值为空时，返回值为WORD的内容，否则返回变量的值。

1. ${VALUE:=WORD}

当变量未定义或者值为空时，返回WORD的值的同时并将WORD赋值给VALUE，否则返回变量的值。

1. ${VALUE:+WORD}

当变量已赋值时，其值才用WORD替换，否则不进行任何替换。

1. ${VALUE:? WORD }

当变量已赋值时，正常替换。否则将消息WORD送到标准错误输出（若此替换出现在SHELL程序中，那么该程序将终止运行）。

### 分支语句

* 格式

**case** $变量名 in

模式1）

命令序列1

  ;;

模式2）

命令序列2

;;

\*）

默认执行的命令序列

;;

**esac**

其中case语句结构特点如下：

1. case行尾必须为单词“in”，每一个模式必须以右括号“）”结束。
2. 双分号“;;”表示命令序列结束。
3. 匹配模式中可是使用方括号表示一个连续的范围，如[0-9]；使用竖杠符号“|”表示或。
4. 最后的“\*）”表示默认模式，当使用前面的各种模式均无法匹配该变量时，将执行“\*）”后的命令序列。

### 内置函数

|  |  |
| --- | --- |
| 函数名 | 功能 |
| read | 获取安按键输入 |
| echo | 输出 |
| printf | 格式化输出 |

## Python

### super限制

* super只能用于新类中。

1. 新类： 有继承类，派生于object类。
2. 经典类：没有继承任何类。

### 全局变量

* 函数中使用全局变量必须用 global声明；

## TCL

### 特点

Tcl 是“工具控制语言（Tool Control Language）”的缩写。Tk 是 Tcl“图形工具箱”的扩展，它提供各种标准的 GUI 接口项，以利于迅速进行高级应用程序开发。

John K. Ousterhout于 1988 年开始开发 Tcl/Tk（读作“tickle tee-kay”），然后是加州大学伯克利分校（UCB）的一名教授继续对它进行开发。Tcl 是以可扩展性、短的学习曲线和易于嵌入为特定目标而设计的。Tk 的开发始于 1989 年，第一个版本于 1991 年问世。Ousterhout 博士在他离开 UCB 之后继续开发 Tcl/Tk，然后由于工作需要，他继续为 Sun Microsystems 公司工作。现在，也就是在写本文之时，他在 Scriptics（它开发出 Ajuba Solution，已由 Interwoven 收购）继续改进该语言，目前的稳定版本是 8.5.13，在写本文的时候，8.6b3 版本正在开发之中。

* Tcl的特性包括：

1. 任何东西都是一条命令，包括语法结构（for,if等）。
2. 任何事物都可以重新定义和重载。
3. 所有的数据类型都可以看作字符串。
4. 语法规则相当简单。
5. 提供事件驱动给Socket和文件。基于时间或者用户定义的事件也可以。
6. 动态的域定义。
7. 很容易用C, C++，或者Java扩展。
8. 解释语言，代码能够动态的改变。
9. 完全的Unicode支持。
10. 平台无关。
11. 和GUI紧密集成。Tk
12. 代码紧凑，易于维护。

* 解析器

1. tclsh

命令行shell，解析TCL 语言。

1. wish

类似于tclsh，它是针对窗口化的 GUI 环境。

# Debug

## ICE

### OpenOCD

* 相关资料

1. 主页:<http://openocd.berlios.de/web/>
2. 论坛:http://forum.sparkfun.com/viewforum.php?f=18
3. doc/openocd.pdf文档(需要编译生成)，里面有各个配置和命令的详细说明。

目前支持多种芯片，可以简单查看src/target/target.c中的target\_types的定义就可以了解支持哪些处理器了.

支持的编程工具，主要是并口的JTAG工具和基于FT2232的JTAG工具。当然，由于源代码都是公开的，并且可以自己编译，所以增加自己定义的工具的驱动也是相当容易的。

* 配置流程

OpenOCD需要针对不同的JTAG工具和不同的目标芯片，创建一个配置文件。可以使用现成的一些配置，一般在使用的JTAG工具的主页上可以找到。

一般而言，配置文件分为4个部分

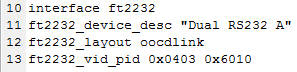
1. 定义各个接口的Port(比如telnet\_port，gdb\_port，tcl\_port)

接口Port的定义一般如下(非特殊情况，无需修改，定义了一些端口号)



1. JTAG工具的定义

JTAG工具的定义，一般在工具的主页里会提供，而且针对不同的目前芯片，这个定义不需要修改。



1. 目标芯片的定义

目标芯片的定义，是用来告诉OpenOCD，JTAG链是如何连接的(以什么顺序，链接了那些目标芯片)。OpenOCD支持的芯片的定义，基本在src/target/target目录中都可以找到对应的配置文件。

**adapter\_khz** 3000

**jtag** newtap <tap\_name> cpu -irlen 4 -ircapture 0x1 -irmask 0xf -expected-id 0x0

**target** create <cpu\_name> cortex\_a8 -chain-position <cpu\_name> -coreid <core\_id> -dbgbase <address>

**reset\_config** trst\_only

* 使用方法

1. 启动OpenOCD

调用openocd可执行文件即可启动OpenOCD，该程序会自动寻找默认配置文件openocd.cfg，或者使用-f参数来指定使用的配置文件：

# openocd -f <配置脚本>

使用OpenOCD配置文件的脚本配置部分，以脚本的方式来实现自动的固件下载。

如果需要使用GDB或者手动操作目标设备的话，在配置文件的脚本部分中，只需要初始化目标芯片的脚本即可。然后可以通过telnet或者gdb或者tcl来连接上OpenOCD。

1. 连接目标芯片

**使用telnet连接：**

#telnet localhost <telnet端口号>

**使用gdb连接：**

启动gdb

#arm-linux-gdb <elf文件>

进入gdb命令行配置remote

>target remote 127.0.0.1:<gdb端口>

**使用tclsh连接：**

#tclsh

1. 执行openocd命令

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 命令 | | 含义 | | | | |
| Target state handling commands | | | | | | |
| **poll** | | 查询目标板当前状态 | | | | |
| **halt** | | 中断目标板的运行 | | | | |
| **resume** [address] | | 恢复目标板的运行，[从address处开始运行] | | | | |
| **step** [address] | | 单步执行，[从address处开始执行一条指令] | | | | |
| **reset** | | 复位目标板 | | | | |
| **exit** | | 退出调试 | | | | |
| Breakpoint commands | | | | | | |
| **bp** <addr> <length> [hw] | | 在地址addr处设置断点，指令长度为length，hw 表示硬件断点 | | | | |
| **rbp** <addr> | | 删除地址addr处的断点 | | | | |
| Memory access commands | | | | | | |
| **mdw[**phys**]** <addr> [count] | 显示addr起始count (默认是1)个数据(大小以word为单位),  默认为cpu的地址，如果使用phys则为mem地址. | | | | | |
| **mdh** **[**phys**]** <addr> [count] | 同mdw，单位为half word. | | | | | |
| **mdb** **[**phys**]** <addr> [count] | 同mdw，单位为byte. | | | | | |
| **mww [**phys**]** <addr> <value> | 向地址addr写入一个word，值为value | | | | | |
| **mwh** **[**phys**]** <addr> < value > | 同mww，单位为half word | | | | | |
| **mwb** **[**phys**]** <addr> < value > | 同mww，单位为byte | | | | | |
| **virt2phys** <addr> | translate a virtual address into a physical address | | | | | |
| **reg** [reg] [val] | display or set a register | | | | | |
| Image file commands | | | | | | |
| **load\_image** <file> <address> [format] | | | | 将<file>载入address起始的内存，format有'bin'、'ihex'、'elf’ | | |
| **dump\_image** <file> <address> <size> | | | | 将address开始的内存读出size字节，保存到<file>中 | | |
| **verify\_image** <file> <address> [format] | | | | 将<file>与内存address开始的数据进行比较，format有'bin'、'ihex‘、'elf’ | | |
| Target command group | | | | | | |
| target **current** | | | | Returns the currently selected target | | |
| target **names** | | | |  | | |
| target **types** | | | | Returns the available target types | | |
| Architecture Specific Commands(ARM command group) | | | | | | |
| **arm** core\_state ['arm'|'thumb'] | | | | | display/change ARM core state | |
| **arm** disassemble <addr> [count ['thumb']] | | | | | disassemble instructions | |
| **arm** mcr cpnum op1 CRn op2 CRm value | | | | | write coprocessor register.  比如，关闭MMU：arm mcr 15 0 1 0 0 0。 | |
| **arm** mrc cpnum op1 CRn op2 CRm | | | | | read coprocessor register  比如：arm mrc 15 0 1 0 0，读出cp15协处理器的寄存器1 | |
| ARM Cortex Commands Group | | | | | | |
| cortex\_a8 **smp\_on** | | | | | | Restart smp handling |
| cortex\_a8 **smp\_off** | | | | | | Stop smp handling |
| cortex\_a8 **cache\_info** | | | | | | target caches info |
| cortex\_a8 **dbginit** | | | | | | Initialize core debug |
| AP Specific Commands Group(Novatek668) | | | | | | |
| dap **info** | | | display ap info | | | |
| dap **apid** | | | display ap id | | | |
| dap **memaccess** | | | extra tck for MEM-AP memory bus access | | | |
| fpga\_668.cpu **eventlist** | | |  | | | |
| fpga\_668.cpu **curstate** | | | displays the current state | | | |
| Jtag Info | | | | | | |
| measure\_clk | | | measure the JTAG clk | | | |
| ms | | | Returns ever increasing milliseconds | | | |
| scan\_chain | | | current scan chain configuration | | | |
| Other commands | | | | | | |
| script <file> | | 执行file文件中的命令 | | | | |
| source <file> | | 执行file，并将file载入当前环境 | | | | |
| help | | 查询帮助信息 | | | | |
| server control | | | | | | |
| telnet\_port <num> | | 设置telenet端口号 | | | | |
| gdb\_port <num> | | 设置gdb连接target 的remote端口 | | | | |
| tcl\_port <num> | |  | | | | |
| interface <name> | | Jtag接口类型，比如ft2232,jlink等。 | | | | |
| ft2232\_device\_desc <string> | | Jtag设备描述 | | | | |
| ft2232\_layout <config> | |  | | | | |
| ft2232\_vid\_pid 0x0403 0x6010 | |  | | | | |
| shutdown | | shut the server down | | | | |

* 基本操作

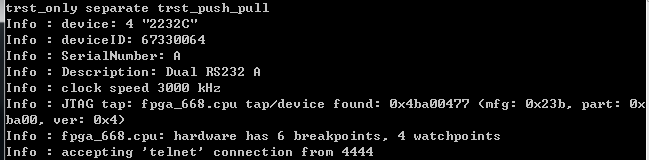
1. 启动ocd服务器

openocd -f <配置脚本>

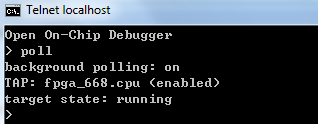


1. 连接ocd服务器

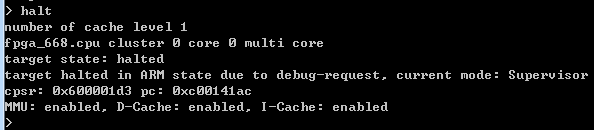
telnet localhost <telnet端口号>

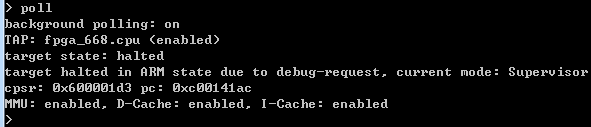


1. poll

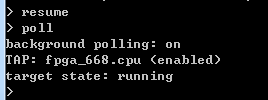


1. halt

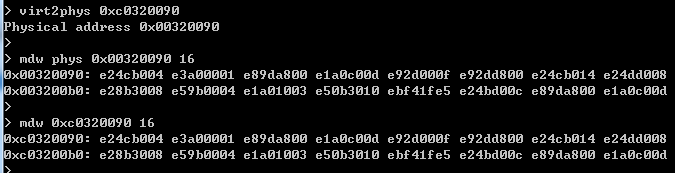




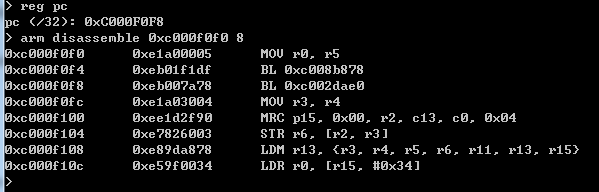
1. resume



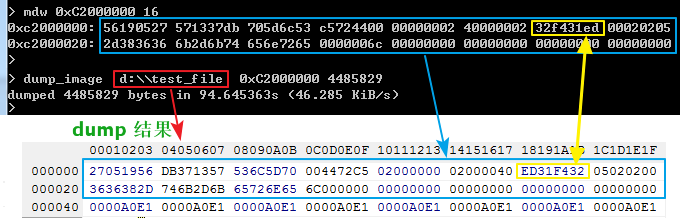
1. 内存显示



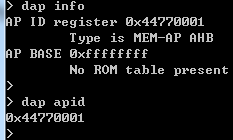
1. 反汇编

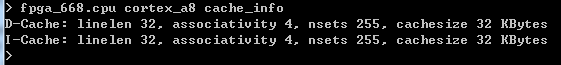


1. 内存dump到文件



1. AP命令





* 脚本的定义

脚本的定义，对于最新版本的OpenOCD尤为重要(最新的OpenOCD去掉了一些配置，而使用脚本的方式来实现)。脚本主要实现了如何操作目标芯片，不同的目标芯片的操作方式会有区别。

1. 脚本定义

脚本采用tcl/tk语法实现，比如：

defaultValue ::PHYBASEm 0xfd630000

defaultValue ::PHYBASEs 0xfd640000

#################################################

proc py { WhichDRAM offset { value 0x100000000 } } {

if { $WhichDRAM == "s" } {

set PHYBASE $::PHYBASEs

} elseif { $WhichDRAM == "m" } {

set PHYBASE $::PHYBASEm

} else {

return "==> Error, not \"m\" nor \"s\" !!"

}

if { $value == 0x100000000 } {

set RealOffset [expr $PHYBASE + ($offset \* 4)]

mdw phys $RealOffset

} else {

set RealOffset [expr $PHYBASE + ($offset \* 4)]

mww phys $RealOffset $value

}

}

########################

proc pym {offset { value 0x100000000 }} {

py m $offset $value

}

########################

proc pys {offset { value 0x100000000 }} {

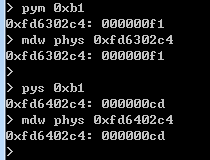
py s $offset $value

}

1. 脚本载入

source xxx.tcl

1. 命令使用



* 示例

1. 下载、运行初始化程序init.bin

halt //暂停

poll //发现单板的MMU或D-cache已经使能

arm920t cp15 2 0 //禁止MMU和D-cache

step //进一步查看是否禁止MMU和D-cache

load\_image init\init.bin 0x40000000

resume 0x40000000 //运行init.bin

1. 下载、运行u-boot

halt

load\_image u-boot\u-boot.bin 0x33f80000

verify\_image u-boot\u-boot.bin 0x33f80000

load\_image u-boot\u-boot.bin 0x30000000

resume 0x33f80000 //运行Uboot

1. 烧写到Nor Flash(uboot命令)

protect off all

erase 0 2ffff

cp.b 30000000 0 30000

1. 烧写到Nand Flash(uboot命令)

nand erase 0 30000

nand write.jffs2 30000000 0 30000

## memory check

### valgrind

Valgrind是一款用于内存调试、内存泄漏检测以及性能分析的软件开发工具。Valgrind这个名字取自北欧神话中英灵殿的入口。

Valgrind的最初作者是Julian Seward，他于2006年由于在开发Valgrind上的工作获得了第二届Google-O'Reilly开源代码奖。

Valgrind遵守GNU通用公共许可证条款，是一款自由软件。

1. 移植
2. 源码下载

valigrind官网: http://valgrind.org/



1. 配置编译环境

修改configure文件, 把armv7\*)改成 armv7\*|arm)

执行

./configure --prefix=$PWD/out --host=arm-linux-gnueabihf

配置成功后显示：

Maximum build arch: arm

Primary build arch: arm

Secondary build arch:

Build OS: linux

Primary build target: ARM\_LINUX

Secondary build target:

Platform variant: vanilla

Primary -DVGPV string: -DVGPV\_arm\_linux\_vanilla=1

Default supp files: exp-sgcheck.supp xfree-3.supp xfree-4.supp glibc-2.X-drd.supp

glibc-2.34567-NPTL-helgrind.supp glibc-2.X.supp

1. 编译

make; make install

编译后生成工具如下：



1. 与mtrace的区别

mtrace也可以用于跟踪 C 的 memory leak, 但不适用于 C++, 原因是 mtrace 通过跟踪 malloc/free而实现, 其可以找出那一行调用malloc后没有调用 free。但是 C++ 的情況是 libstdc++ 调用malloc,需要跟踪 new的调用。

1. 使用方法

使用 valgrind 后被程序运行会慢 20 倍左右。

1. 查找memory leak 和内存读写错误

$ valgrind --tool=memcheck --leak-check=full <PROG> <PROG\_ARG ...>

运行结束后会输出结果到 console。可查看官网文件"Understanding

Valgrind memory leak reports"，了解memcheck 报表格式。重点是 possibly leak 表示 valgrind 无法判断它是 still reachable 或 definitely leak。still reachable 可能是某些连接库内部使用的memory pool, 程序退出时才会释放, 所以严格说起来不是memory leak。在这个阶段, 我们关心的是 definitely leak。

1. 查看profile heap 的用量, 可看到不同时间 heap 在各函数使用的比例

$ valgrind--tool=massif <PROG> <PROG\_ARG ...>

运行结束后会输出结果在 massif.out.PID。然后执行 ms\_print 看结果

$ ms\_print massif.out.PID

1. 获取程序离开前的detailed snapshot

先执行 valgrind；

在另一个 shell 下执行 vgdb detailed\_snapshot </path/to/snapshot>

阅读当下的 snapshot: ms\_print </path/to/snapshot>

## gdb

### 常用命令

|  |  |  |  |
| --- | --- | --- | --- |
| 命令 | 含义 | | |
| General Commands | | | |
| help [command] | 列出帮助信息，或是列出某个命令的帮助信息 | | |
| quit | 退出gdb | | |
| target remote ip:port | 远程连接 | | |
| file <FILE> | 载入文件FILE(不会下载到目标板上) | | |
| load [FILE] | 下载文件到目标板上，默认文件为指定过的(比如file 命令指定的，或是gdb 运行时指定的文件) | | |
| monitor <command …> | 调用gdb 服务器软件的命令，比如使用openond时：  “monitor mdw 0x0”调用openocd的命令“mdw 0x0”。  “monitor halt” 停止开发板上其它可能运行的程序。  “monitor arm920t cp15 2 0” 禁止MMU | | |
| 查看源码 | | | |
| list <FUNCTION> | | 列出某个函数FUNCTION | |
| list <LINENUM> | | 显示源文件的某行LINENUM附件的源码 | |
| list | | 接着前一次继续显示 | |
| list - | | 显示前一次之前的源码 | |
| break \*<address> | | 在某个地址上设置断点，比如 break \*0x84 | |
| list <FILENAME:FUNCTION>  或list <FILENAME:LINENUM> | | 显示指定文件的一段程序 | |
| info source | | 查看当前源程序 | |
| info stack | | 查看堆栈信息 | |
| info args | | 查看当前的参数 | |
| 执行程序 | | | |
| step | 单步执行，会跟踪进入一个函数 | | |
| next | 单步执行，指令则不会进入函数，next简写为n | | |
| nexti | 单步执行，执行一条汇编指令 | | |
| continue | 继续执行程序，加载程序后也可以用来启动程序 | | |
| 断点操作 | | | |
| break <FUNCTION> | 在函数入口设置断点，break简写为b | | |
| break <LINENUM> | 在当前源文件的某一行上设置断点 | | |
| break <FILENAME:LINENUM> | 在指定源文件的某一行上设置断点 | | |
| info br | 查看断点 | | |
| delete <number> | 删除断点 | | |
| diable <number> | 禁止断点 | | |
| enable <number> | 使能断点 | | |
| watch <EXPRESSION> | 当指定变量被写时，程序被停止 | | |
| rwatch <EXPRESSION> | 当指定变量被读时，程序被停止 | | |
| 数据操作 | | | |
| print < EXPRESSION > | 查看数据 | | |
| set varible=value | 设置变量 | | |
| x /NFU ADDR | 检查内存值。  比如“x /4ub 0x0”将会显示0 地址开始到4 个字节 | | |
| N | | 代表数据大小 |
| F | | 代表输出格式：  x ：16 进制整数格式  d ：有符号十进制整数格式  u ：无符号十进制整数格式  f ：浮点数格式 |
| U | | 代表输出格式：  b ：字节(byte)  h ：双字节数值  w ：四字节数值  g ：八字节数值 |

## Terminal

terminal一般指用于连接服务器的客户端软件，其将服务器远程传输的显示内容在客户端进行显示。

### 终端服务器

* 获取终端服务器类型

reset –r

* 设置中断服务器的显示窗口

1. 获取终端尺寸

stty size

1. 设定列的值

stty cols N

1. 设定行的值

stty line N

### SecureCRT

* 修改最大显示列数

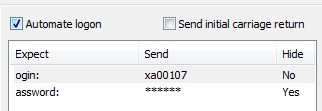
1. 设置Options 🡪 Global Options 🡪 Terminal 🡪 Appearance中Maximum Columns。
2. 重新连接。

* 常见问题

1. 流控配置导致不能输入。

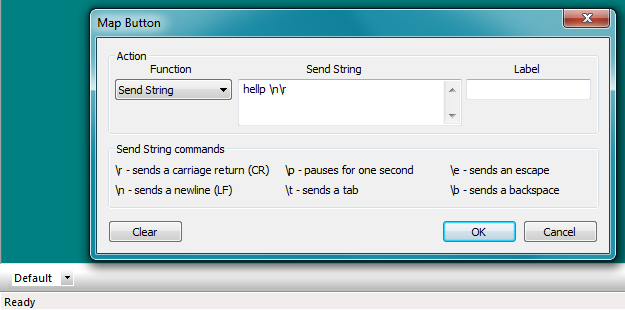
Session Options 🡪 Connection 🡪Serial 🡪 Flow Control，将原先默认选中的 RTS/CTS取消掉，再重新connect开发板，再次连上后，此时就可以从键盘输入了。

* 自动登录



* 自动发送

1. 右键点击状态栏，弹出bar选项，选择button bar;
2. 右键点击button bar，弹出button 选项，学则new button，弹出按键发送菜单；
3. 输入要发送的字符串，点击OK；



* 其他终端工具

1. putty

# Office

## PSPad

### hex显示格式

* 显示单位

setting 🡪program settings🡪hex editor🡪bytes per colum

## Word

### 大小写转换

* 不区分文本内容的转换。

首先选中需要转换的文本区域，然后使用系统字体选项中的大小写转换功能，选择不同的转换方法，确定即可。

* 特定内容的文本转换。

使用系统的替换功能，选中<使用通配符>选项，输入区分大小写的源文本和目标文本，选全部替换即可。

### 自动编辑

* 拼写检查

1. 取消Option🡪Proofing中自动检测，然后点击“重新检测”。
2. 对于outlook，Option🡪mail🡪spelling and audo correct
3. 按F7进入错误管理。

### 域显示

排版时目录和超链接显示{ xxxx}之类的代码，看起来不直观，这个代码是成为<域>的功能，如果要看其内容，解决方法如下：

* 如果只改变单个域，单击此域或域结果，按下 Shift+F9 组合键进行切换。
* 如果要应用到所有域，按下 Alt+F9 组合键进行切换。

### 目录页码编号

排版时需要把目录所在页面和正文所在的页面分别编码，可在需要分离的页面如下操作：

1. 点击菜单栏中：页面布局-分隔符-分节符；
2. 选择分节符:下一节的下一页；
3. 点击菜单栏中：插入-页码；
4. 选择页码:设置页码格式(可直接在页眉或页脚处手动输入页码格式)；
5. 关闭页眉页脚。

### 目录项页码格式

自动生成的目录项其页码格式和主页面页眉或页脚中的格式是一样的，如果要改变目录中的格式，可按照如下方式进行：

1. 根据目录页码格式需要，更改页眉或页脚的页码显示格式；
2. 更新目录；
3. 再次根据页码要求，改回页眉或页脚的页码显示格式。

### 目录项小圆点格式

1. 自动生成的目录圆点有的看起来比较稀疏看起来比较粗糙，解决方法：
2. 选中整个目录；
3. 点击菜单栏中：开始-段落-制表位；
4. 在制表位对话框中，在“前导符”中选择圆点(根据实际需要选择前导符，其他选项不要改)；
5. 点击确定即可；
6. 对于圆点比较粗的目录项，直接双击圆点选中后取消字体加粗即可。

### 文档比较

review🡪compare

### 图片合成

1. 将图片粘贴到WORD，然后使用插入图形功能直接在原始图片上绘图；
2. 绘制完成后将原始图片和绘制的图片分别粘贴到绘图板不同区域；
3. 使用透明选择将附加绘制的图片选择，并移动到原始图片之上；
4. 选中需要的区域保存。

### 粘贴代码

* html格式

1. 将代码转化为html格式，然后在浏览器上复制文本，并粘贴到word.
2. 代码复制时采用html格式(比如:nodepat)，然后粘贴

* 直接插入代码截图。
* 增加背景

page layout🡪page borders🡪shading🡪style

### 快捷键

* 复制并粘贴

Ctrl+d

* 向后对齐

Shift+tab

## Excel

### Filter

1. 选择要设置的单元格；
2. 选择菜单栏的“数据”→“有效性”→出现“数据有效性”弹出窗口；
3. 在“设置”选项中→“有效性条件”→“允许”中选择“序列”→右边的“忽略空值”和“提供下拉菜单”全部打勾;
4. 在“来源”下面输入数据→按“确定”就OK了。

### 输入运算符

运算符会被自动解析为数学运算，输入按如下方法。

1. 先将该单元格设置成文本格式，然后即可输入。
2. 在输入减号前，输入英文的单引号（强制使该单元格格式为文本），然后输入减号。进行替换。
3. 复制一个减号，如sss-，复制这里面的减号，然后黏贴需要输入减号的单元格。
4. 其他（输出文本的函数）。

### 百分比

输入格式更改为percentage，使用：条件格式(conditionalFormatting)进行设置。

### 文本转换

1. mm:ss转s

=TEXT("0:"&A1,"[s]")

使用字符串连接符号&在mm:ss前增加0:，即0:mm:ss，然后格式化为[s]格式，即转化为秒。

### 排序

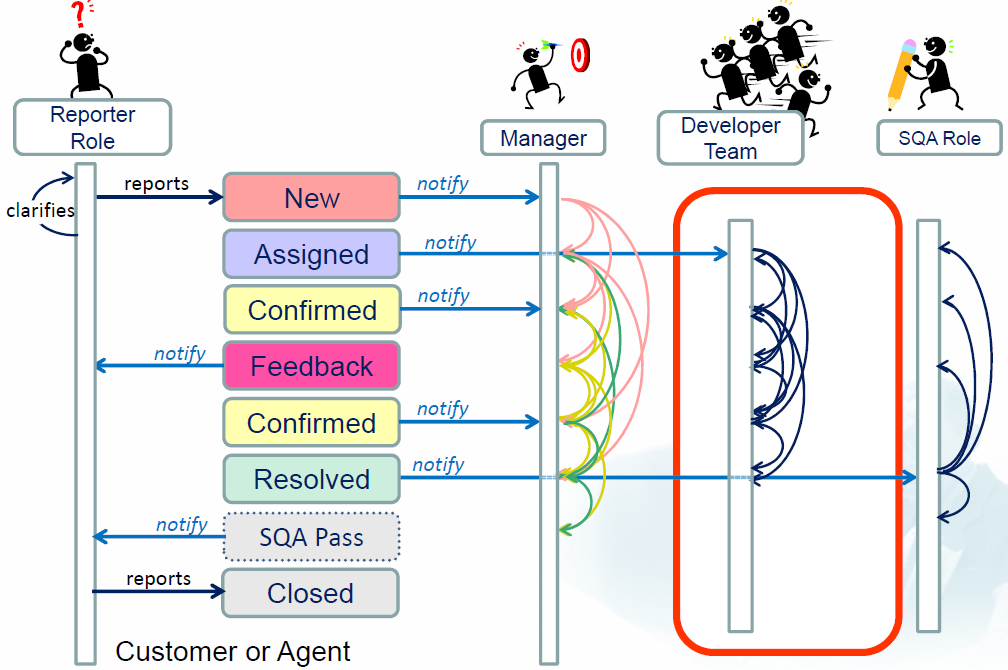
data🡪sort选项。对于某些列的数据二次处理，增加辅助列，然后进行排序。

# Manage

## maints

### 操作步骤

1. Mantis Account
2. Apply to Manager for “Project” Account
3. Issue Flow



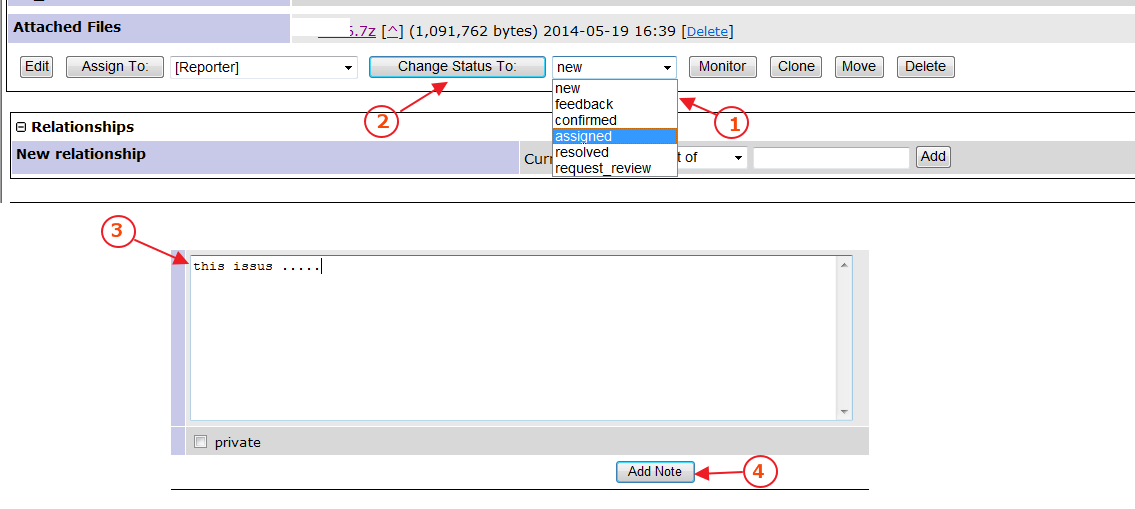
### 状态更新

1. Issue Status

|  |  |
| --- | --- |
| Issue Status | 含义 |
| New | 新Issue |
| Assigned | Issue已分给工程师，但解Issue的人员还没看到 |
| Acknowledged | 解问题的人员上Mantis看到此Issue，即给Acknowledged，表示收到问题了 |
| Confirmed | 解问题的人员确定有这个Issue，并可以复制 |
| Feedback | 这个是对SCL或SQA, 例如需要SS提供什么方法, 或釐清的机置，需同时改Assign to |
| Resolved | RD问题已解决 |
| SQA Pass | QA已测过 |
| Closed | SS已Closed此Issue |

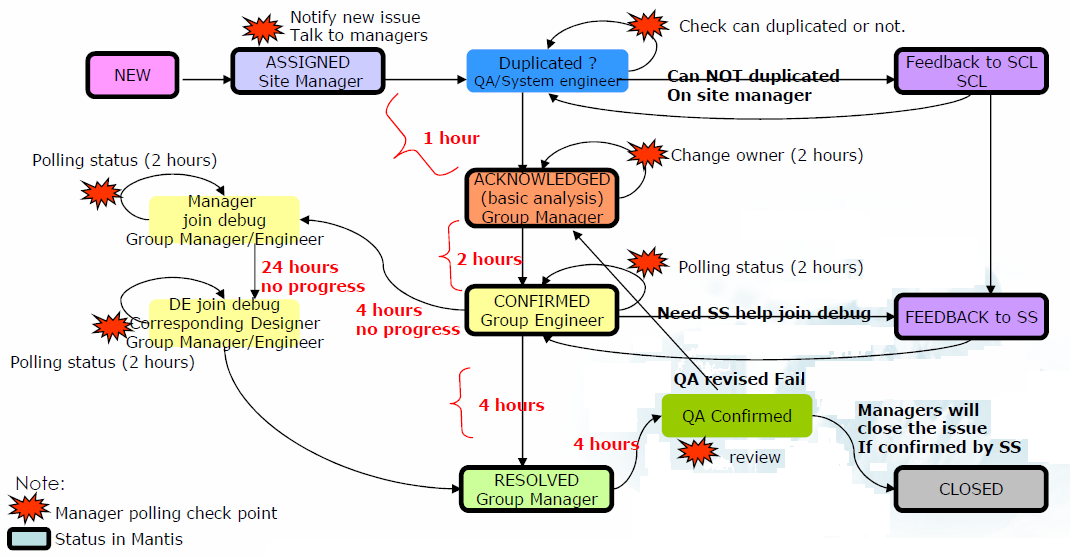
1. update Issue Status

如需状态更新，则如需操作



如果不更新状态，也可以只增加note.

### 状态转换

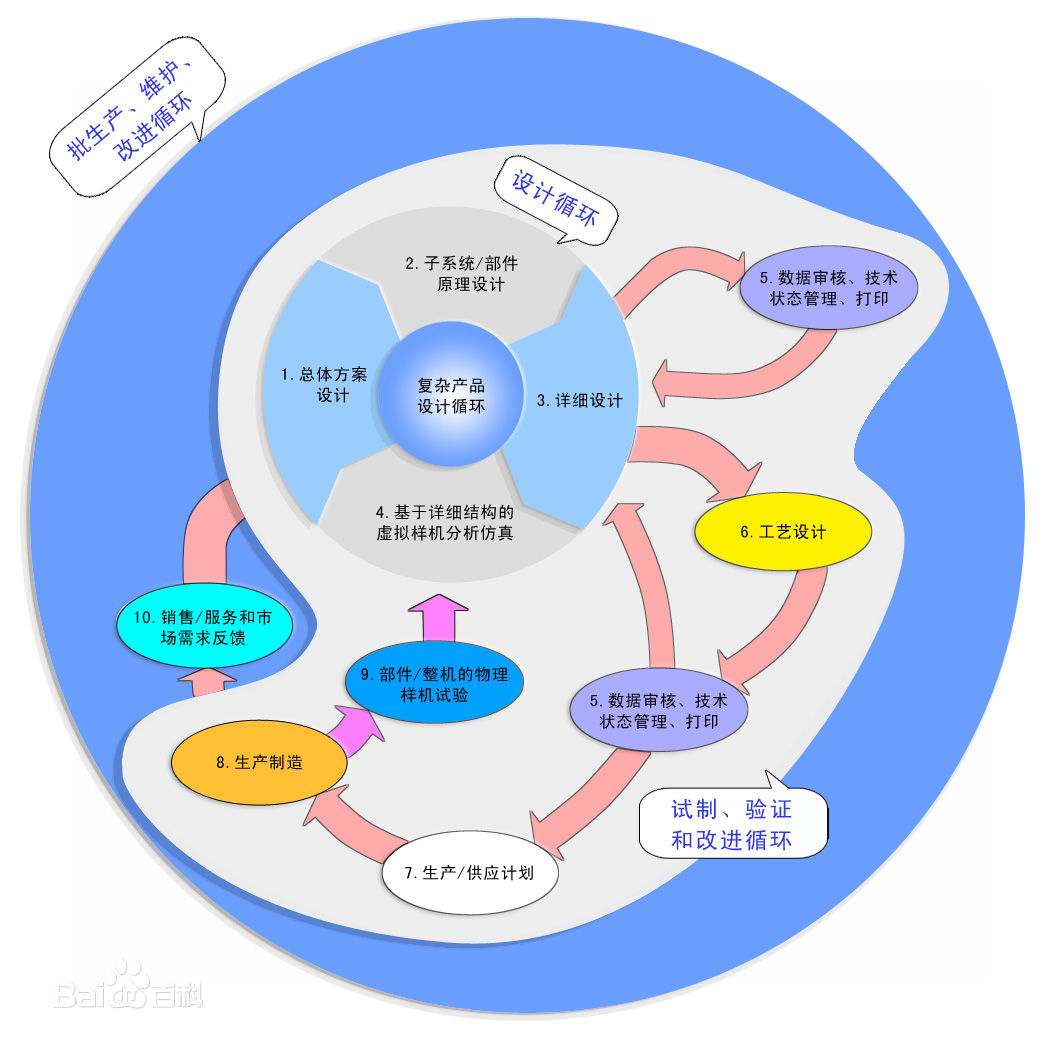


## PDM

### 状态转换

产品数据管理PDM（ProductDataManagement）技术很难有一个准确的定义加以描述。1995年初，主要致力于研究PDM技术和相关计算机集成技术的国际权威咨询公司CIM data给PDM作了一个概括性的定义：“PDM是一门用来管理所有与产品相关的信息和所有与产品相关的过程的技术”。这个定义从广义的角度解释了PDM技术。但就现阶段PDM的发展情况而言，可以给出一个较为具体的定义：“PDM技术以软件技术为基础，是一门管理所有与产品相关的信息(包括电子文档、数字化文档数据库记录等)和所有与产品相关的过程（包括审批/发放、工程更改、一般流程、配置管理等）的技术。提供产品全生命周期的信息管理，并可以在企业范围内为产品设计与制造建立一个并行化的协作环境。”

制造过程数据文档管理PDM 为管理企业的生产资源和制造过程数据而设计，能有效的组织工艺过程卡片、零件蓝图、三维数模、刀具清单、质量文件和数控程序等生产作业文档，实现企业车间现场无纸化生产。





## CQ

## ERP

### 过程资产积累

* 个人文档书写标准

1. 章节格式

* 文档工具统一标准

1. 流程图：ppt
2. UML图:
3. 工作日志：
4. 进度控制：
5. QA
6. 项目配置，比如烧录，产品特性，由具体负责人维护，统一呈现。

## SVN

### 常见错误

* Can’t move ‘.svn/tmp/entries’ to ‘.svn/entries’

该问题是 NTFS的实现bug，跟Windows Search 这项服务有关，运行services.msc，找到这项服务，stop并disable。

## Git

### 创建本地仓库

* 初始化一个目录作为初始版本

进入目录，执行git init命令，则在目录下生成.git文件夹，用于存放版本信息。

* 创建一个本地分支从另一个作为服务器的分支

git clone <分支的路径>

### 添加和删除

* 添加所有文件

git add \*

* 添加隐藏文件

git –f 文件名

### 提取和提交

* 同步另一个分支的修改到本地

1. git pull
2. git fetch

* 提交本地修改到版本控制

git commit -a

* 推送本地修改到服务器

1. 推送提交的本地修改：git push
2. 推送单个tag，命令格式为：git push tagname
3. 推送所有tag，命令格式为：git push --tags

### 常用信息查看

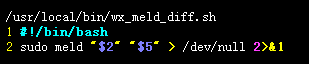
* 查看服务器分支信息

git remote -v

### 配置比较工具

* 使用可视化工具meld

创建脚本，使git diff的参数传递与meld工具的对应参数一致，然后增加配置项: diff.external=**/**usr**/**local**/**bin**/**wx\_meld\_diff.sh， 脚本内容如下：



* 忽略文件权限修改

如果不需要提交或者用比较工具查看一些仅修改了权限但没修改内容的文件，则可以让git忽略文件读写权限的修改，增加配置项：core.filemode false，则提交时或者比较时则会忽略仅修改了权限的文件。

### 网络代理配置

安装代理软件后，将代理软件配置为对8087端口监听。然后在git中增加配置:

git config --global http.proxy http://127.0.0.1:8087

git config --global https.proxy http://127.0.0.1:8087

git config --global http.sslVerify false

### patch功能

* 创建patch

git format-patch master --stdout > test.patch

* 应用patch

git apply test.patch

* 查看patch修改点

git apply --stat test.patch

* 检查patch是否会出错

git apply --check test.patch

## English

### 字符读法

* 键盘字符对照表

|  |  |  |  |
| --- | --- | --- | --- |
| symbol | 英文 | 含义 | 符号 |
| ~ | tilde |  |  |
| wave mark |  |
| xbar | 位于字母上 |
| ` | backquote |  | · |
|  | 年代简写,如1999年的简写为`99 |
| ! | exclamation |  | #1 |
| @ | at |  | #2 |
| # | number | 后跟数字 | #3 |
| pound | 拨号器或键盘 |
| hash | 哈希算法 |
| sharp | 如C#，F# |
| $ | dollar |  | ￥ |
|  | 形似S,source，表示参数 |
| % | percent |  | …… |
| ^ | caret | 用于“增补”文字 | #6 |
| exponent | 位于字母右上方，表示指数幂 |
| up mark |  |
| & | ampersand |  | #7 |
| and |  |
| \* | asterisk |  | #8 |
| star |  |
| multiply |  |
| ( ) | paren left/right |  | #9, #0 |
| opening/closing parentheses |  |
| opening/closing round bracket | 英式读法 |
| \_ | underscore |  | —— |
| underline |  |
| dash(两个- -) |  |
| - | minus | 减号 |  |
| negative (sign) | 负数符号 |
| hyphen |  |
| middle line |  |
| + | plus | 加号 |  |
| positive (sign) | 正数符号 |
| = | equal (mark) |  |  |
| { } | brace left/right |  |  |
| opening/closing curly bracket | 英式读法 |
| [ ] | bracket left/right |  | 【】 |
| opening/closing square bracket | 英式读法 |
| | | vertical bar |  |  |
| pipe |  |
| \ | backslash |  | 、 |
| : | colon |  |  |
| ; | semicolon |  |  |
| " | double quote |  | “ ” |
| second (sign) | 时间简写中表示“秒” |
| double prime | 代数中的符号简记，比如:A`` |
| ' | quote |  | ‘ ’ |
| apostrophe | 单词中省略字母或名词所有格 |
| minute (sign) | 时间简写中表示“分” |
| prime | 代数中的符号简记，比如:A` |
| < > | opening/closing angle bracket |  | 《 》 |
| less/greater than |  |
| smaller/bigger than |  |
| , | comma | 分隔断句 | ， |
| thousand,million,billion | 由在数字中的位置决定 |
| . | period |  | 。 |
| dot | 网址或文件名后缀分隔 |
| ? | question |  | ？ |
| / | slash |  | 、 |
| 🡪 | arrow |  |  |
| Esc | escape | 跳出，逃跑 |  |
| Fn | function n | 功能键F1~F12 |  |
| PrtScn | print screen | 打印屏幕 |  |
| SysRq | system request | 系统请求 |  |
| ScrLk | scroll lock | 页面滚动锁定 |  |
| Tab | tabulate | 将内容排列成表格的式样 |  |
| CapsLk | captions lock | 字母大小写锁定 |  |
| Ctrl | control | 控制 |  |
| Alt | alter | 改变 |  |
| PgUp | page up | 向上一页 |  |
| PgDn | page down | 向下一页 |  |
| NmLk | Number Lock | 数字键锁定 |  |
| Ins | Insert | 输入模式切换为插入模式 |  |

* 非键盘特殊符号

|  |  |  |
| --- | --- | --- |
| √ | tick |  |
| × | cross |  |
| ‖ | parallel |  |
| // | comment mark |  |
| ... | ellipsis |  |
| § | section |  |
| 。 | period | . |
| ￥ | yuan mark |  |
| 《》 | French quotes | < > |
| -- | dash | 两个- |
| ※ |  |  |
| 『』 |  |  |

* 数学符号

|  |  |  |
| --- | --- | --- |
| ± | plus or minus | 正负号 |
| × | is multiplied by | 乘号 |
| ÷ | is divided by | 除号 |
| ≠ | is not equal to | 不等于号 |
| ≡ | is equivalent to | 全等于号 |
| ≌ | is equal to or approximately eq | 等于或约等于号 |
| ≈ | is approximately equal to | 约等于号 |
| ≮ | is not less than | 不小于号 |
| ≯ | is not more than | 不大于号 |
| ≤ | is less than or equal to | 小于或等于号 |
| ≥ | is more than or equal to | 大于或等于号 |
| ‰ | permillage mark | 千分号 |
| ∞ | infinity | 无限大号 |
| ∝ | varies as | 与…成比例 |
| ∵ | since; because | 因为 |
| ∴ | hence | 所以 |
| ∠ | angle | 角 |
| ⌒ | semicircle | 半圆 |
| ⊙ | circle | 圆 |
| ○ | circumference | 圆周 |
| π | pi | 圆周率 |
| △ | triangle | 三角形 |
| ⊥ | perpendicular to | 垂直于 |
| ∪ | union of | 并，合集 |
| ∩ | intersection of | 交，通集 |
| ∫ | the integral of | …的积分 |
| ∑ | (sigma) summation of | 总和 |
| ° | degree | 度 |
| ℃ | Celsius system | 摄氏度 |

* mark和sign区别

英语中symbol表示的是“符号”（所有手写、机写的可辨认的），所有的数学符号（如正、负、加、减、乘、除）是读“某某sign”的，而其它的一般读“某某mark”。

* 数字分隔符表示

表示数字时，“,”要根据它的位数读thousand,million,billion之类，（请注意：这样的用法仅在英语中看到，有些欧洲国家如德国却刚刚相反：用句点进行3位数字之间的分隔，用逗点作小数点）；

* 括号的配对读法

括号的符号名称属于集合名称（也就是该名称既可以用于左边的符号，也可用于右边的符号，就好比parent/brother/sister等集合名词既可以指父亲/哥哥/姐姐，也可以指母亲/弟弟/妹妹），如果要进一步区分，则前者需要加opening，后者需要加ending或closing，这跟elder brother/sister表示哥哥/姐姐，而younger brother/sister表示弟弟/妹妹是一个道理。这种用法在英语中叫“开关用法”：前者表示开启，后者表示关闭，是“配对”使用的。

### 计时

在欧美历史书上，耶稣诞生前的年代被称为“西元前”，常用B.C（Before Christ 的缩写，意为“基督以前”）表示；耶稣诞生那年以后的年代是“西元后”，简称“西元”。

既不存在西元前0年，也不存在西元0年，西元前1年之后的一年，就是西元元年(即公元1年)。

### 计数