IBM PC／AT

硬件和 XENIX 系统资料汇编之四

# XENIX软件命令参考手册

张洪斌　吴志敏　胡民远　　译

北京科海总公司培训中心

中国科学院软件研究所

一九八七年四月

IBMPC／AT硬件和XENIX系统资料汇编

（之四）

# XENIX 软件命令参考手册

张世斌　吴志敏　胡民远　　译

北京科海总公司培训中心
中国科学院软件研究所

一九八七年四月

**编辑**：科海培训中心教材部
**发行**：科海培训中心资料组
**地址**：北京2725信箱　科海培训中心
资料组
（北京海淀路332路黄庄站旁）
**印刷**：河北省蠡县印刷厂

# 编 者 序

IBM PC已从PC、PC/XT推进到PC/AT。PC/AT以Intel80286为主CPU，具有丰富的硬件资源。鉴于目前DOS系统基本上是一种单用户系统，许多硬件资源未得到充分利用，许多用户都要求在PC/AT上配备多用户多任务的XENIX系统。

XENIX系统是UNIX系统在以Intel为主CPU的微机上的实现，该系统由Microsoft公司开发。目前在PC/AT上运行的XENIX相当于UNIX的System Ⅲ或System Ⅴ。

为了更好地在国内推广PC/AT及其兼容机，中国科学院软件研究所在其雄厚的技术力量基础上，积多年研究、开发UNIX系统之经验，开发成功多种XENIX中西文信息处理系统并移植到几乎所有PC/AT的兼容机和部分386机上。为了更好地推广XENIX中英文信息处理系统，科海培训中心和中国科学院软件研究所组织了一批专家和技术人员，收集并编译整理了有关XENIX及IBM PC/AT的全部技术资料。

这些资料包括以下几类：

Ⅰ. IBM PC/AT硬件资料，包括硬件安装及组装手册、技术手册和维护手册。

Ⅱ. XENIX基本系统的安装、基本用户指南、命令参考手册、系统管理手册和直观shell手册。

Ⅲ. XENIX开发系统方面的软件开发手册、库函数程序员手册、系统调用和子程序手册。

Ⅳ. XENIX系统上运行的汇编语言和各种高级语言（C、Fortran、Cobol、Basic）的用户指南和参考手册。

Ⅴ. XENIX正文格式化处理手册。

Ⅵ. XENIX系统上配备的最新版本INFORMIX和UNIFY数据库管理系统用户及参考手册。

Ⅶ. 中西文兼容的C—XENIX系统安装和基本使用手册。

全套资料约400万字，分装成20本。

全书的主要译校任务由中国科学院软件研究所的专家、技术人员承担，科海培训中心负责编辑、印刷和发行工作。

由于时间仓促，本资料汇编中必有不少错漏之处，敬请读者批评指正，以便再版时更正。

主编 孙玉方

董洪皋

# 本 书 概 述

本书对用于《IBM PC XENIX软件开发系统》中的命令和可用于操作系统核心的系统命令进行了说明。用于XENIX软件开发系统的命令用字母（CP）标记，在第一部分中加以说明。字母C代表“命令”，字母P代表“程序”。由子程序和系统调用组成的系统命令用字母（S）标记。这些命令在第二部分中加以说明。附录A是对于系统调用和库函数的交叉参考。列于此附录下的函数是在不同的库中建立的和直接调用系统原语的那些函数。

为了参考其它书目，现说明下列字母的含义：（C）代表“命令”，（M）代表“杂类”；（F）代表文件格式部分。这些函数列于《IBM PC XENIX 命令参考手册》中，字母（CT）代表了那些来自《IBM PC XENIX 正文处理系统》的函数。这些函数列于《IBM PC XENIX 正文格式指南》一书的附录A中。

**有关出版物**

* 《IBM PC XENIX 软件开发指南》
* 《IBM PC XENIX 库函数程序员指南》
* 《IBM PC XENIX C编译参考手册》
* 《IBM PC XENIX 汇编参考手册》
* 《IBM PC XENIX 安装指南》
* 《IBM PC XENIX 直观shell》
* 《IBM PC XENIX 系统管理手册》
* 《IBM PC XENIX 基本操作指南》
* 《IBM PC XENIX 命令参考手册》

# 目　录

# 第一部分 软件开发命令

## 引　　论

这一部分说明在“软件开发系统”中用CP加以标志的命令的用法。为了与“IBM PC XENIX命令参考手册”和“IBM PC XENIX正文格式指南”两书中的命令区别开来，这一部分中的每一条命令都用字母（CP）加以标记。

下面的这条命令作为例子说明这一部分的编排格式。这条EXAMPLE（CP）命令不是一条真正的XENIX命令，而是作为一个样本说明这一部分的命令是什么形式出现的。

**EXAMPLE（CP）**

**名字**

example——说明本书编排方式。

**句法**

除非特别说明，本部分的命令任选项和其他自变量按如下句法给出：

name [options] [cmdars]

其中：

name　一个可执行文件的文件名或者路径名。

options　单一的字母表示命令的任选项。一般地，大多数任选项之前用一条横线“-”表示。任选项既可以分别独立地说明，例如：-a -b -c -d，又可以组合在一起说明，例如：-abcd。说明任选项的方法依赖于每个命令的格式。最新的任选项说明方法是可以将自变量赋给任选项。例如，许多命令的-f任选项，可以后接文件名作为自变量。

cmdarg　路径名或其他不以“-”开始的命令自变量。它也可以仅仅是一条横线“-”，这时它代表标准输入。

**参见**

getopt（C），getopt（S）

**诊断**

结束时，每条命令返回两个字节的状态标志。其中之一由系统提供，说明产生终止的原因；另一个（正常终止的情况下）是由程序提供的（见wait(s)，exit(s)）。对于正常终止，状态标志的前一个字节置为0；对于成功地执行，后一字节也习惯地置为0，而非零则表示出错。参数错误，数据有问题或不可存取数都属于出错之列。对于这种返回，可以冠以不同的名称，例如：出口码(exit code)出口态(exit status)或返回码（return code），而且只在特殊情况下加以说明。

**注释**

并不是所有命令都需要有任选项和自变量的。

**ADB（CP）**

**名字**

adb—调用一个通用的调试程序。

**格式**

adb [-*w*] [-*prompt*] [*objfil* [*core file*] ]

**说明**

adb用来检测文件并为IBM PC XENIX程序提供一个受控环境。

通常*objfil*是可执行程序文件，最好包含一个符号表；如果没有符号表，那么即使该文件仍能被检查，但adb的符号特性也不能用了。*objfil*的缺省形式是a.out。*corefile*假定是执行*objfil*后产生的一个内存映象文件；*corefile*的缺省形式是core。

对adb的请求由标准输入读入，而回答在标准输出上给出。如果写了-*w*标志，则需要时创建*objfil*和*corefile*，并且将这两个文件打开准备读写，以便可以用adb修改文件。Quit (ctrl-\)和Interrupt (Del) 导致机器返回去执行下一条adb命令。-p任选定义了提示字符串。它可以是字符的任意组合，缺省时为一星号“*”。

对adb的请求一般形式是：

[*address*] [, *count*] [*command*] [; ]

如果出现*address*项，则将*dot*置为*address*，*dot* 的初值是0。

*address*是一特殊表达式，形式如下：

[*segment*] *offset*

其中*segment*给出一特定文件或者一数据段的地址；*offset*给出从一段开始的位移量。如果没有给出*segment*，则要用命令中给出的最后一个段值。

对一个地址的解释取决于使用它的上下文。如果正在调试一个子进程，则地址就按通常意义解释成该子进程的地址空间中的地址。若想了解地址映射的详情，参见“地址”。对大多数命令来说，*count*指出了该命令可执行多少次。*count*的缺省值是1。

1. **表达式**

*dot*的值。

\+ 由当前增量增加后的*dot*的值。

\- 由当前增量减去后的*dot*的值。

" 打入的最后一个*address*。

integer 以0开始表示是八进制数；以#或0x开始表示是十六进制数；否则是十进制数。

integer.fraction

32位浮点数。

'cccc'

多至4个字符的ASCII值。可以用“\”来转义一个“'”。

<name *name*的值。name可以是变量名，也可以是寄存器名。adb持有一些仅由单个字母或数字命名的变量(见“变量”)。如果*name*是寄存器名，则该寄存器的值是由*core file*内系统标题得到的。寄存器的名字是：**ax bx cx dx di si bp fl ip cs ds ss es sp**。名字fl表示状态标志。

symbol

*symbol* 是以非数字打头的一串由大、小写字母、下线及数字组成的序列。*sym-*

*bol*的值取*objfil*中的符号表。如果需要，可在s*ymbol*前加上“_”或“～”。

_symbol

在C语言中，一个外部符号“真名”由“-”开始。为了与程序中的内部或隐含的变量相区分，可能需要使用这种名字。

(exp)　表达式*exp*的值。

**单目算符**

*exp　exp指出的单元中的内容。

-exp　整数反号。

～exp　按位求补。

**双目算符**

（它比单目算符的结合能力弱，但具有左结合性。）

$e_1+e_2$　整数加法。

$e_1-e_2$　整数减法。

$e_1*e_2$　整数乘法。

$e_1\%e_2$　整数除法。

$e_1\&e_2$　按位逻辑加。

$e_1|e_2$　按位逻辑减。

$e_1\%e_2$　$e_2$除$e_1$所剩余数。

$e_1\#e_2$　$e_1$舍入到$e_2$的下一个倍数。

2.命令

多数命令由一个动词后跟一个或一串修饰符构成。下面的动词是可用的（命令“/”后可以跟“*”；欲得到进一步的细节，见“地址”）。

?f　按格式f从*objfil*中的*address*指示的位置开始打印。

/f　按格式f从*corefile*中的*address*所指的位置开始打印。

=f　按格式f指定的风格打印*address*值（对于i，把引用随后的字的指令部分按格式打印）。

*format*　由一个或多个表明打印风格的字符组成。对每一个这样的字符都可以在其前加上一个十进制整数，以表示该字符重复的次数，在一步步按打印格式执行过程中，每个格式字符所产生的量加入到*dot*中，如果没有给出打印格式，则用最后一个格式。可用的格式符是以下这些：

o2　按八进制打印2个字节。所有由adb输出八进制都前置一个0。

O4　按八进制打印4个字节。

q2　打印带正负号的八进制数。

Q4　打印带正负号的长八进制数。

d2　打印十进制数。

D4　打印长十进制数。

x2　按十六进制打印2个字节。

X4　按十六进制打印4个字节。

u2　作为不带符号的十进制数来打印。

U4　打印长的不带符号的十进制数。

f4　把32位字位串的值当成浮点数打印。

F8　打印双字长浮点数。

b1　按八进制打印编址的字节。

c1　打印编址的字符。

C1　按照下面的转义约定打印编址的字符。

字符值从000到040被打印成一个@后接在0100到0140区间中与000到040对应的字符。字符@被打印成@@。

sn　从编址的字符开始打印，直到碰到零字符。

Sn　用@转义约定，打印一字符串。n是包括零结尾在内的串的长度。

Y4　按照日期格式（见ctime (S)）打印4个字节。

i n　作为机器指令来打印。n是机器指令占用的字节数。这种打印风格导致变量1和变量2被分别置成源和目标的位移量。

a0　以符号的形式打印dot 的值。这些符号已经过检查，以证实它们属于以下几种合适的类型：

/　局部或全部的数据符号。

?　局部或全部的正文符号。

=　局部或全部的绝对符号。

A0　按绝对形式打印*dot*值。

p2　将编址按符号形式打印，符号检索规则与上面的a一样。

t0　当前面加一个整数时，制表定位到下一个适当的制表定位处。例如，8t间隔8个空格跳到下一个制表定位处。

r 0　打印一个空格。

n0　打印一个换行。

"..." 0

打印被括起来的字符串。

^　当前增量加入*dot*。不打印。

+　*dot*加1。不打印。

-　*dot*减1。不打印。

换行

如果前一命令临时地建立*dot*增量，换行使之成为永久的增量。按*count*等于1重复前面的命令。

[? /] l value mask

从*dot* 指示的位置开始的单词被mask屏蔽掉，同value比较，直到找到与value相同的位置。如果用L，则每次按4个字节而不是2个字节比较。如果不匹配，则*dot*就不改变；否则将*dot*置成与value相匹配的位置。如果省略了mask，则用-1来代替。

[? /] w value…

将2个字节的value写入所指定位置。如果命令是W，则写4个字节。当向子进程地址空间写的时候，不允许是奇数地址。

[? /] m segnum fpos size

对给定段的文件的位置和大小建立新值。如果不给出大小，则只改变文件位置。segnum 一定要是一个段的段号，而且这个段已存在于存储映象之中(见“地址)。如果给出“？”，说明的是正文段；如果是“/”，则为数据段。

[? /] M segnum fpos size

在存储映象中创建一新段。该段给出文件的位置fpos，物理大小size。segnum不能为存储映象中已有的。如果是“？”任选，则创建正文段；如果为“/”则创建数据段。

>name

将指定的变量或寄存器的位置赋给*dot*。

! 调用shell读行中跟在“!”后的其余字符。

$ modifer

杂类命令。可用的modifer是：

<f 从文件f中读出命令，然后返回。

>f 将输出写入文件*f*，如果文件*f*不存在则创建它。

r 打印出由ip指示的通用寄存器及指令。*dot*置成ip。

b 打印所有的检查点以及与其相关的命令及计数。

c C语言栈的回溯。如果给了*address*，则它作为当前栈区的地址（而不是bp）。如果是C，则每个活动函数中全部自动的和静态的名字和（16位的）值都被打印出来。如果给出了*count*，则仅打印出前*count*个栈区的内容。

e 打印外部变量的名和值。

w 置输出页宽为*address*（缺省值为80）。

s 为检查符号匹配而设置的界为*address*（缺省值为255）。

o 所有输入输出都看成八进制数。

d 所有输入输出都看成十进制数。

x 所有输入输出都看成十六进制数。

q 从adb中退出。

v 按八进制打印所有非零变量。

m 打印地址映象。

: modifier

管理子进程，可用的修饰符为：

br c 在*address*指示的位置设置一个检查点。该检查点产生中断前执行*count*-1次。每次遇到这个检查点都执行命令c。如果这条命令将*dot*置成0，则该检查点就产生中断。

dl 删除*address*处的检查点。

r [argaments]

把*objfil*当作子进程运行。如果*address*显式给出，则就从这点进入程序；否则，程序从标准的入口点进入。*count*指出了在停止前要略过多少检查点。子进程的自变量可以在命令的同一行上给出。由<或>开始的自变量

可以为命令准备好标准的输入输出。在子进程的入口处所有的信号都打开。

R [arguments]

除了把自变量传给程序之前要先传给shell之外，与r命令相同。这意味着shell的元字符也可以作为文件名。

co *s*

子进程继续进行并把信号s传给它。（见signal（S）。如果给出了*address*。如果没有给出信号，则发出使子进程停止的信号。跳过检查点同r一样处理。

s *s*

除了该子进程每次走一步作*count*次以外，与co一样。如果没有当前子进程，则*objfil*作为子进程来执行，如同r中一样。在这种情况下不能发信号；命名后边剩下的部分当作子进程的自变量。

k　　终止当前子进程（如果有的话）。

**3.变量**

adb 提供了一这变量。指定的变量在开始时由adb置值，以后不用。以数字代表的变量专门作通信之用。用法如下：

0　最后一个打印的值。

1　一个指令源的最后位移部分。

2　变量1的先前值。

在入口处，根据文件*corefile*中的系统标题将下面的变量置值。如果*corefile*不是core 文件，则这些值就根据*objfil*文件设置。

b　数据段的基址。

d　数据段的大小。

e　入口点。

m　执行类型。

n　段号。

s　栈段大小。

t　正文段大小

**4.地址**

adb 中的地址指的是文件中的位置或指的是存储中的位置。当在存储中没有当前进程时，adb 的地址作为文件位置考虑，并且正文和数据是从*objfil*和*corefile*中读出的。当有进程时，比如一个:r命令之后，地址作为存储位置考虑。

所有的正文段和数据段都有相应的存储映象入口。每一入口有单独一个段号。另外，每一入口还据有文件位置在该段第一个字节和在文件中该段的物理大小。当运行一个进程时，段的入口有一虚拟大小，它说明在当前时刻存储中该段的大小。在执行中，这个大小可以改变。

在给出地址且无进程运行的情况下，则与该地址相应的文件位置按如下方式计算：

effective－file－address＝file－position＋offset

如果有进程运行，则存储的位置就是所给段的位移量offset。当且仅当下列情况，