AIX 7.2 版

作業系統管理

請注意

使用本資訊及其支援的產品之前，請先閱讀第 347 頁的『注意事項』中的資訊。

此版本適用於 AIX 7.1 版，以及所有後續版次和修改內容，除非新版中另有指示。

目錄

關於本文件

本文件包含使用者和系統管理者所需的完整資訊，當您執行諸如備份及還原系統、管理實體及邏輯儲存體、調整適當的分頁空間大小等作業時，需要使用這些資訊來選取適當的選項。它提供如何執行管理邏輯磁區、儲存體及資源等作業的完整資訊。系統使用者可以學習如何執行諸如 執行指令、處理程序、處理檔案及目錄，以及基本列印等作業。

其他有助於使用者及系統管理者的主題包括：建立及調整分頁空間大小、管理虛擬記憶體、備份及還原系統、管理硬體及虛擬裝置、使用「系統資源控制器 (SRC)」、保護檔案安全、使用儲存媒體、自訂環境檔案，以及撰寫 Shell Script。您也可以在作業系統隨附的文件 CD 中找到本文件。

強調顯示

下列為本文件所使用的強調顯示慣例：

粗體	指定指令、子常式、關鍵字、檔案、結構、目錄及系統已預先定義其名稱的其他項目。亦指定圖形物件，如使用者選取的按鈕、標籤及圖示。
斜體	指定由使用者提供實際名稱或值的參數。
等寬字體	指定特定資料值的範例、類似您可能看到顯示文字的範例、類似您以程式設計者身分可能寫出程式碼部分的範例、系統的訊息，或您應實際鍵入的資訊。

AIX 中區分大小寫

AIX® 作業系統中的所有項目皆區分大小寫，亦即區分大寫和小寫字母。例如，您可以使用 **ls** 指令來列出檔案。如果您鍵入 LS，則系統會回應找不到該指令。同樣地，**FILEA**、**FiLea** 及 **filea** 即使位於相同目錄中，仍是三個不同的檔名。為了避免執行不想要的動作，請務必使用正確的大小寫。

ISO 9000

本產品的開發和製造過程使用 ISO 9000 註冊的品質系統。

作業系統管理

系統管理者及使用者可以學習如何執行諸如執行指令、處理程序、處理檔案與目錄、備份與還原系統、管理實體及邏輯儲存體，以及基本列印等作業。

其他有助於使用者及系統管理者的主題包括： 建立及調整分頁空間大小、管理虛擬記憶體、備份及還原系統、管理硬體及虛擬裝置、使用「系統資源控制器 (SRC)」、保護檔案安全、使用儲存媒體、自訂環境檔案，以及撰寫 shell Script。您也可以在作業系統隨附的文件 CD 中找到本主題。

作業系統管理是個人的工作，在 UNIX 文件中通常是指系統管理者。遺憾的是，僅有少數系統管理者的活動才能真正在意義上稱為管理。本章及相關手冊就是要輔助系統管理者完成其大量職責。

本作業系統會提供它自己特殊的系統管理支援版本，以使得使用更加簡便，並增進安全及完整性。

作業系統管理的新增功能

閱讀「作業系統」和裝置管理主題集合的新增或重大變更相關資訊。

如何查看新增功能或變更資訊

在 PDF 檔案中，您可能會在新增及變更資訊中看到修訂標記（>| 及 |<）。

2018 年 9 月

下列資訊是對此主題集合所做之更新的摘要：

· 已新增 第 52 頁的『IBM License Metric Tool 的 AIX 使用度量資料（SLM 標籤）』 主題的相關資訊。

作業系統管理

您可以使用指令來管理系統啟動及備份，從而關閉系統、系統 shell 與環境、系統資源，以及 AIX 的其他不同部分。

作業系統管理是個人的工作，在 UNIX 文件中通常是指系統管理者。遺憾的是，僅有少數系統管理者的活動才能真正在意義上稱為管理。本章及相關手冊就是要輔助系統管理者完成其大量職責。

本作業系統會提供它自己特殊的系統管理支援版本，以使得使用更加簡便，並增進安全及完整性。

可用的系統管理介面

除了慣用的指令行系統管理以外，本作業系統還提供 SMIT 介面。

下列是 SMIT 介面：

· 「系統管理介面工具 (SMIT)」，是功能表型的使用者介面，它可從您所選擇的選項來建構指令並執行它們。

  使用 SMIT，您可以：

  – 安裝、更新及維護軟體
  – 配置裝置
  – 將磁碟儲存單位配置到磁區群組及邏輯磁區中
  – 製作並擴充檔案系統及分頁空間
  – 管理使用者及群組
  – 配置網路及通訊應用程式
  – 列印
  – 執行問題判斷

- 排程工作
- 管理系統資源及工作量
- 管理系統環境
- 管理叢集系統資料

· 物件導向的圖形使用者介面，它支援與 SMIT 相同的系統管理作業，但會簡化系統管理作業，方法是：

- 透過錯誤檢查及對話框設計減少使用者錯誤
- 為新的或複雜的作業提供逐步程序
- 為更有經驗的管理者提供進階選項
- 使複雜的資料或系統物件之間的關係能讓人看得更清楚
- 發生預先定義的事件時，監視系統活動並警示管理者
- 提供相關的輔助說明、概觀、秘訣及線上文件的鏈結

軟體重要產品資料

軟體產品及其可安裝選項的某些相關資訊是在「軟體重要產品資料 (SWVPD)」資料庫中進行維護。

SWVPD 是由一組指令及「物件資料管理程式 (ODM)」物件類別（用於維護軟體產品資訊）所組成。SWVPD 指令提供給使用者，用於查詢 (**lslpp**) 及驗證 (**lppchk**) 已安裝的軟體產品。ODM 物件類別可定義所維護之軟體產品資訊的範圍及格式。

installp 指令使用 ODM 來維護 SWVPD 資料庫中的下列資訊：

· 已安裝之軟體產品的名稱

· 軟體產品的版本

· 軟體產品的版次，可指出軟體產品之外部程式設計介面的變更

· 軟體產品的修正層次，可指出不影響軟體產品之外部程式設計介面的變更

· 軟體產品的修訂層次，可指出後來建立到正規修正層次中的小更新

· 修訂識別欄位

· 組成軟體產品或選項之檔案的名稱、總和檢查及大小

· 軟體產品的安裝狀態：套用中、已套用、確認中、已確認、拒絕中或已破壞。

作業系統更新

作業系統套裝軟體被分成檔案集，每個檔案集包含一組邏輯相關的客戶可遞送檔案。 每個檔案集均可個別安裝及更新。

檔案集的修訂版本，是使用版本、版次、維護及修訂 (VRMF) 層次來進行追蹤。依照慣例， 每次套用 AIX 檔案集更新時，都會調整修訂層次。每次套用 AIX 維護套件或技術層次時，都會調整修正層次，並將修訂層次重設為零。AIX 版本（如 AIX 6.1）的初次安裝稱為基本安裝。作業系統會提供其特性及功能的更新，這些更新可能包裝成維護套件、 技術層次、暫時性程式修訂 (PTF) 或服務套件（PTF 群組）。

維護套件及技術層次

維護套件及技術層次提供旨在將版次升級的新功能。VRMF 的維護部分會在維護套件中更新。例如，AIX 6.1 的第一個維護套件是 6.1.1.0；第二個維護套件是 6.1.2.0，依此類推。若要列出維護套件，請使用 **oslevel -r** 指令。

若要判斷已在特定系統上安裝的維護套件或技術層次，請鍵入：

```
oslevel
```

若要判斷哪些檔案集需要更新，以使系統達到特定的維護套件或技術層次（在此範例中是 6.1.1.0），請使用下列指令：

```
oslevel -l 6.1.1.0
```

若要判斷是否安裝了建議的維護套件或技術層次（在此範例中是 6100-02），請使用下列指令：

```
oslevel -r 6100-02
```

若要判斷哪些檔案集需要更新，以使系統達到 6100-02 維護套件或技術層次，請使用下列指令：

```
oslevel -rl 6100-02
```

若要判斷特定檔案集的維護套件或技術層次（在此範例中是 bos.mp），請使用下列指令：

```
lslpp -L bos.mp
```

PTF
於版次之間，您可能會接收到更正或防止發生特定問題的 PTF。 特定的安裝可能需要部分、全部，或者甚至不需要可用的 PTF。

建議維護套件
建議維護套件是一組已經在各技術層次間通過廣泛測試的 PTF，並建議用於預防性維護方面。

臨時修正程式
臨時修正程式類似 PTF，但通常是在沒有 PTF 時才會提供。當 PTF 會將系統升級至下一個維護層次，但使用者可能想要其系統仍維持在現行層次時，也會推出臨時修正程式。

若要判斷版本及版次層次、維護套件、技術層次、服務套件層次，以及哪些檔案集需要升級才能達到特定層次時，請參閱 oslevel 及 lslpp 指令（ 位於 *Commands Reference*）。

系統啟動

當基本作業系統啟動時，系統會起始一系列複雜作業。在正常狀況下，會自動執行這些作業。

當您要指示系統重新啟動時，存在部分狀況；例如，讓系統辨識新安裝的軟體，重設週邊裝置，執行諸如檢查檔案系統的日常維護作業，或者從系統當機或損毀中回復。如需這些程序的相關資訊，請參閱：

相關工作
重建損毀的開機映像檔
下面的程序說明如何識別損毀的開機映像檔及如何重建開機映像檔。

管理系統啟動
有多個當您要將系統開機或重新開機時可能遇到的範例情節。若要將系統關機或重新開機，您可以使用 shutdown 或 reboot 指令。當多個使用者登入系統時，您應該使用 shutdown 指令。

重新啟動執行中的系統
因為 **reboot** 可能會讓應該終止的處理程序繼續執行， 對所有系統而言，**shutdown** 是較常用的方法。

有兩種方法可以將系統關機並重新開機：**shutdown** 與 **reboot**。當多個使用者登入系統時，一律使用 **shutdown** 方法。

作業	SMIT 捷徑	指令或檔案
重新啟動多使用者系統	smit shutdown	**shutdown -r**
重新啟動單一使用者系統	smit shutdown	**shutdown -r** 或 **reboot**

從遠端對無回應系統進行重新開機
遠端重新開機機能允許透過本機（整合）系統埠，將系統重新開機。

POWER5 整合系統埠類似序列埠， 但是系統埠只能用於特別支援的功能。

當該埠接收到 **reboot_string** 時，即會重新啟動系統。這個機能在系統未回應、但尚能檢修系統埠岔斷時非常有用。一次僅可在一個本機系統埠上啟用遠端重新開機。使用者被期望提供自己對該埠的外部安全保護。此機能在最高裝置岔斷類別中執行，若是 UART（通用非同步接收/傳輸）快速清除傳輸緩衝區時失敗，可能會導致其他裝置的資料遺失（如果那些裝置的緩衝區在此期間溢位的話）。建議僅在已懸置且無法從遠端登入的機器上使用此機能來重新開機。檔案系統不會同步化，而尚未清除的資料可能會流失一部分。強力建議您啟用遠端重新開機的埠不要用於任何其他目的（特別是檔案轉送），以防止無意中重新開機。

有兩個本機系統埠屬性可控制遠端重新開機作業。

reboot_enable

指出是否在收到遠端 **reboot_string** 時就馬上啟用此埠來重新開機，如果要馬上啟用此埠，是否在重新開機前進行系統傾出。

```
no         - 指出遠端重新開機已停用
reboot     - 指出遠端重新開機已啟用
dump       - 指出遠端重新開機已啟用，且在重新啟動系統之前，將在主要
             傾出裝置上進行系統傾出
```

reboot_string

當啟用遠端重新開機特性時，指定序列埠將掃描的遠端 **reboot_string**。當啟用遠端重新開機特性，並在埠上接收到 **reboot_string** 時，會傳輸一個 > 字元，同時系統也準備好要重新開機了。如果接收到 1 字元，則會重新啟動系統；1 以外的任何字元都會中斷重新開機處理程序。**reboot_string** 的最大長度是 16 個字元，且不可以包含空格、冒號、等號、空值、換行或 Ctrl-\ 字元。

可透過 SMIT 或指令行來啟用遠端重新開機。若使用 SMIT，已配置的 TTY 可使用**系統環境 -> 管理遠端重新開機機能**路徑。另外，當配置新的 TTY 時，可從**新增 TTY** 或**變更/顯示 TTY 的性質**功能表來啟用遠端重新開機。 您可透過**裝置 -> TTY** 路徑來存取這些功能表。

從指令行使用 **mkdev** 或 **chdev** 指令來啟用遠端重新開機。例如，下列指令會啟用遠端重新開機（使用傾出選項），並將重新開機字串設為 **tty1** 上的 **ReBoOtMe**。

```
chdev -l tty1 -a remreboot=dump -a reboot_string=ReBoOtMe
```

此範例僅使用資料庫中的現行 **reboot_string**，在 **tty0** 上啟用遠端重新開機（將在下一次重新開機時生效）。

```
chdev -P -l tty0 -a remreboot=reboot
```

如果 tty 正作為標準埠，那麼在啟用遠端重新開機之前，您必須使用 **pdisable** 指令。您可以使用 **penable** 以便在以後重新啟用這個埠。

相關資訊

系統埠與序列埠之間的功能差異

從硬碟開機以進行維護

您可以從硬碟以維護模式開機：

先決要件

可啟動的抽取式媒體（磁帶或 CD-ROM）不得在磁碟機中。同時，請參照硬體文件取得特定指示，以在特定機型上啟用維護模式開機。

程序

從硬碟以「維護」模式開機：

1. 若要重新開機，請將機器關機，再重新開啟電源，或按下重設鈕。
2. 按下按鍵順序，從硬體文件中指定的維護模式中重新開機。
3. 機器將啟動至配置主控台裝置的某個點。

   如果必須擷取系統傾出，則會在主控台上顯示系統傾出功能表。

   註：

   a. 如果在擷取傾出時無法配置主控台，則系統將懸置。系統必須從抽取式媒體中開機才能擷取傾出。

   b. 一按下重設按鈕，系統會自動傾出至指定的傾出裝置。若要變更執行系統中的主要或次要傾出裝置指定，請參閱 **sysdumpdev** 指令。
4. 如果沒有任何系統傾出，或如果已經複製，則會顯示診斷作業指示。請按 Enter 鍵，繼續執行**功能選取**功能表。

5. 從**功能選取**功能表中，您可以選取診斷或單一使用者模式：

   單一使用者模式：若要在單一使用者環境中執行維護，請選擇此選項（選項 5）。系統繼續開機，並進入單一使用者模式。此模型可執行需要系統處於單機模式的維護工作，且必要時可以執行 **bosboot** 指令。

相關資訊

啟動系統傾出

啟動損毀的系統

在部分案例中，您可能必須啟動沒有適當關機而停止（損毀）的系統。

此程序的必要條件為：

· 系統因異常狀況而損毀，且未適當地關機。

· 您的系統已關閉。

此程序包含當系統無法自損毀中回復時，如何開機的基本方法。請執行下列步驟：

1. 請確定所有硬體與週邊裝置均已正確地連接。
2. 開啟所有週邊裝置。
3. 查看畫面以取得自動硬體診斷的相關資訊。
   a) 如果任何硬體診斷測試未順利完成，請參照硬體文件。
   b) 如果所有硬體診斷測試均順利完成，請開啟主機。

重設不明的 *root* 密碼

下列程序說明如何在無法使用或不知道系統 root 密碼時，回復 root 專用權的存取權。

下列程序需要一些系統停機時間。如有可能，請將停機時間排定在對工作量影響最小的期間，以免發生資料或功能遺失。

此作法範例情節中的資訊已使用特定版本的 AIX 進行測試。依據您的 AIX 版本及層次，所得到的結果可能大不相同。

1. 將與現行安裝相同版本及層次的產品媒體插入適當的磁碟機。
2. 開啟機器電源。
3. 出現圖示螢幕或是當您聽到兩聲嗶嗶聲時，請重複按 F1 鍵，直到出現 **系統管理服務**功能表為止。
4. 選取**多重開機**。
5. 選取**安裝來源**。
6. 選取容納產品媒體的裝置，然後選取**安裝**。
7. 選取 AIX 版本圖示。
8. 按 F1 鍵然後按 Enter 鍵，將您的現行系統定義為系統主控台。
9. 選取您偏好的語言編號，然後按 Enter 鍵。
10. 鍵入 3，選擇**啟動維護模式來進行系統復原**，然後按 Enter 鍵。
11. 選取**存取 Root 磁區群組**。
    這時會顯示一則訊息， 說明如果您此時變更 root 磁區群組，則在沒有重新開機的情況下，您將無法返回「安裝」功能表。
12. 鍵入 0，然後按 Enter 鍵。
13. 從清單鍵入適當的磁區群組號碼，然後按 Enter 鍵。
14. 鍵入 1，選取**存取此磁區群組並啟動 shell**，然後按 Enter 鍵。
15. 在 #（數字符號）提示中，於指令行提示中鍵入 **passwd** 指令，重設 root 密碼。

    例如：

```
# passwd
Changing password for "root"
root's New password:
Enter the new password again:
```

16. 若要將緩衝區中的所有項目寫入硬碟並將系統重新開機，請鍵入下列指令：

```
sync;sync;sync;reboot
```

當登入畫面出現時，您在步驟 15 中設定的密碼現在應該允許存取 root 專用權。

相關資訊
passwd 指令
reboot 指令

啟動具有平面圖形的系統
如果機器只安裝了平面圖形子系統，之後若新增其他圖形配接卡至系統，會發生下列狀況：

1. 新圖形配接卡新增至系統，並安裝其相關裝置驅動程式軟體。
2. 系統重新開機，並發生下列其中一項：
   a. 如果系統主控台定義為 /dev/lft0（**lscons** 會顯示此資訊），則在重新開機時，系統會要求使用者選取系統主控台要使用的顯示器。如果使用者選取圖形配接卡（非 TTY 裝置），則它也會變成新的預設顯示器。如果使用者選取 TTY 裝置代替 LFT 裝置，則不會出現任何系統登入。請重新開機，則會顯示 TTY 登入畫面。假設使用者新增其他圖形配接卡至系統，且系統主控台是 LFT 裝置，則使用者不應選取 TTY 裝置作為系統主控台。
   b. 如果系統主控台定義為 TTY，則在重新開機時，新增加的顯示器配接卡會變成預設顯示器。

      註：因為 TTY 原是系統主控台，因此它仍將繼續作為系統主控台。
3. 如果系統主控台是 /def/lft0，則在重新開機後，會停用 DPMS，以便在畫面上顯示無限期的系統主控台選項文字。若要重新啟用 DPMS，請再次重新啟動系統。

部署執行層次 *Script* 執行
執行層次 Script 可讓使用者在變更執行層次時，啟動及停止已選取的應用程式。

將執行層次 Script 放在專屬於執行層次的 /etc/rc.d 子目錄中：

- /etc/rc.d/rc2.d
- /etc/rc.d/rc3.d
- /etc/rc.d/rc4.d
- /etc/rc.d/rc5.d
- /etc/rc.d/rc6.d
- /etc/rc.d/rc7.d
- /etc/rc.d/rc8.d
- /etc/rc.d/rc9.d

/etc/rc.d/rc 會在執行層次變更時，執行在指定目錄中找到的 Script，首先執行停止應用程式 Script，然後執行啟動應用程式 Script。

註：以 K 開頭的 Script 就是停止 Script，而以 S 開頭的 Script 就是啟動 Script。

修改 */etc/inittab* 檔案
有四個指令可用來修改 etc/inittab 檔案中的記錄。

新增記錄 - mkitab 指令

若要新增記錄至 /etc/inittab 檔案，請在指令提示上鍵入下列：

```
mkitab Identifier:Run Level:Action:Command
```

例如，若要新增 tty2 的記錄，請在指令提示上鍵入下列：

```
mkitab tty002:2:respawn:/usr/sbin/getty /dev/tty2
```

在上面的範例中：

項目	說明
tty002	指定您要定義執行層次的物件。
2	指定執行此處理程序的執行層次。
respawn	指定 **init** 指令應對此處理程序採取的動作。
/usr/sbin/getty /dev/tty2	指定要執行的 shell 指令。

變更記錄 - chitab 指令

若要變更 **/etc/inittab** 檔案的記錄，請在指令提示上鍵入下列：

```
chitab Identifier:Run Level:Action:Command
```

例如，若要變更 tty2 的記錄，使此處理程序在執行層次 2 及 3 上執行，請鍵入：

```
chitab tty002:23:respawn:/usr/sbin/getty /dev/tty2
```

在上面的範例中：

項目	說明
tty002	指定您要定義執行層次的物件。
23	指定執行此處理程序的執行層次。
respawn	指定 **init** 指令應對此處理程序採取的動作。
/usr/sbin/getty /dev/tty2	指定要執行的 shell 指令。

列出記錄 - lsitab 指令

若要列出 /etc/inittab 檔案中的所有記錄，請在指令提示上鍵入下列：

```
lsitab -a
```

若要列出 /etc/inittab 檔案中的特定記錄，請鍵入：

```
lsitab ID
```

例如，若要列出 tty2 的記錄，請鍵入：lsitab tty2。

移除記錄 - rmitab 指令

若要從 /etc/inittab 檔案中移除記錄，請在指令提示上鍵入下列：

```
rmitab ID
```

例如，若要移除 tty2 的記錄，請鍵入：rmitab tty2。

相關概念

系統執行層次

系統執行層次可指定系統狀態並定義會啟動哪些處理程序。

重新啟動非作用中的系統

您的系統可能因為硬體問題、軟體問題或兩者，變成非作用中的系統。

此程序會引導您完成更正問題並重新啟動系統的步驟。如果在完成此程序後，系統仍處於非作用中，請參照硬體文件中的問題判斷資訊。

請使用下列程序來重新啟動非作用中系統：

硬體檢查
有數種可以用來檢查硬體的程序。

執行下列各項以檢查硬體：

檢查電源：

如果系統上的「開啟電源」燈號仍然開啟，請前往底下的**檢查操作面板顯示器**。

如果系統上「開啟電源」燈號未開啟，請檢查電源是否開啟及系統插頭是否已插入。

檢查操作面板顯示器：

如果您的系統有操作員面板顯示器，請檢查它是否顯示任何訊息。

如果系統上的操作員面板顯示器是空白的，請前往底下的**啟動顯示器或終端機**。

如果系統上的操作員面板顯示器不是空白的，請查看該裝置的服務手冊，以取得「操作員面板顯示器」中有關數字所代表的相關資訊。

啟動顯示器或終端機：

檢查顯示器或終端機的數個部分，如下所示：

· 確定顯示器纜線已固定連接在顯示器與主機上。
· 確定鍵盤纜線已固定連接。
· 確定滑鼠纜線已固定連接。
· 確定已打開顯示器，且「開啟電源」燈號是亮的。
· 調整顯示器的亮度控制。
· 確定終端機的通訊設定正確。

如果系統現在已能啟動，表示硬體檢查已更正了問題。

相關工作

重新啟動系統
除了檢查硬體並檢查處理程序外，您還可以重新啟動系統，來重新啟動非作用的系統。

檢查處理程序
已停止或停滯的處理程序可能會讓系統變成非作用中。

檢查處理程序
已停止或停滯的處理程序可能會讓系統變成非作用中。

請執行下列以檢查系統處理程序：

1. 重新啟動行捲動
2. 使用 Ctrl+D 鍵順序
3. 使用 Ctrl+C 鍵順序
4. 自遠端終端機或主機登入
5. 遠端結束已停止的處理程序

重新啟動行捲動

使用 Ctrl-S 鍵順序停止行捲動，並執行下列動作來重新啟動：

1. 啟動發生問題處理程序的視窗或 shell。
2. 按下 Ctrl-Q 鍵順序重新啟動捲動。
   Ctrl-S 鍵順序會停止行捲動，Ctrl-Q 鍵順序則重新啟動行捲動。

如果捲動檢查並未更正非作用中系統的問題，請跳至下一節**使用 Ctrl-D 鍵順序**。

使用 Ctrl-D 鍵順序：

1. 啟動發生問題處理程序的視窗或 shell。

2. 按下 Ctrl-D 鍵順序。Ctrl-D 鍵順序會傳送檔案結束 (EOF) 信號至處理程序。Ctrl-D 鍵順序會關閉視窗或 shell，並讓您登出。

如果 Ctrl-D 鍵順序並未更正非作用中系統的問題，請跳至下一節**使用 Ctrl-C 鍵順序**。

使用 Ctrl-C 鍵順序：

執行下列動作，結束已停止的處理程序：

1. 啟動發生問題處理程序的視窗或 shell。
2. 按下 Ctrl-C 鍵順序。Ctrl-C 鍵順序會停止現行搜尋或過濾。

如果 Ctrl-C 鍵順序並未更正非作用中系統的問題，請跳至下一節**從遠端終端機或主機登入**。

從遠端終端機或主機登入：

使用兩種方式之一，從遠端登入：

- 如果有多台終端機連接至您的系統，則請從另一台終端機登入系統。
- 鍵入 **tn** 指令，從網路上的其他主機登入（如果系統有連接網路），如下所示：

```
tn YourSystemName
```

當您使用 **tn** 指令時，系統會要求標準登入名稱與密碼。

如果您可以從遠端終端機或主機登入系統，請跳至下一節**遠端結束已停止的處理程序**。

如果您無法從遠端終端機或主機登入系統，則需要重新啟動系統。

您也可以啟動系統傾出，以便判斷系統變成非作用中的原因。

遠端結束已停止的處理程序：

您可以執行下列動作，自遠端終端機結束已停滯的處理程序：

1. 鍵入下列 **ps** 指令， 列出作用中的處理程序：

```
ps -ef
```

-e 及 **-f** 旗標可識別所有作用中及非作用中的處理程序。

2. 指定已停滯處理程序的處理程序 ID。

如在識別處理程序時需要協助，請使用 **grep** 指令及搜尋字串進行搜尋。例如，若要結束 **xlock** 處理程序，請鍵入下列以尋找處理程序 ID：

```
ps -ef | grep xlock
```

grep 指令可以讓您搜尋 **ps** 指令的輸出，以便識別特定處理程序的處理程序 ID。

3. 鍵入下列 **kill** 指令結束處理程序：

註： 您必須具有 root 使用者權限，才能在不是由您起始的處理程序上使用 **kill** 指令。

```
kill -9 ProcessID
```

如果您無法識別有問題的處理程序，則最近啟動的處理程序可能就是非作用中系統的原因。如果您認為的確如此，請結束最近的處理程序。

如果您的處理程序檢查無法更正非作用中系統的問題，則需要重新啟動系統。

相關概念

硬體檢查

有數種可以用來檢查硬體的程序。

相關工作

重新啟動系統

除了檢查硬體並檢查處理程序外，您還可以重新啟動系統，來重新啟動非作用的系統。

相關資訊
系統傾出機能

重新啟動系統
除了檢查硬體並檢查處理程序外，您還可以重新啟動系統，來重新啟動非作用的系統。

如果第 8 頁的『硬體檢查』及第 8 頁的『檢查處理程序』的程序無法更正非作用中系統的問題，則您必須重新啟動系統。

註 : 重新啟動系統之前，請完成系統傾出。

1. 檢查開機裝置的狀態。

   您的系統可使用抽取式媒體、外部裝置、小型電腦系統介面 (SCSI) 裝置、 整合裝置電子 (IDE) 裝置或區域網路 (LAN) 來開機。請決定哪一種方法適用於您的系統， 然後遵循下列指示來檢查開機裝置：

   - 若為抽取式媒體（如磁帶），請確定媒體已正確地插入。
   - 若為 IDE 裝置，請驗證是否每一個配接卡的 IDE 裝置 ID 設定都是唯一的。如果只有一個裝置連接至配接卡，則 IDE 裝置 ID 必須設定為主要裝置。
   - 若為外部連接的裝置（如磁帶機），請確定：
     - 已開啟裝置的電源。
     - 裝置纜線已正確地連接至裝置與主機。
     - 備妥指示器是亮的（如果裝置有指示器的話）。
   - 若為外部 SCSI 裝置，請驗證 SCSI 位址設定是唯一的。
   - 若為 LAN，請驗證網路已啟動且可運作。

   如果開機裝置皆正確運作，請繼續下一個步驟。
2. 請執行下列動作載入作業系統：
   a) 關閉系統電源。
   b) 等待一分鐘。
   c) 打開系統電源。
   d) 等待系統開機。

如果作業系統無法載入，請從維護模式或硬體診斷程式啟動硬碟。

如果您仍然無法重新啟動系統，請使用 SRN 向您的客戶服務代表報告非作用中系統所發生的問題。

相關概念
硬體檢查
有數種可以用來檢查硬體的程序。

相關工作
檢查處理程序
已停止或停滯的處理程序可能會讓系統變成非作用中。

相關資訊
系統傾出機能

建立開機映像檔
若要安裝基本作業系統或存取不是從系統硬碟機開機的系統，您需要開機映像檔。此程序說明如何建立開機映像檔。開機映像檔會隨著裝置類型而改變。

當第一次安裝系統時，**bosboot** 指令會從 RAM（隨機存取記憶體）磁碟檔案系統映像檔及作業系統核心建立開機映像檔。會將開機映像檔轉送到特定的媒體（如硬碟）。當機器重新啟動時，會將開機映像檔從該媒體載入記憶體中。如需 **bosboot** 指令的相關資訊，請參閱 **bosboot**。

相關的 RAM 磁碟檔案系統含有下列裝置的裝置配置常式：

- 磁碟
- 磁帶

· CD-ROM

· 網路記號環、乙太網路或 FDDI 裝置

· 您必須具有 root 使用者權限，才能使用 **bosboot** 指令。

· /tmp 檔案系統至少必須有 20 MB 的可用空間。

· 實體磁碟必須包含開機邏輯磁區。欲判斷要指定哪些磁碟裝置，請在指令提示上鍵入下列指令：

```
lsvg -l rootvg
```

lsvg -l 指令會列出 root 磁區群組（rootvg) 上的邏輯磁區。從此清單中，您可以找出開機邏輯磁區的名稱。

然後，在指令提示上鍵入下列：

```
lsvg -M rootvg
```

lsvg -M 指令會列出包含各種邏輯磁區的實體磁碟。

在開機邏輯磁區上建立開機映像檔
如果要安裝基本作業系統（不論是新的安裝或更新），請呼叫 **bosboot** 指令以在開機邏輯磁區上放置開機映像檔。開機邏輯磁區是在安裝期間內，由「邏輯磁區管理程式 (LVM)」在磁碟上所建立的一種實際連續的區域。

如需此程序的必要條件清單，請參閱第 10 頁的『建立開機映像檔』。

bosboot 指令可執行下列動作：

1. 檢查檔案系統，查看是否有足夠的空間可以建立開機映像檔。
2. 使用 **mkfs** 指令及原型檔案建立 RAM 檔案系統。
3. 呼叫 **mkboot** 指令，將核心與 RAM 檔案系統合併到開機映像檔。
4. 將開機映像檔寫入開機邏輯磁區。

若要在硬式磁碟的預設開機邏輯磁區上建立開機映像檔，請在指令提示上鍵入下列：

```
bosboot -a
```

或：

```
bosboot -ad /dev/ipldevice
```

註： 如果在建立開機映像檔時發生 **bosboot** 指令失敗，請勿重新開機。請先解決問題，再執行 **bosboot** 指令才能順利完成。

您必須重新啟動系統，新的開機映像檔才能夠使用。

建立網路裝置的開機映像檔
您可以建立用於乙太網路開機或記號環開機的開機映像檔。

如需此程序的必要條件清單，請參閱第 10 頁的『建立開機映像檔』。

若要建立乙太網路開機的開機映像檔，請在指令提示上鍵入下列：

```
bosboot -ad /dev/ent
```

若為記號環開機，則請鍵入：

```
bosboot -ad /dev/tok
```

系統執行層次
系統執行層次可指定系統狀態並定義會啟動哪些處理程序。

例如，當系統執行層次是 3 時，會啟動定義為要在該執行層次上操作的所有處理程序。當接近開機處理程序的系統開機階段結尾時，會從 /etc/inittab 檔案的 initdefault 項目讀取執行層次。系統會在該執行層次上運作，直到收到要變更層次的信號。可使用 **init** 指令變更系統執行層次。對於為該處理程序定義執行層次的每個處理程序，/etc/inittab 檔案均包含一筆記錄。當系統開機時，**init** 指令會讀取 /etc/inittab 檔案，以判斷要啟動哪些處理程序。

下面是目前定義的執行層次：

項目	說明
0-9	當 **init** 指令變更為執行層次 0-9 時，它會結束在目前執行層次上的所有處理程序，然後重新啟動與新執行層次相關的任何處理程序。
0-1	保留以供未來作業系統使用。
2	預設的執行層次。
3-9	可以依據使用者的喜好設定來定義。
a、b、c	當 **init** 指令要求變更執行層次 **a**、**b** 或 **c** 時，它並不會結束在現行執行層次上的處理程序；它只會啟動指派為新執行層次的任何處理程序。
Q, q	告知 **init** 指令重新檢查 /etc/inittab 檔案。

相關工作
修改 /etc/inittab 檔案
有四個指令可用來修改 etc/inittab 檔案中的記錄。

識別系統執行層次
在作業系統上執行維護或變更系統執行層次之前，您可能需要檢查各種執行層次。

此程序說明如何識別系統運作的執行層次，以及如何顯示之前的執行層次歷程。**init** 指令會判斷系統執行層次。

識別現行執行層次

在指令行上，鍵入 cat /etc/.init.state。系統會顯示一位數；那就是現行執行層次。請參閱 **init** 指令或 /etc/inittab 檔案，以取得執行層次的詳細資訊。

顯示先前執行層次的歷程
您可以使用 **fwtmp** 指令來顯示之前的執行層次歷程。

註：您的系統上必須安裝 bosext2.acct.obj 程式碼，才能使用此指令。

1. 以 root 使用者的身分登入。
2. 在指令提示上鍵入下列：

```
/usr/lib/acct/fwtmp </var/adm/wtmp |grep run-level
```

系統會顯示如下的資訊：

```
run-level 2  0 1 0062 0123 697081013 Sun Feb  2 19:36:53 CST 1992
run-level 2  0 1 0062 0123 697092441 Sun Feb  2 22:47:21 CST 1992
run-level 4  0 1 0062 0123 698180044 Sat Feb 15 12:54:04 CST 1992
run-level 2  0 1 0062 0123 698959131 Sun Feb 16 10:52:11 CST 1992
run-level 5  0 1 0062 0123 698967773 Mon Feb 24 15:42:53 CST 1992
```

配置多使用者系統上的執行層次
您可以變更多使用者系統上的執行層次。

1. 檢查 /etc/inittab 檔案，確認您欲變更的執行層次可支援所要執行的處理程序。

   因為 getty 處理程序控制系統主控台的終端機線路存取及其他登入，所以它特別重要。請確定已在所有執行層次上啟用 getty 處理程序。

2. 使用 **wall** 指令通知所有使用者，指出您要變更執行層次並要求使用者登出。

   如需 **wall** 指令的相關資訊，請參閱 **wall**。
3. 使用 **smit telinit** 捷徑存取**設定系統執行層次**功能表。
4. 在**系統執行層次**欄位中，鍵入新執行層次。
5. 按 Enter 鍵，以執行此程序中的所有設定。
   系統回應會告知您哪些處理程序會因執行層次的變更而終止或啟動，並顯示訊息：

   ```
   INIT: New run level: n
   ```

   其中的 *n* 是新執行層次號碼。

配置單一使用者系統上的執行層次

您可以變更單一使用者系統上的執行層次。

1. 檢查 /etc/inittab 檔案，確認您欲變更的執行層次可支援所要執行的處理程序。

   因為 getty 處理程序控制系統主控台的終端機線路存取及其他登入，所以它特別重要。請確定已在所有執行層次上啟用 getty 處理程序。如需 inittab 檔案的相關資訊，請參閱 inittab。
2. 使用 smit telinit 捷徑存取**設定系統執行層次**功能表。

   如需 telinit 指令的相關資訊，請參閱 telinit。
3. 在**系統執行層次**欄位中，鍵入新的系統執行層次。
4. 按 Enter 鍵，以執行此程序中的所有設定。

   系統回應會告知您哪些處理程序會因執行層次的變更而終止或啟動，並顯示訊息：

   ```
   INIT: New run level: n
   ```

   其中的 *n* 是新執行層次號碼。

開機處理程序

有三種類型的系統開機，以及兩種在啟動作業系統時所需的資源。

開機處理程序期間，系統會測試硬體、載入並執行作業系統，以及配置裝置。若要開啟作業系統，必須要有下列資源：

· 打開或重設機器之後可載入的開機映像檔。

· 對 root (/) 及 /usr 檔案系統的存取權限。

有三種類型的系統開機：

項目	說明
硬碟開機	啟動機器用於正常作業。
無磁碟網路開機	透過網路遠端啟動無磁碟或無資料工作站。啟動機器用於正常作業。一或多台遠端檔案伺服器提供無磁碟或無資料工作站開機所需的檔案及程式。
維護開機	以維護模式從硬碟、網路、磁帶或 CD-ROM 啟動機器。系統管理者可以執行作業（如安裝新的或更新的軟體及執行診斷檢查）。

硬碟開機期間，會在安裝作業系統時建立的本端磁碟上找到開機映像檔。在開機處理程序期間，系統會配置在機器中找到的所有裝置，並起始設定系統操作所必要的其他基本軟體（如「邏輯磁區管理程式」）。在此處理程序的結尾，檔案系統會被裝載並備妥以供使用。

相同的一般基本要求適用於無磁碟網路用戶端。其亦需要開機映像檔及對作業系統檔案樹的存取權限。無磁碟網路用戶端沒有本端檔案系統，且會透過遠端存取的方式取得其所有資訊。

相關概念

處理系統開機

啟動系統以用於一般作業時，大部分使用者會執行硬碟開機。系統會尋找其磁碟機上開機處理程序必要的所有資訊。

維護開機處理程序
可能會發生需要開機來執行特殊作業（如安裝新的或更新的軟體， 執行診斷檢查或用於維護）的狀況。在此情況下，系統會從可開機的媒體（如 CD-ROM、DVD、磁帶機、網路或磁碟機）啟動。

RAM 檔案系統
RAM 檔案系統（開機映像檔的一部分）會完全常駐於記憶體，並包含允許開機處理程序繼續的所有程式。RAM 檔案系統中的檔案是開機類型專用的。

處理系統開機
啟動系統以用於一般作業時，大部分使用者會執行硬碟開機。系統會尋找其磁碟機上開機處理程序必要的所有資訊。

當以開啟電源開關（冷開機）啟動系統，或使用 **reboot** 或 **shutdown** 指令（暖開機）來重新啟動系統時，於系統備妥使用之前，必須發生一些事件。這些事件可以分成下列階段：

相關概念
開機處理程序
有三種類型的系統開機，以及兩種在啟動作業系統時所需的資源。

韌體階段
韌體會準備系統以載入並執行作業系統。

其起始設定階段包括下列步驟：

1. 韌體會對啟動作業系統所需的系統資源執行基本測試。
2. 韌體會檢查使用者開機清單（可用開機裝置清單）。您可以使用 **bootlist** 指令來變更此開機清單，使其符合您的需求。 若非揮發性隨機存取記憶體 (NVRAM) 中的使用者開機清單無效，或若找不到有效的開機裝置，則會檢查預設開機清單。不管是哪一種狀況，都會使用在開機清單中找到的第一個有效開機裝置來用於系統啟動。如果 NVRAM 中存在有效的使用者開機清單，則會按次序檢查清單中的裝置。如果不存在使用者開機清單，則會檢查匯流排上的所有配接卡及裝置。不管是那一種狀況，都會在連續的迴圈中檢查裝置，直到為系統啟動找到有效的開機裝置為止。

   註：系統會維護儲存於 NVRAM 中的預設開機清單，以使用於標準模式開機。個別服務模式開機清單也會儲存在 NVRAM 中，您必須參照模型的特定硬體指示，以瞭解如何存取服務模式開機清單。
3. 當找到有效的開機裝置時，會檢查第一個記錄或程式磁區號碼 (PSN)。如果它是有效的開機記錄，則會將它讀入記憶體，並將它新增至記憶體中的 IPL 控制區塊。主要開機記錄資料中包含開機裝置上開機映像檔的開始位置、開機映像檔的長度，以及載入記憶體中開機映像檔之位置的指示。
4. 從 NVRAM 中指定的位置開始，會將開機映像檔從開機裝置循序讀入記憶體。磁碟開機映像檔是由核心、RAM 檔案系統及基本自訂裝置資訊所組成。
5. 將控制移交給核心，它會開始系統起始設定。
6. 核心會執行 **init**，而 init 則會執行 `rc.boot` Script 的階段 1。

當核心起始設定階段完成時，會開始基本裝置配置。

基本裝置配置階段
init 處理程序會啟動 `rc.boot` Script。`rc.boot` Script 的第一階段會執行基本裝置配置。

`rc.boot` Script 的第一階段包括下列步驟：

1. 開機 Script 會呼叫 **restbase** 程式，使用壓縮的自訂資料於 RAM 檔案系統中建置自訂的「物件資料管理程式 (ODM)」資料庫。
2. 開機 Script 會啟動配置管理程式，該管理程式會存取階段 1 ODM 配置規則以配置基本裝置。
3. 配置管理程式會啟動 **sys**、**bus**、**disk**、SCSI 以及「邏輯磁區管理程式 (LVM)」及 rootvg 磁區群組配置方法。
4. 配置方法會載入裝置驅動程式、建立特殊檔案並更新 ODM 資料庫中的自訂資料。

啟動系統
此程序會完成系統開機階段。

1. **init** 處理程序會開始執行 rc.boot Script 的階段 2。rc.boot 的階段 2 包括下列步驟：
   a) 呼叫 **ipl_varyon** 程式以轉開 rootvg 磁區群組。
   b) 將硬碟檔案系統裝載至其正常裝載點上。
   c) 執行 **swapon** 程式以啟動分頁。
   d) 將自訂資料從 RAM 檔案系統中的 ODM 資料庫複製到硬碟檔案系統中的 ODM 資料庫。
   e) 結束 rc.boot Script。
2. rc.boot Script 的階段 2 完成之後，開機處理程序會從 RAM 檔案系統切換至硬碟上儲存的檔案系統。
3. 然後 **init** 處理程序會執行 /etc/inittab 檔案中之記錄所定義的處理程序。/etc/inittab 檔案的其中一個指令會執行 rc.boot Script 的階段 3，包括下列步驟：
   a) 裝載 /tmp 硬碟檔案系統。
   b) 啟動配置管理程式階段 2 以配置所有剩餘裝置。
   c) 使用 **savebase** 指令，將自訂的資料儲存至開機邏輯磁區。
   d) 結束 rc.boot Script。

在此處理程序的結尾，系統會啟動並準備好可供使用。

維護開機處理程序
可能會發生需要開機來執行特殊作業（如安裝新的或更新的軟體， 執行診斷檢查或用於維護）的狀況。在此情況下，系統會從可開機的媒體（如 CD-ROM、DVD、磁帶機、網路或磁碟機）啟動。

事件的維護開機順序與正常開機順序類似。

1. 韌體會對啟動作業系統所需的系統資源執行基本測試。
2. 韌體會檢查使用者開機清單。您可以使用 **bootlist** 指令來變更使用者開機清單，使其符合您的需求。如果不變性隨機存取記憶體（non-volatile random access memory，NVRAM）中的使用者開機清單無效，或如果找不到有效的開機裝置，則會檢查預設開機清單。 不論是哪一種狀況，都會使用在開機清單中找到的第一個有效開機裝置來用於系統啟動。

   註：若為正常開機，作業系統會維護儲存在 NVRAM 中的預設開機清單及使用者開機清單。同時會維護個別的預設開機清單及使用者開機清單，以在維護模式中啟動系統。
3. 當找到有效的開機裝置時，會檢查第一個記錄或程式磁區號碼 (PSN)。如果它是有效的開機記錄，則會將它讀入記憶體，並將它新增至記憶體中的起始程式載入 (IPL) 控制區塊。主要開機記錄資料中包含開機裝置上開機映像檔的開始位置、開機映像檔的長度，以及當開機映像檔載入記憶體後，開始執行進入點的偏移量。
4. 從 NVRAM 中指定的位置開始，會將開機映像檔從開機裝置循序讀入記憶體。
5. 將控制移交給核心，其會開始執行 RAM 檔案系統中的程式。
6. ODM 資料庫內容會判斷存在哪些裝置，且 **cfgmgr** 指令會動態配置所有找到的裝置（包括要包含根檔案系統的所有磁碟）。
7. 如果使用 CD-ROM、DVD、磁帶或網路來啟動系統，則不會轉開 rootvg 磁區群組（或 rootvg），因為 rootvg 可能不存在（如同在新的系統上安裝作業系統時一樣）。此時可能會進行網路配置。 執行維護開機時，不會產生分頁。

在此處理程序的結尾，系統會準備好可以進行安裝、維護或診斷。

註：如果從硬碟啟動系統，轉開 rootvg，在 RAM 檔案系統中裝載硬碟 root 檔案系統及硬碟 /usr 檔案系統，則會顯示一個可讓您進入各種診斷模式或單一使用者模式的功能表。 如果選取單一使用者模式，您可以繼續開機處理程序並進入單一使用者模式，其中 **init** 執行層次設為字母 S。然後，系統會備妥以進行維護、軟體更新或執行 **bosboot** 指令。

相關概念
開機處理程序

有三種類型的系統開機，以及兩種在啟動作業系統時所需的資源。

RAM 檔案系統

RAM 檔案系統（開機映像檔的一部分）會完全常駐於記憶體，並包含允許開機處理程序繼續的所有程式。RAM 檔案系統中的檔案是開機類型專用的。

維護開機 RAM 檔案系統可能沒有邏輯磁區常式，因為可能不需要轉開 rootvg。 不過，於硬碟啟動期間，應當儘可能趕快轉開 rootvg 並啟動分頁。雖然，在這兩個開機範例情節中存在差異，但 RAM 檔案系統的結構不會有太大的改變。

init 指令（位於 RAM 檔案系統上）是設計於開機處理程序期間使用的基本開機指令直譯器程式。 此開機指令直譯器程式透過呼叫 `rc.boot` Script 來控制開機處理程序。 `rc.boot` Script 決定從哪一個裝置啟動機器。開機裝置決定必須在 RAM 檔案系統上配置哪些裝置。如果是透過網路啟動機器，則需要配置網路裝置，才能遠端裝載用戶端檔案系統。 如果是透過磁帶、CD-ROM 或 DVD 啟動，則會配置主控台以顯示基本作業系統 (BOS) 安裝功能表。`rc.boot` Script 識別開機裝置之後，會從 RAM 檔案系統呼叫適當的配置常式。在開機處理程序期間，開機指令直譯器程式會呼叫 `rc.boot` Script 兩次，以符合兩個配置階段。於磁碟或網路開機期間，若呼叫了實際的 **init** 指令，則會第三次呼叫 `rc.boot`。`inittab` 檔案包含 `rc.boot` 段落，以完成機器的最終配置。

因為要配置各種裝置類型，所以每個開機裝置的 RAM 檔案系統亦是唯一的。原型檔案與每個類型的開機裝置相關。原型檔案是組成 RAM 檔案系統的檔案範本。**bosboot** 指令會使用 **mkfs** 指令，以便使用各種原型檔案來建立 RAM 檔案系統。請參閱 **bosboot** 指令，以取得詳細資訊。

相關概念

開機處理程序
有三種類型的系統開機，以及兩種在啟動作業系統時所需的資源。

疑難排解系統啟動

使用這些疑難排解方法，可以解決系統啟動時可能發生的一些基本問題。如果疑難排解資訊未解決您的問題，請聯絡客戶服務代表。

無法開機的系統

如果系統無法從硬碟開機，您還是能夠存取系統，以便查明並更正問題。

如果您的系統無法從硬碟開機，請參閱安裝與移轉 中的疑難排解您的安裝，以取得如何不開機而存取系統的程序。

此程序可以讓您取得系統提示，使您可以嘗試從系統回復資料，或執行更正動作，使系統能夠從硬碟開機。

註：

- 此程序僅供資深的系統管理員使用，因為這些管理員瞭解如何在無法自硬碟開機的系統中啟動或回復資料。大部分的使用者不應嘗試此程序，而應聯絡客戶服務代表。
- 剛剛完成新安裝的系統管理員不應使用此程序，因為在此情況下，系統不含回復所需的資料。如果在新的安裝完成後無法自硬碟開機，則請聯絡客戶服務代表。

相關參考

開機問題診斷
有許多因素會造成系統無法開機。

開機問題診斷

有許多因素會造成系統無法開機。

部分因素包括：

- 硬體問題
- 開機磁帶或光碟損毀
- 網路開機伺服器的配置不適當
- 檔案系統損壞
- 在 Script（如 `/sbin/rc.boot`）中發生錯誤

如果開機處理程序中止（參考碼為 2702），並顯示訊息「授與的記憶體不足」，請使用 HMC 來增加可用於該分割區的授與記憶體數量。

相關概念

無法開機的系統

如果系統無法從硬碟開機，您還是能夠存取系統，以便查明並更正問題。

系統備份

一旦您的系統在使用中，您的下一考量應是備份檔案系統、目錄及檔案。如果有備份檔案系統，就可以在萬一硬碟損毀時，還原檔案或檔案系統。有不同的方法可以備份資訊。

備份檔案系統、目錄及檔案意味著大量時間及精力的投入。同時，所有的電腦檔案都可能容易被有意或意外地變更或消除。

小心： 硬碟損毀時，會損毀硬碟上所包含的資訊。回復已摧毀資料的唯一方法，就是從您的備份檔中取出資訊。

如果您使用謹慎、有系統的方法來備份您的檔案系統，您一定能輕鬆地還原檔案或檔案系統的最新版本。

備份資訊有數種方法。最常用的方法之一稱為依名稱備份、檔名保存或一般備份。這是檔案系統、目錄或檔案的副本，保留作為檔案轉送或萬一無意間原始資料遭到變更或摧毀時備用。此種備份的方法在指定 **i** 旗標時執行，可用來製作個別檔案及目錄的備份。這是個別使用者備份其帳戶的常用方法。

另一個常用的方法稱為依 *i-node* 備份、檔案系統保存或保存備份。此種備份方法是在不指定 **i** 旗標時執行。這是用來作為未來參考、歷史用途，或在原始資料損壞或遺失時，進行復原之用。可用它來製作整個檔案系統的備份。這是系統管理者備份大型檔案群組（如 /home 中的所有使用者帳戶）的常用方法。檔案系統備份可讓漸進式備份得以輕鬆執行。漸進式備份可備份自指定的前一次備份以來已修改的所有檔案。

compress 及 **pack** 指令可讓您壓縮檔案以儲存；一旦還原了檔案，**uncompress** 及 **unpack** 指令可解壓縮這些檔案。壓縮及解壓縮檔案的處理程序會花一些時間。但壓縮後的資料在備份媒體上會佔用較少的空間。如需這些指令的相關資訊，請參閱 **compress**、**pack**、**uncompress** 及 **unpack**。

因為數個指令可建立備份及保存檔。所以，需要標記已備份的資料，指出使用哪個指令來起始備份， 及製作備份的方式（依名稱或依檔案系統）。

項目	說明
backup	依名稱或依檔案系統備份檔案。如需相關資訊，請參閱 **backup**。
mksysb	建立 rootvg 的可安裝映像檔。如需相關資訊，請參閱 **mksysb**。
cpio	將檔案複製到保存儲存體，或從其中複製檔案。如需相關資訊，請參閱 **cpio**。
dd	轉換及複製檔案。常用來將資料轉換及複製到執行其他作業系統的系統（如大型電腦），或從其中轉換及複製資料。**dd** 不會將多個檔案群組成一個保存檔；它是用來操作及移動資料。如需相關資訊，請參閱 **dd**。
tar	建立或操作 tar 格式的保存檔。如需相關資訊，請參閱 **tar**。
rdump	依檔案系統將檔案備份至遠端機器的裝置上。如需相關資訊，請參閱 **rdump**。
pax	(POSIX 符合保存檔公用程式）讀取及寫入 **tar** 及 **cpio** 保存檔。如需相關資訊，請參閱 **pax**。

相關概念

BSD 4.3 系統管理程式的備份

BSD 4.3 系統管理程式可以備份資料。

相關工作

備份使用者檔案或檔案系統

您可以使用兩種程序來備份檔案及檔案系統：SMIT 捷徑 **smit backfile** 或 **smit backfilesys**，以及 **backup** 指令。

備份概念
在開始備份資料之前，您需要瞭解資料類型、原則，以及您可以使用的媒體。

備份原則
單一備份原則無法滿足所有使用者的需求。例如，適用於供一位使用者使用之系統的原則，可能並不適合可供 100 位使用者使用的系統。同樣地，在許多檔案每日變更的系統中所發展出的策略，用於資料不常變更的系統時可能是無效率的。

無論適合您站台的備份策略為何，重要的是存在著這樣的策略，並且經常定期地進行備份。如果沒有施行良好的備份策略，一旦資料遺失就難以回復。

只有您才能判斷適合您系統的最佳備份原則，但下列一般性指導方針可能會有些幫助：

- **確定您可以自嚴重的資料遺失事件中回復**。

  系統在發生任何單一硬式磁碟故障後能夠繼續執行嗎？ 若所有硬碟都失效，您能回復您的系統嗎？ 如果您因火災或遭竊而遺失備份磁片或磁帶，您能回復您的系統嗎？ 如果資料遺失，重建資料的難度有多大？ 請考慮各種可能、甚至不可能發生的重大遺失，設計出可讓您在發生任何情況的遺失後都能夠回復系統的備份原則。

- **定期檢查您的備份**。

  備份媒體及其硬體可能不可靠。如果無法將資料重新讀取到硬式磁碟，保存的大量備份磁帶或磁片就毫無用處。為確保備份可用，應定期顯示備份磁帶的目錄（使用 **restore -T** 或針對保存磁帶使用 **tar -t**）。如果您使用磁片來備份，並且有多個軟式磁碟機，請從建立備份所用之磁碟機以外的磁碟機讀取磁片。為安全起見，您可能想要用另一組媒體來重複每一層次 0 備份。如果您使用串流的磁帶機來備份，則可用 **tapechk** 指令來執行磁帶上基本的一致性檢查。如需這些指令的相關資訊，請參閱 **restore -T**、**tar -t** 及 **tapechk**。

- **保留舊的備份**。

  開發一個規律的循環，以重覆使用您的備份媒體；但請不要重覆使用您所有的備份媒體。有時，在數月之後，您或其他某位系統使用者才會注意到一個重要的檔案損壞或遺失。為應付這樣的可能狀況，請儲存舊的備份。例如，您可能有下列三種備份磁帶或磁片的循環：

  - 每週一次，重複利用除星期五磁片外的所有每日磁片。
  - 每月一次，重複利用除每月最後一個星期五磁片外的所有星期五磁片。此動作可使先前四個週五的磁片一定可用。
  - 每季一次重新循環所有月用磁片，但除了上月的磁片。 將每季的最後一個每月磁片無限期地保存，最好是保存在另一棟建築物裡。

- **備份前先檢查檔案系統**。

  從已損壞檔案系統製作的備份可能毫無用處。製作備份前，用 **fsck** 指令檢查檔案系統的完整性是一個好原則。如需相關資訊，請參閱 **fsck**。

- **確定在備份期間檔案不在使用中**。

  製作備份時請勿使用系統。如果系統在使用中，檔案在備份時可能會變更，而備份將會不正確。

- **對系統進行重大變更之前先備份系統**。

  在執行任何硬體測試或修復工作之前，或安裝任何新裝置、程式或其他系統特性之前，備份整個系統始終是一個好原則。

- **其他因素**。

  在規劃及實施備份策略時，請考量下列幾點：

  - 資料多久變更一次？ 作業系統資料不會經常變更，所以您不需要經常備份它。另一方面，使用者資料通常會頻繁地變更，所以您應該時常備份它。
  - 系統上有多少使用者？ 使用者的數目會影響儲存媒體的數量和所需備份的頻率。
  - 重建資料有多困難？ 您必須謹慎思考，如果沒有備份的話，就無法重新建立某些資料。

具備適當的備份策略來保留您的資料是很重要的。評估站台的需要，可幫助您評定對您最有益的備份原則。您要經常並定期執行使用者資訊備份。如果沒有實施良好的備份策略，則回復資料流失會很困難。

註：備份具名管線（FIFO 特殊檔案）時，管線可能關閉也可能開啟。然而，如果對開啟的具名管線執行備份，還原會失敗。還原 FIFO 特殊檔案時，重建它所需要的僅是其 i-node，因為 i-node 包含了其全部特性資訊。具名管線的內容與還原無關。因此，備份期間，在製作好備份之前，檔案大小為零（所有 FIFO 均關閉）。

小心：系統備份及還原程序需要在類型與製作備份所在之平台相同的平台上還原系統。特別是 CPU 及 I/O 介面板的類型必須相同。

備份媒體

有許多不同的備份媒體類型可供使用。根據您的硬體及軟體，決定適用於您特定系統配置之不同類型的備份媒體。

可以使用數種類型的備份媒體。您的特定系統配置可用的備份媒體類型取決於您的軟體及硬體。最常用的類型是磁帶（8 公釐磁帶及 9 磁軌磁帶）、磁片（5.25 英吋磁片及 3.5 英吋磁片）、遠端保存檔，以及替代的本端硬碟。除非使用 **backup -f** 指令指定不同的裝置，否則，**backup** 指令會自動將其輸出寫入 `/dev/rfd0`（此為軟式磁碟機）。

小心：執行 **backup** 指令會導致遺失先前在選取的備份媒體上儲存的全部資料。

磁片

磁片是標準的備份媒體。除非使用 **backup -f** 指令指定不同的裝置，否則 **backup** 指令會自動將其輸出寫入 `/dev/rfd0` 裝置（此為軟式磁碟機）。若要將資料備份到預設的磁帶機，請鍵入 `/dev/rmt0`，然後按 Enter 鍵。

在處理磁片時，請務必小心。因為每一資訊在磁片上佔用極小的區域，所以小刮傷、灰塵、食物、煙粒都會使資訊無法使用。請記住下列：

- 請勿接觸錄製表面。
- 讓磁片避開磁鐵和磁場來源，如電話、口述機與電算機。
- 將磁片避開極熱處與極冷處。建議的溫度範圍為攝氏 10 度到攝氏 60 度（華氏 50 度到華氏 140 度）。
- 適當的維護會防止資訊的流失。
- 定期製作您磁片的備份副本。

小心：須用正確類型的軟碟機及磁片，才能順利地儲存資料。如果您在 3.5 英吋軟碟機中使用了錯誤的磁片，則可能摧毀磁片上的資料。

軟碟機使用下列 3.5 英吋磁片：

- 1 MB 容量（儲存大約 720 KB 的資料）
- 2 MB 容量（儲存大約 1.44 MB 的資料）

磁帶

由於磁帶具有高容量和持久性，所以常被用來儲存大型檔案或許多檔案，如檔案系統的保存副本。也會使用它們將許多檔案從某個系統轉送到另一個系統。但不會廣泛使用磁帶來儲存經常存取的檔案，因為其他媒體可提供更快速的存取時間。

使用指令如 **backup**、**cpio** 及 **tar** 會建立磁帶檔案，這些指令開啟磁帶機、寫入磁帶機並關閉磁帶機。

備份策略

備份大量資料的方法有兩種。

- 完整系統備份
- 漸進式備份

若要瞭解這兩種備份類型，以及哪種類型適合某個站台或系統，去瞭解檔案系統結構及資料的放置方式很重要。決定了放置資料的策略之後，就可以製定該資料的備份策略。

相關工作

實作排定的備份

此程序說明如何開發及使用 Script， 用以執行使用者檔案的每週完整備份及每日漸進式備份。

系統資料對使用者資料

資料的定義為程式或文字。在此處的討論中，將資料分成兩類：

- 系統資料，組成作業系統及其擴充。此資料一律保存於系統檔案系統中（即 /（根）、 /usr、/tmp、/var 等等）。
- 使用者資料通常是個人為完成其特定作業所需的本端資料。這種資料保存在 /home 檔案系統或特別為使用者資料建立的檔案系統中。

不會將使用者程式及文字置於設計用來包含系統資料的檔案系統中。例如，系統管理程式可能會建立一個新的檔案系統，並將其裝載到 /local 上。/tmp 是個例外，其用於暫時儲存系統及使用者資料。

備份

一般而言，會保留使用者及系統資料的備份，以防不小心移除資料或磁碟發生故障。使用者資料與系統資料分別保存可方便管理備份。

以下是將系統資料與使用者資料分別保存的理由：

- 使用者資料的變更往往比作業系統資料的變更頻繁許多。如果不將系統資料與使用者資料備份到同一備份映像檔，則映像檔會小得多。使用者的數目會影響備份所需的儲存媒體及頻率。
- 分別保存的作法能夠更快速、輕鬆地還原使用者資料。將作業系統與使用者資料一起還原需要額外的時間及精力。這是因為用來回復作業系統資料的方法，牽涉到從抽取式媒體（磁帶或 CD）啟動系統及安裝系統備份。

若要備份系統資料，請使用 **umount** 指令卸載所有的使用者檔案系統，包括 /home 在內。若這些檔案系統在使用中，就無法卸載它們。請將備份時間排定在使用率較低的時候，以便卸載它們；如果使用者資料檔案系統仍維持裝載中，會將它們與作業系統資料一起備份。 使用 **mount** 指令以確保僅裝載作業系統檔案系統。

裝載的檔案系統只有 /、/usr、/var 及 /tmp，而且 **mount** 指令的結果可能類似於下列輸出：

```
節點   裝載          裝載到    vfs       日期          選項
             /dev/hd4     /        jfs   Jun 11 10:36  rw,log=/dev/hd8
             /dev/hd2     /usr     jfs   Jun 11 10:36  rw,log=/dev/hd8
             /dev/hd9var  /var     jfs   Jun 11 10:36  rw,log=/dev/hd8
             /dev/hd      /tmp     jfs   Jun 11 10:36  rw,log=/dev/hd8
```

在確定已卸載所有使用者檔案系統之後，您現在已準備好要備份作業系統資料。

完成作業系統的備份後，請使用 **smit mount** 指令來裝載使用者檔案系統。然後可依您的需要備份檔案、檔案系統或其他磁區群組。

相關概念

系統映像檔及使用者定義的磁區群組備份

rootvg 儲存在硬碟或磁碟群組上，包含啟動檔案、BOS、配置資訊及任何選用性軟體產品。 使用者定義的磁區群組（亦稱作非 *rootvg* 磁區群組）通常包含資料檔及應用軟體。

系統抄寫（複製）

複製作業會將配置資料連同使用者或系統資料一起儲存。例如，您可能想要抄寫系統或磁區群組；有時候它稱為複製作業。

然後您可以將此映像檔安裝到另一個系統上，並可以像在第一個系統上一樣使用它。**mksysb** 指令是用來複製 rootvg 磁區群組，該群組包含作業系統；而 **savevg** 指令是用來複製磁區群組。

備份檔案及儲存媒體的指令摘要

這些指令可用於備份檔案及儲存資料。

項目	說明
backup	備份檔案與檔案系統
compress	壓縮與展開資料
cpio	保存儲存體與目錄中的檔案複製
fdformat	格式化磁片
flcopy	磁片複製
format	格式化磁片
fsck	檢查檔案系統一致性與互動地修復檔案系統
pack	壓縮檔案
restore	從本端裝置複製先前由 **backup** 指令來建立的備份檔案系統或檔案
tapechk	檢查串流磁帶機的一致性
tar	操作保存檔
tcopy	複製磁帶
uncompress	壓縮與展開資料
unpack	展開檔案

管理系統備份

有多種方法可以備份您的系統及還原系統備份。

備份使用者檔案或檔案系統

您可以使用兩種程序來備份檔案及檔案系統：SMIT 捷徑 **smit backfile** 或 **smit backfilesys**，以及 **backup** 指令。

- 如果您要備份正在使用中的 i-node 檔案系統，請先卸載它們以防止發生不一致的情況。

小心： 如果您嘗試備份裝載的檔案系統，就會顯示警告訊息。**backup** 指令雖會繼續執行，但可能在檔案系統中發生不一致的情況。此警告不適用於 root (/) 檔案系統。

- 若要防止發生錯誤，請確定備份裝置最近已經清除過。

若要備份使用者檔案及檔案系統，您可以使用 SMIT 捷徑 **smit backfile** 或 **smit backfilesys**。

可使用 SMIT 介面，依名稱備份單一及小型檔案系統（如本端系統上的 /home）。請注意，SMIT 無法以 **backup** 指令所提供之格式以外的其他格式來製作保存檔。而且，透過 SMIT 並不能使用 **backup** 指令的每個旗標。備份期間如果需要多個磁帶或磁碟，SMIT 可能會當掉。如需相關資訊，請參閱 *Commands Reference, Volume 1* 中的 **backup** 指令說明。

當您想要備份大型及多個檔案系統時可使用 **backup** 指令。可指定層次號碼，以控制備份的資料量（完整為 0；遞增為 1-9）。只有使用 **backup** 指令，才能在備份時指定層次號碼。

backup 指令會以下列兩種備份格式之其中一種來建立副本：

- 使用 **-i** 旗標依名稱備份的特定檔案。
- 使用 **-Level** 及 **FileSystem** 參數依 i-node 備份的整個檔案系統。 當從備份還原檔案系統時，會進行磁碟重整。

小心： 對於其使用者 ID (UID) 或群組 ID (GID) 大於 65535 的檔案，無法依 i-node 進行正確備份。依 i-node 備份這些檔案時，會截斷其 UID 或 GID，因此，在還原時它們的 UID 或 GID 屬性是錯誤的。對於這些情況，必須依名稱備份。

備份使用者檔案或檔案系統作業		
作業	***SMIT* 捷徑**	**指令或檔案**
備份使用者檔案	`smit backfile`	1. 登入您的使用者帳戶。 2. 備份：`find . -print | backup -ivf /dev/rmt0`
備份使用者檔案系統	`smit backfilesys`	1. 卸載您預計備份的檔案系統。例如：`umount all` 或 `umount /home /filesys1` 2. 驗證檔案系統。例如：`fsck /home /filesys1` 3. 以 i-node 備份。例如：`backup -5 -uf/dev/rmt0/home/libr` 4. 使用下列指令還原檔案：**restore -t**

註： 如果此指令產生錯誤訊息，您必須重複整個備份。

相關概念

系統備份

一旦您的系統在使用中，您的下一考量應是備份檔案系統、目錄及檔案。如果有備份檔案系統，就可以在萬一硬碟損毀時，還原檔案或檔案系統。有不同的方法可以備份資訊。

還原已備份的檔案

正確備份資料後，根據所使用的備份指令類型，有數種不同的方法可以還原資料。

為正確還原，您需要瞭解備份或保存檔是如何建立的。每個備份程序均提供有還原資料的相關資訊。 例如，若使用 **backup** 指令，可指定依檔案系統或依名稱的備份。還原該備份的方式必須與建立備份的方式（依檔案系統或依名稱）相同。如需 **backup** 指令的相關資訊，請參閱 **backup**。

有數個指令可還原備份的資料，例如：

項目	**說明**
restore	複製由 **backup** 指令建立的檔案。如需使用此指令的相關資訊，請參閱下節。
rrestore	將遠端機器上備份的檔案系統複製到本端機器上。如需相關資訊，請參閱 **rrestore**。
cpio	將檔案複製到保存儲存體，或從其中複製檔案。如需相關資訊，請參閱 **cpio**。
tar	建立或操作 **tar** 保存檔。如需相關資訊，請參閱 **tar**。
pax	（POSIX 符合保存檔公用程式）讀取及寫入 **tar** 及 **cpio** 保存檔。如需相關資訊，請參閱 **pax**。

下列幾節討論 **restore** 及 **smit** 指令。

註：

- 檔案的還原方式必須與其備份時的方法相同。例如，如果以名稱備份檔案系統，則也必須以名稱還原它。
- 當需要一片以上的磁片時，**restore** 指令會先讀取所裝載的磁片， 然後提示您插入新磁片，並等待您的回應。插入新的磁片之後，請按 Enter 鍵，繼續還原檔案。

使用 restore 指令還原檔案

使用 **restore** 指令來讀取由 **backup** 指令所撰寫的檔案，並在您的本端系統上還原它們。

請參閱下列範例：

· 若要列出先前已備份之檔案名稱，請鍵入：

```
restore -T
```

從 /dev/rfd0 預設備份裝置讀取資訊。如果備份個別檔案，則僅檔名被顯示。如果備份整個檔案系統，則 i-node 號碼也會顯示。

· 若要還原檔案成主檔案系統，請鍵入：

```
restore -x -v
```

-x 旗標會從備份媒體擷取所有檔案，然後將它們還原到檔案系統中的適當位置。此 **-v** 旗標顯示還原每一個檔案時的進度報告。如果檔案系統備份已還原，則檔案會以它們的 i-node 號碼命名。否則就只會顯示名稱。

· 若要複製 /home/mike/manual/chap1 檔案，請鍵入：

```
restore -xv /home/mike/manual/chap1
```

此指令從備份媒體擷取 /home/mike/manual/chap1 檔案，然後還原它。/home/mike/manual/chap1 檔案必須是 **restore -T** 指令可以顯示的名稱。

· 若要複製目錄 manual 中的所有檔案，請鍵入：

```
restore -xdv manual
```

此指令會將 manual 目錄及其中的檔案還原。如果目錄不存在，就會在現行目錄中建立 manual 目錄，以保存所還原的檔案。

請參閱 *Commands Reference, Volume 4* 中的 **restore** 指令，以取得完整語法。

使用 smit 指令還原檔案

使用 **smit** 指令來執行 **restore** 指令， 這會讀取由 **backup** 指令所撰寫的讀案，並在您的本端系統上還原它們。

1. 在提示時鍵入：

   ```
   smit restore
   ```

2. 在 **Target DIRECTORY** 欄位中輸入您的項目。此為您要存放還原檔案的目錄。
3. 繼續進行 **BACKUP 裝置**或 **FILE** 欄位，然後輸入輸出裝置名稱，如下列原始資料磁帶機範例所示：

   ```
   /dev/rmt0
   ```

   如果裝置無法使用，則會出現如下的訊息：

   ```
   無法開啟 /dev/rmtX，無此檔案或目錄。
   ```

   此訊息指出系統無法取得裝置驅動程式，因為 /dev 目錄中沒有 **rmtX** 的檔案存在。只有處於 available 狀態的項目，才會出現在 /dev 目錄中。

4. 就 **NUMBER of blocks to read in a single input** 欄位而言，建議您使用預設值。
5. 按 Enter 鍵，以還原指定的檔案系統或目錄。

建立遠端保存檔

使用這個程序，可以將檔案保存至遠端磁帶機。

執行 AIX 系統時，無法像在系統本端裝載磁帶機一樣，將遠端磁帶機裝載至系統中；不過，您可使用 **rsh** 指令將資料傳送至遠端機器的磁帶機。下列程序僅會寫入單一磁帶。多磁帶保存檔需要特殊應用軟體才能執行。

下列程序採用下列內容：

blocksize
代表目標磁帶機區塊的大小。

remotehost
為目標系統（裝載磁帶機的系統）名稱。

sourcehost
為來源系統（正在進行保存的系統）名稱。

/dev/rmt0
為遠端磁帶機的名稱。

pathname
代表必要目錄或檔案的完整路徑名稱。

使用下列指示時，假設本端及遠端使用者都是 root。

1. 請確定您擁有對遠端機器的存取權。
   來源機器必須對裝載磁帶機的系統具有存取權限。（可使用該系統上任一已定義使用者來存取目標系統，但使用者名稱必須具有 root 權限才能執行下列許多步驟。）
2. 使用您喜好的編輯器，在目標系統的 /（根）目錄中建立名為 `.rhosts` 的檔案，使來源系統能夠存取目標系統。
   您需要將授權的主機名稱及使用者 ID 新增至此檔案。若要判斷 `.rhosts` 檔案的來源機器名稱，您可以使用下列指令：
   ```
   host SourceIPaddress
   ```
   為了配合此範例，假設您將下一行新增至 `.rhosts` 檔案：
   ```
   sourcehost.mynet.com   root
   ```
3. 儲存檔案，然後使用下列指令變更其許可權：
   ```
   chmod 600 .rhosts
   ```
4. 使用 **rsh** 指令在來源機器中測試您的存取權限。例如：
   ```
   rsh remotehost
   ```
   如果一切都已設定正確，則應授與您對遠端機器的 shell 存取權。您應該看不到詢問使用者名稱的登入提示。請鍵入 `exit` 登出此測試 shell。
5. 判斷適當的磁帶機區塊大小。
   下列是建議值：

項目	說明
9-磁軌或 0.25 吋 媒體區塊大小：	512
8-mm 或 4-mm 媒體區塊大小：	1024

   如果您不確定並想要檢查磁帶機的現行區塊大小，請使用 **tctl** 指令。例如：
   ```
   tctl -f /dev/rmt0 status
   ```
   如果您要變更磁帶區塊大小，請使用 **chdev** 指令。例如：
   ```
   chdev -l rmt0 -a block_size=1024
   ```
6. 使用下列方法之一來建立您的保存檔：

依名稱備份

若要從遠端建立依名稱備份的保存檔，請使用下列指令：

```
find pathname -print | backup -ivqf- | rsh
remotehost \
         "dd of=/dev/rmt0 bs=blocksize conv=sync"
```

依 inode 備份

若要從遠端建立依 inode 備份的保存檔，請先卸載您的檔案系統，然後使用 **backup** 指令。例如：

```
umount /myfs
backup -0 -uf- /myfs | rsh remotehost \
          "dd of=/dev/rmt0 bs=blocksize conv=sync"
```

建立及複製保存檔到遠端磁帶

若要建立及複製保存檔到遠端磁帶機，請使用下列指令：

```
find pathname -print | cpio -ovcB | rsh remotehost \
    "dd ibs=5120 obs=blocksize of=/dev/rmt0"
```

建立 tar 保存檔

若要從遠端建立 **tar** 保存檔，請使用下列指令：

```
tar -cvdf- pathname | rsh remotehost \
         "dd of=/dev/rmt0 bs=blocksize conv=sync"
```

建立遠端傾出

若要從遠端建立 /myfs 檔案系統的遠端傾出，請使用下列指令：

```
rdump -u -0 -f remotehost:/dev/rmt0 /myfs
```

-u 旗標可告知系統去更新 /etc/dumpdates 檔案中的現行備份層次記錄。**-0** 是 *Level* 旗標的設定。備份層次 0 則是指定備份 /myfs 目錄中的所有檔案。如需相關資訊，請參閱 *Commands Reference, Volume 4* 中的 **rdump** 指令說明。

7. 使用下列方法之一來還原您的遠端保存檔：

依名稱還原備份

若要依名稱還原遠端備份保存檔，請使用下列指令：

```
rsh remotehost "dd if=/dev/rmt0 bs=
blocksize" | restore \
     -xvqdf- pathname
```

依 inode 還原備份

若要依 inode 還原遠端備份保存檔，請使用下列指令：

```
rsh remotehost "dd if=/dev/rmt0 bs=
blocksize" | restore \
    -xvqf- pathname
```

還原遠端 cpio 保存檔

若要還原使用 **cpio** 指令建立的遠端保存檔，請使用下列指令：

```
rsh remotehost "dd if=/dev/rmt0 ibs=blocksize obs=5120" | \
     cpio -icvdumB
```

還原 tar 保存檔

若要還原遠端 **tar** 保存檔，請使用下列指令：

```
rsh remotehost "dd if=/dev/rmt0 bs=blocksize" | tar -xvpf- pathname
```

還原遠端傾出

若要還原 /myfs 檔案系統的遠端傾出，請使用下列指令：

```
cd /myfs
rrestore -rvf remotehost:/dev/rmt0
```

從備份映像檔中還原使用者檔案

如果您需要還原因意外而損毀的備份映像檔， 最困難的問題是判斷哪一個備份磁帶含有此檔案。**restore -T** 指令可以用來列出保存檔內容。在 /tmp 目錄中還原檔案是個好方法，這樣就不會不小心改寫使用者的其他檔案。

請確定裝置已連接且可供使用。若要檢查可用性，請鍵入：

```
lsdev -C | pg
```

如果備份策略包含漸進式備份，則從使用者找出檔案最近一次修改的時間是非常有用的。這有助於判斷哪一個漸進式備份含有此檔案。如果無法取得此資訊或找到的資訊不正確，則以反向次序 (7, 6, 5, ...) 開始搜尋漸進式備份。若為遞增式檔案系統備份，則 **restore** 指令的 **-i** 旗標（互動模式），在尋找與還原失去的檔案時非常有用 （從 /home 檔案系統的備份中還原個別使用者的帳戶時，互動模式也非常有用。）

下表中的程序說明如何實行層次 0（完整）還原目錄或檔案系統。

從備份映像檔中還原作業		
作業	*SMIT* 捷徑	指令或檔案
還原個別的使用者檔案	**smit restfile**	請參閱 **restore** 指令。
還原使用者檔案系統	**smit restfilesys**	1. **mkfs /dev/hd1** 2. **mount /dev/hd1 /filesys** 3. **cd /filesys** 4. **restore -r**
還原使用者磁區群組	**smit restvg**	請參閱 **restvg -q** 指令。

還原對未鏈結或已刪除的系統檔案庫之存取權

如果無法使用現有的 **libc.a** 程式庫，則無法辨識大部分的作業系統 (OS) 指令。

這種問題的最可能原因如下：

- /usr/lib 中的鏈結不再存在。
- /usr/ccs/lib 中的檔案已刪除。

下列程序說明如何還原 **libc.a** 程式庫的存取權。下列程序需要系統停機。如有可能，請將停機時間排定在對工作量影響最小的期間，以免發生資料或功能遺失。

此作法範例情節中的資訊已使用特定版本的 AIX 進行測試。依據您的 AIX 版本及層次，所得到的結果可能大不相同。

相關資訊

mount 指令
unmount 指令
reboot 指令

還原已刪除的符號鏈結

使用下列程序，可以將符號鏈結從 /usr/lib/libc.a 程式庫還原回 /usr/ccs/lib/libc.a 路徑。

此作法範例情節中的資訊已使用特定版本的 AIX 進行測試。依據您的 AIX 版本及層次，所得到的結果可能大不相同。

1. 以 root 權限鍵入下列指令，將 **LIBPATH** 環境變數設定為指向 /usr/ccs/lib 目錄：

```
# LIBPATH=/usr/ccs/lib:/usr/lib
# export LIBPATH
```

   此時，您應該能夠執行系統指令。
2. 若要將鏈結從 /usr/lib/libc.a 程式庫及 /lib 目錄還原到 /usr/lib 目錄，請鍵入下列指令：

```
ln -s /usr/ccs/lib/libc.a /usr/lib/libc.a
ln -s /usr/lib /lib
```

   此時，指令應該能像以前一樣執行。如果您還是無法存取 shell，請略過此程序的剩餘部分，並繼續下一個段落：第 27 頁的『還原已刪除的系統程式庫檔案』。
3. 請鍵入下列指令，取消設定 LIBPATH 環境變數。

```
unset LIBPATH
```

還原已刪除的系統程式庫檔案

此還原已刪除之系統程式庫檔案程序，需要將系統停機。待系統開機時，就會從最新的 **mksysb** 磁帶還原程式庫。

1. 重新開機之前，請確定 bosinst.data 檔案中的 **PROMPT** 欄位是設為 yes。
2. 將最新的 **mksysb** 磁帶插入磁帶機。

   mksysb 必須包含和已安裝的系統相同的 OS 及維護套件或技術層次。如果從 **mksysb** 還原的 libc.a 程式庫和已安裝系統上的層次發生衝突，您將無法發出指令。
3. 將機器重新開機。
4. 當出現圖示畫面時，或當您聽到兩聲嗶聲時，請重複按 F1 鍵，直到顯示出系統管理服務功能表為止。
5. 選取**多重開機**。
6. 選取**安裝來源**。
7. 選取容納 **mksysb** 的磁帶機，然後選取**安裝**。

   可能需要數分鐘才會出現下一個提示。
8. 按 F1 鍵然後按 Enter 鍵，將您的現行系統定義為系統主控台。
9. 選取您偏好的語言編號，然後按 Enter 鍵。
10. 鍵入 3 並且按 Enter 鍵，以選取**啟動維護模式進行系統復原**。
11. 選取**存取 Root 磁區群組**。這時會顯示一則訊息， 說明如果您此時變更 root 磁區群組，則在沒有重新開機的情況下，您將無法返回「安裝」功能表。
12. 鍵入 0，然後按 Enter 鍵。
13. 從清單鍵入適當的磁區群組號碼，然後按 Enter 鍵。
14. 鍵入 **2**，並且按 Enter 鍵，以選取存取此磁區群組。
15. 鍵入下列指令，裝載 / (root) 及 /usr 檔案系統：

```
mount /dev/hd4 /mnt
mount /dev/hd2 /mnt/usr
cd /mnt
```

16. 若有需要還原 libc.a 程式庫的符號鏈結，請鍵入下列指令：

```
ln -s /usr/ccs/lib/libc.a /mnt/usr/lib/libc.a
```

    執行指令之後，請執行下列一項：

    · 如果指令順利完成，請跳至步驟 20。

    · 如果出現訊息顯示鏈結已存在，請繼續步驟 17。
17. 發出下列指令（其中 *X* 是適當的磁帶機編號），設定磁帶機的區塊大小：

```
tctl -f /dev/rmtX rewind
tctl -f /dev/rmtX.1 fsf 1
```

```
restbyname -xvqf /dev/rmtX.1 ./tapeblksz
cat tapeblksz
```

如果 **cat tapeblksz** 指令的值不等於 512，請鍵入下列指令，以 **cat tapeblksz** 指令的值取代 *Y*：

```
ln -sf /mnt/usr/lib/methods /etc/methods
/etc/methods/chgdevn -l rmtX -a block_size=Y
```

您應該會收到訊息，指出已經變更 rmt*X*。

18. 鍵入下列指令（其中 *X* 是適當的磁帶機編號），確定磁帶位於正確位置以還原程式庫：

```
tctl -f /dev/rmtX rewind
tctl -f /dev/rmtX.1 fsf 3
```

19. 使用下列指令之一（其中 *X* 是適當的磁帶機編號），還原缺少的程式庫：

    - 若只要還原 libc.a 程式庫，請鍵入下列指令：

      ```
      restbyname -xvqf /dev/rmtX.1 ./usr/ccs/lib/libc.a
      ```

    - 若要還原 /usr/ccs/lib 目錄，請鍵入下列指令：

      ```
      restbyname -xvqf /dev/rmtX.1 ./usr/ccs/lib
      ```

    - 若要還原 /usr/ccs/bin 目錄，請鍵入下列指令：

      ```
      restbyname -xvqf /dev/rmtX.1 ./usr/ccs/bin
      ```

20. 鍵入下列指令，將資料寫入磁碟：

```
cd /mnt/usr/sbin
./sync;./sync;./sync
```

21. 鍵入下列指令，卸載 /usr 及 / (root) 檔案系統：

```
cd /
umount /dev/hd2
umount /dev/hd4
```

如果有任一 **umount** 指令失敗，請循環開啟此機器的電源並再次開始此程序。

22. 鍵入下列指令，將系統重新開機：

```
reboot
```

系統重新開機之後，就應該可以使用作業系統 (OS) 指令。

重建損毀的開機映像檔

下面的程序說明如何識別損毀的開機映像檔及如何重建開機映像檔。

如果您的機器正在運作，且您知道開機映像檔已損毀或刪除，請以 root 權限執行 **bosboot** 指令來重建該開機映像檔。

小心：如果您懷疑開機映像檔已損毀，千萬不要重新啟動系統。

下面的程序假設您的系統因開機映像檔損毀而未正確重新啟動。如果可能，請將停機時間排在對工作量影響最小的期間，以保護系統，防止可能發生的資料或功能遺失。

此作法範例情節中的資訊已使用特定版本的 AIX 進行測試。依據您的 AIX 版本及層次，所得到的結果可能大不相同。

1. 將產品媒體插入適當的磁碟機。
2. 遵循系統隨附的指示開啟機器電源。
3. 從**系統管理服務**功能表，選取**多重開機**。
4. 從下一個畫面，選取**安裝來源**。

5. 選取容納產品媒體的裝置，然後選取**安裝**。
6. 請選取 AIX 版本圖示。
7. 遵循線上指示，直到您可以選取用於安裝的模式為止。屆時，請選取**啟動維護模式以進行系統回復**。
8. 選取**存取 Root 磁區群組**。
9. 遵循線上指示，直到您可以選取**存取此磁區群組並啟動 shell** 為止。
10. 使用 **bosboot** 指令，以重建開機映像檔。例如：

    ```
    bosboot -a -d /dev/hdisk0
    ```

    如果指令失敗且您收到下面的訊息：

    ```
    0301-165 bosboot：警告！bosboot 失敗 - 請不要嘗試啟動裝置。
    ```

    嘗試使用下列其中一個選項來解決問題，然後重新執行 **bosboot** 指令，直到您順利建立開機映像檔為止：

    · 刪除預設的開機邏輯磁區 (hd5)，然後建立一個新的 hd5。

    或

    · 在硬碟上執行診斷程式。依需要修復或更換。

    如果 **bosboot** 指令繼續失敗，請聯絡您的客戶支援代表。

    小心： 如果 **bosboot** 指令在建立開機映像檔時失敗，請不要重新啟動機器。
11. 當 **bosboot** 指令順利完成時，請使用 **reboot** 指令重新啟動系統。

相關概念
系統啟動
當基本作業系統啟動時，系統會起始一系列複雜作業。在正常狀況下，會自動執行這些作業。

相關資訊
bosboot 指令

製作 *JFS* 的連線備份

製作已裝載的日誌型檔案系統 (JFS) 或已增強的日誌型檔案系統 (JFS2) 的連線備份，會建立內含該檔案系統的邏輯磁區靜態映像檔。

若要製作已裝載 JFS 的連線備份，必須境映到檔案系統及其日誌所在的邏輯磁區。

註： 因為檔案寫入是非同步的，所以分割的副本可能並未包含所有執行分割前剛剛寫入的資料。在分割開始後才開始的任何修改可能不會出現在備份中。因此在執行分割時，建議檔案系統活動應減至最低。

此作法範例情節中的資訊已使用特定版本的 AIX 進行測試。依據您的 AIX 版本及層次，所得到的結果可能大不相同。

若要將 /home/xyz 檔案系統的鏡映副本分割到名為 /jfsstaticcopy 的新裝載點，請鍵入下列指令：

```
chfs -a splitcopy=/jfsstaticcopy /home/xyz
```

您可以使用 **copy** 屬性來控制要使用哪個鏡映副本作為備份。如果使用者沒有指定副本，則預設值為第二個鏡映副本。例如：

```
chfs -a splitcopy=/jfsstaticcopy -a copy=1 /home/xyz
```

此時，檔案系統的唯讀副本即可在 /jfsstaticcopy 中找到。在副本分割後對原始檔案系統所做的所有變更都不會反映在此備份中。

若要在 /testcopy 裝載點將 JFS 分割映像檔重新整合成鏡映副本，請使用下列指令：

```
rmfs /testcopy
```

rmfs 指令會從分割狀態中移除檔案系統副本，並可讓它重新整合成鏡映副本。

製作及備份 *JFS2* 的 *Snapshot*

您可以製作已裝載 JFS2 的 Snapshot，它會在某個時間點建立檔案系統的一致區塊層次映像檔。

此作法範例情節中的資訊已使用特定版本的 AIX 進行測試。依據您的 AIX 版本及層次，所得到的結果可能大不相同。

即使在用來建立 Snapshot 的檔案系統（稱為 *snappedFS*）持續變更時，Snapshot 映像檔仍會維持穩定的狀態。Snapshot 會保留製作 Snapshot 時，*snappedFS* 擁有的安全許可權。

在下列範例情節中，只需使用一個指令 **backsnap**，即可在不卸載或停止檔案系統的情形下建立 Snapshot，並將該 Snapshot 備份到抽取式媒體。Snapshot 還有其他用途，如依照檔案或目錄在製作 Snapshot 時的原樣來存取它們。您可以使用 SMIT 或 **backsnap** 及 **snapshot** 指令，執行各種 Snapshot 程序。

若要建立 /home/abc/test 檔案系統的 Snapshot，並將它備份（依名稱）到磁帶機 /dev/rmt0，請使用下列指令：

```
backsnap -m /tmp/snapshot -s size=16M -i f/dev/rmt0 /home/abc/test
```

此指令會為 JFS2 檔案系統 (/home/abc/test) 的 Snapshot 建立 16 MB 的邏輯磁區。Snapshot 會裝載在 /tmp/snapshot 上，然後根據名稱，將 Snapshot 備份到磁帶機。備份完成之後，Snapshot 仍然處於裝載狀態。如果您要在備份完成時移除 Snapshot，請在 **backsnap** 指令中使用 **-R** 旗標。

相關資訊

檔案系統
backsnap 指令
chfs 指令
rmfs 指令
snapshot 指令

製作及備份 *JFS2* 的外部 *Snapshot*

您可以製作已裝載 JFS2 的 Snapshot，它會在某個時間點建立檔案系統的一致區塊層次映像檔。

即使在用來建立 Snapshot 的檔案系統（稱為 *snappedFS*）持續變更時，Snapshot 映像檔仍會維持穩定的狀態。Snapshot 會保留製作 Snapshot 時，*snappedFS* 擁有的安全許可權。

在下列實務範例中，將使用 **backsnap** 指令，在不卸載或靜止檔案系統的情況下，建立一個外部 Snapshot 並將該 Snapshot 備份到抽取式媒體。Snapshot 還有其他用途，如依照檔案或目錄在製作 Snapshot 時的原樣來存取它們。您可以使用 SMIT 或 **backsnap** 及 **snapshot** 指令，執行各種 Snapshot 程序。

若要建立 /home/abc/test 檔案系統的外部 Snapshot，並根據名稱將它備份到 /dev/rmt0 磁帶機，請執行下列指令：

```
backsnap -m /tmp/snapshot -s size=16M -if/dev/rmt0 /home/abc/test
```

前一個指令會為 /home/abc/test JFS2 檔案系統的 Snapshot 建立 16 MB 的邏輯磁區。Snapshot 會裝載在 /tmp/snapshot 目錄上，然後根據名稱，將 Snapshot 備份到磁帶機。完成備份之後，會卸載 Snapshot，但是仍可用。如果您要在備份完成時移除 Snapshot，請在 **backsnap** 指令中使用 -R 旗標。

相關資訊

檔案系統

製作及備份 *JFS2* 的內部 *Snapshot*

您可以製作已裝載 JFS2 的 Snapshot，它會在某個時間點建立檔案系統的一致區塊層次映像檔。

即使在用來建立 Snapshot 的檔案系統（稱為 *snappedFS*）持續變更時，Snapshot 映像檔仍會維持穩定的狀態。Snapshot 會保留製作 Snapshot 時，*snappedFS* 擁有的安全許可權。

在下列實務範例中，將使用 **backsnap** 指令，在不卸載或靜止檔案系統的情況下，建立一個內部 Snapshot 並將該 Snapshot 備份到抽取式媒體。Snapshot 還有其他用途，如依照檔案或目錄在製作 Snapshot 時的原樣來存取它們。您可以使用 SMIT 或 **backsnap** 及 **snapshot** 指令，執行各種 Snapshot 程序。

若要建立 /home/abc/test 檔案系統的內部 Snapshot，並根據名稱將它備份到 /dev/rmt0 磁帶機，請執行下列指令：

```
backsnap -n mysnapshot -if/dev/rmt0 /home/abc/test
```

前一個指令會建立 /home/abc/test 檔案系統之名為 mysnapshot 的內部 Snapshot。從 /home/abc/test/.snapshot/mysnapshot 目錄存取 Snapshot，然後將其備份到磁帶機。如果您要在備份完成之後移除 Snapshot，請在 **backsnap** 指令中使用 -R 旗標。

相關資訊
檔案系統

壓縮檔案（*compress* 及 *pack* 指令）
使用 **compress** 指令及 **pack** 指令來壓縮檔案以儲存。

使用 **uncompress** 指令及 **unpack** 指令來展開已還原的檔案。

壓縮與展開檔案的處理程序耗費時間；但是，將檔案壓縮之後，資料在備份媒體上所佔的空間就會減少。

若要壓縮檔案系統，請使用下列其中一種方法：

- 同時使用 **-p** 旗標與 **backup** 指令。
- 使用 **compress** 或 **pack** 指令。

壓縮檔案的優點包括：

- 在透過網路傳送之前，先壓縮檔案，可節約金錢與時間。
- 節省儲存體並保存系統資源：
  - 製作備份之前先壓縮檔案系統，可節省磁帶空間。
  - 壓縮由在夜間執行的 shell Script 所建立的日誌檔；讓 Script 在結束之前壓縮檔案是很容易的。
  - 壓縮目前未被存取的檔案。例如，可以壓縮屬於已離職使用者的檔案，並將它置於磁碟或磁帶的 **tar** 檔案中，於日後還原使用。

註：

- **compress** 指令在執行壓縮時，可能會用盡檔案系統中的空間。該指令會在刪除任何未壓縮的檔案之前，先建立壓縮檔，所以它需要比檔案總大小還要大 50% 的空間。
- 檔案可能會因為已壓縮，而無法再壓縮。如果 **compress** 指令無法降低檔案大小，則指令失敗。

請參閱 **compress** 指令，以取得回覆值的相關明細， 但通常，壓縮檔案時所發生的問題可以彙總如下：

- 壓縮時，該指令可能用盡了檔案系統中的工作空間。因為 **compress** 指令會在刪除任何解壓縮檔之前，先建立壓縮檔，因此需要檔案 50% 到 100% 大小的額外空間。
- 檔案可能會因為已壓縮，而無法再壓縮。如果 **compress** 指令無法減少檔案大小，表示指令失敗。

使用 *compress* 指令來壓縮檔案
使用 **compress** 指令來減少採用適應 Lempel-Zev 編碼方式的檔案之大小。

由 *File* 參數指定的每一個原始檔案， 將會轉換成檔名附加 .Z 的壓縮檔。壓縮檔仍將保留原始檔案相同的所有權、模式、與存取及修改時間。若未指定檔案，標準輸入將壓縮至標準輸出。如果壓縮無法降低檔案大小，將寫入一訊息至標準錯誤，且原始檔案不會被置換。

使用 **uncompress** 指令，將壓縮檔還原為它們的原始格式。

壓縮數量將視輸入資料大小、由 *Bits* 變數所指定的每一程式碼位元數目、與共同子字串配送而定。 一般而言，原始程式或英文文字可降低至 50% 至 60%。**compress** 指令的壓縮方式， 通常會比使用適應 Huffman 編碼方式的 **pack** 指令更為緊密，並且會使用較少的時間來進行計算。

例如，若要壓縮 foo 檔案並將百分比壓縮寫入標準錯誤，請鍵入：

```
compress -v foo
```

請參閱 *Commands Reference, Volume 1* 中的 **compress** 指令，以取得完整語法。

使用 *pack* 指令來壓縮檔案
使用 **pack** 指令，可以採用 Huffman 編碼方式的壓縮格式，來儲存 *File* 參數所指定的檔案。

輸入檔將被置換為壓縮檔案，其名稱衍生自原始檔案 (*File*.z)，其存取模式、存取與修改日期、 及擁有者均與原始檔案相同。輸入檔不能超過 253 個位元組，以便容納新增的 .z 字尾。如果 **pack** 指令順利完成，原始檔案將會刪除。

使用 **unpack** 指令，將壓縮檔還原為它們的原始格式。

如果 **pack** 指令無法建立較小的檔案，將會停止處理程序，並報告無法節約空間。（無法節約空間，一般而言是發生在一致字元分佈的小檔案上。）節約的空間數量是根據輸入檔大小、與字元頻率份佈。由於解碼樹形成每一 .z 檔案的第一個部分，如果檔案小於此 3 個區塊，您將無法節約空間。一般而言，文字檔將可降低 25% 至 40%。

pack 指令的結束值為無法壓縮的檔案數量。在下列任何狀況下，壓縮無法完成：

- 檔案已壓縮。
- 輸入檔名稱超過 253 個位元組。
- 檔案含有鏈結。
- 檔案是目錄。
- 檔案無法開啟。
- 壓縮作業沒有節省任何儲存體區塊。
- 名稱為 *File*.z 的檔案已存在。
- 無法建立 .z 檔案。
- 處理時發生 I/O 錯誤。

例如，若要壓縮檔案 chap1 和 chap2，請鍵入：

```
pack chap1 chap2
```

這會壓縮 chap1 及 chap2，然後以 chap1.z 及 chap2.z 的檔名來取代它們。**pack** 指令會顯示每一個檔案大小減少的百分比。

請參閱 *Commands Reference, Volume 4* 中的 **pack** 指令，以取得完整語法。

將壓縮檔解壓縮（*uncompress* 及 *unpack* 指令）
使用 **uncompress** 及 **unpack** 指令，將壓縮檔解壓縮。

使用 uncompress 指令將檔案解壓縮

使用 **uncompress** 指令來還原由 **compress** 指令所壓縮的原始檔案。由 *File* 變數所指定的壓縮檔，將被移除並置換為展開後的副本。解壓縮後的檔案名稱與壓縮過的版本名稱相同， 但沒有 .Z 的副檔名。展開後的檔案仍保留原始檔案相同的所有權、模式與存取權，以及修改時間。若未指定檔案，標準輸入將展開至標準輸出。

雖然和 **uncompress** 指令很像， 但 **zcat** 指令一定會將展開後的輸出寫入至標準輸出。

例如，若要解壓縮 foo 檔案，請鍵入：

```
uncompress foo
```

請參閱 *Commands Reference, Volume 5* 中的 **uncompress** 指令，以取得完整語法。

使用 unpack 指令將檔案解壓縮

使用 **unpack** 指令，將由 **pack** 指令所建立的檔案解壓縮。**unpack** 指令會針對每一個指定的檔案，搜尋一個名稱為 *File*.z 的檔案。如果這個檔案是壓縮過的檔案， **unpack** 指令會以其解壓縮後的版本來置換該檔案。**unpack** 指令會移除 *File* 的 .z 字尾，來重新命名新檔案。新檔案具有與原始壓縮檔案相同的存取模式、存取與修改日 期，以及擁有者。

unpack 指令只能用在以 .z 結尾的檔案。因此，當您指定的檔名結尾不是 .z 時，**unpack** 指令便會加上字尾，並在目錄中搜尋具有該字尾的檔名。

結束值為 **unpack** 指令無法解壓縮的檔案數目。如果發生下列任何狀況，便無法將檔案解壓縮：

- 檔名（.z 除外）具有 253 個以上的位元組。
- 檔案無法開啟。
- 檔案不是壓縮檔。
- 解出來之檔案的檔名已存在。
- 無法建立解壓縮的檔案。

註 : 如果要解壓縮的檔案具有鏈結，則 **unpack** 指令會在標準錯誤中寫入警告。新解出來之檔案的 i-node（索引節點）與其建立所在的壓縮檔不同。然而，任何其他鏈結至壓縮檔案的原始 inode 號碼檔案，仍存在且會被壓縮。

例如，若要解壓縮壓縮檔 chap1.z 和 chap2.z，請鍵入：

```
unpack chap1.z chap2
```

如此會展開壓縮檔 chap1.z 及 chap2.z，並將其置換成檔案 chap1 及 chap2。

註 : 使用 **unpack** 指令時，檔名加不加 .z 字尾都可以。

請參閱 *Commands Reference, Volume 5* 中的 **unpack** 指令，以取得完整語法。

系統映像檔及使用者定義的磁區群組備份

rootvg 儲存在硬碟或磁碟群組上，包含啟動檔案、BOS、配置資訊及任何選用性軟體產品。 使用者定義的磁區群組（亦稱作非 *rootvg* 磁區群組）通常包含資料檔及應用軟體。

您可以使用 SMIT 或指令程序來備份系統及磁區群組的映像檔。備份映像檔有兩種用途。一種用途是用系統備份映像檔來還原損毀的系統。另一種用途是將已安裝及配置的軟體從一個系統轉送到其他系統。

SMIT 程序使用 **mksysb** 指令，來建立可儲存在磁帶或檔案中的備份映像檔。如果選擇磁帶，備份程式會將開機映像檔寫入磁帶，使其適用於安裝。

註 :

- 啟動磁帶無法在 PowerPC 型個人電腦上製作，也不能用來啟動這種電腦。
- 如果選擇 SMIT 方法來備份，必須先安裝 bos.sysmgt 軟體套件中的 sysbr 檔案集。

相關概念

備份

一般而言，會保留使用者及系統資料的備份，以防不小心移除資料或磁碟發生故障。使用者資料與系統資料分別保存可方便管理備份。

備份系統映像檔和使用者定義磁區群組

您可以製作系統映像檔及使用者定義的磁區群組的備份。

備份 rootvg 磁區群組之前：

- 所有硬體必須均已安裝，包括外部裝置，如磁帶機與光碟機。
- 此備份程序需要 sysbr 檔案集，它位於「BOS 系統管理工具與應用程式」軟體套件中。鍵入下列指令以判斷 sysbr 檔案集是否已安裝在您的系統上：

```
lslpp -l bos.sysmgt.sysbr
```

如果系統已安裝 sysbr 檔案集，請繼續備份程序。

如果 **lslpp** 指令沒有列出 sysbr 檔案集，請先安裝它，再繼續備份程序。

```
installp -agqXd device bos.sysmgt.sysbr
```

其中 device 是軟體的位置；例如，/dev/rmt0 代表磁帶機。

備份使用者定義的磁區群組之前：

· 儲存之前，必須轉開磁區群組，且必須裝載檔案系統。

小心： 執行 **savevg** 指令，會造成之前儲存在所選輸出媒體中的所有資料流失。

· 請確定最近已清除過備份裝置，以避免發生錯誤。

下列程序說明如何製作系統的可安裝映像檔。

備份您的系統作業		
作業	***SMIT* 捷徑**	**指令或檔案**
備份 **rootvg** 磁區群組	1. 以 root 身分登入。 2. 裝載要備份的檔案系統。[1] **smit mountfs** 3. 卸載另一個本端目錄上裝載的所有本端目錄。**smit umountfs** 4. 在 /tmp 目錄中至少有 8.8 MB 的可用磁碟空間。[2] 5. 備份：**smit mksysb** 6. 防範寫入備份媒體。 7. 記錄任何備份 root 及使用者密碼。	1. 以 root 身分登入。 2. 裝載要備份的檔案系統。[1] 請參閱 **mount** 指令。 3. 卸載另一個本端目錄上裝載的所有本端目錄。請參閱 **umount** 指令。 4. 在 /tmp 目錄中至少有 8.8 MB 的可用磁碟空間。[2] 5. 備份。請參閱 **mksysb** 指令。 6. 防範寫入備份媒體。 7. 記錄任何備份 root 及使用者密碼。
驗證「備份磁帶」[3]	**smit lsmksysb**	
備份使用者定義的磁區群組 [4]	**smit savevg**	1. 必要時，請在備份前先修改檔案系統大小。[5] **mkvgdata** *VGName* 然後編輯 /tmp/vgdata/*VGName*/*VGName*.data 2. 儲存磁區群組。請參閱 **savevg** 指令。

註：

1. **mksysb** 指令不會備份裝載在 NFS 網路上的檔案系統。
2. **mksysb** 指令需要此工作空間以供持續備份使用。使用 **df** 指令，它會以 512 位元組區塊為單位進行報告，以便判斷 /tmp 目錄中的可用空間。必要時，使用 **chfs** 指令以變更檔案系統的大小。
3. 此程序會列出 **mksysb** 備份磁帶的內容。內容清單會驗證磁帶上的大部分資訊， 但不會驗證是否可以啟動磁帶安裝。從磁帶開機是驗證 **mksysb** 磁帶上的開機映像檔是否可以正確運作的唯一方式。
4. 如果您要從備份映像檔中排除使用者定義的磁區群組檔案，請建立名為 /etc/exclude.*volume_group_name* 的檔案，其中 *volume_group_name* 是您要備份的磁區群組名稱。然後編輯 /etc/exclude.*volume_group_name*，並輸入您不想併入備份映像檔的檔名型樣。此檔案中的型樣會輸入至符合 **grep** 指令慣例的型樣，以判斷要從備份中排除哪些檔案。
5. 如果您選擇要修改 *VGName*.data 檔案以變更檔案系統的大小，則不得在 **savevg** 指令中指定 **-i** 旗標或 **-m** 旗標，因為這樣會改寫 *VGName*.data 檔案。

相關資訊
安裝選用性軟體產品及服務程式更新
安裝系統備份

預先備份配置
在建立來源系統的備份映像檔之前，請先配置來源系統。不過，如果您計劃使用備份映像檔來安裝其他配置不同的目標系統，請在配置來源系統之前 建立映像檔。

來源 系統是從其中建立備份的系統。目標 系統是安裝備份所在的系統。

安裝程式僅自動安裝對所安裝機器進行硬體配置所需的裝置支援。因此，如果您使用系統備份來安裝其他機器，就可能需要在製作備份映像檔並將其用於安裝一或多個目標系統之前，先在來源系統上安裝其他裝置。

使用 SMIT 捷徑 `smit devinst`，可以在來源系統上安裝其他裝置支援。

· 如果來源及目標系統上有足夠的磁碟空間，則安裝所有裝置支援。

· 若來源及目標系統上的磁碟空間有限，則選擇性地安裝裝置支援。

備份會將下列配置從來源系統轉送到目標系統：

· 分頁空間資訊

· 邏輯磁區資訊

· rootvg 資訊

· 邏輯分割區放置（如果您已選取對映選項）。

相關資訊
安裝選用性軟體及服務程式更新
自訂安裝

檔案系統裝載與卸載
執行備份之前，您必須裝載所有要備份的檔案系統，並卸載所有不要備份的檔案系統。

備份方法程序只會備份 rootvg 中，已裝載的檔案系統。因此，在開始之前，必須先裝載您想要備份的所有檔案系統。同樣地，必須卸載不要備份的檔案系統。

如果將本端目錄裝載到同一檔案系統中的另一個本端目錄上，則此備份程序會將檔案備份兩次。例如，如果將 `/tmp` 裝載到 `/usr/tmp`，則會將 `/tmp` 目錄中的檔案備份兩次。這種重複可能會超出檔案系統可以保留的檔案數目，因而導致未來在安裝備份壓縮檔時失敗。

備份的安全考量
如果在其他系統上安裝備份映像檔，為安全起見，您可能不想要將密碼及網址複製到目標系統。

此外，將網址複製到目標系統，會建立重複的網址，使網路通訊中斷。

還原備份映像檔
安裝備份映像檔時，系統會檢查目標系統是否有足夠的磁碟空間，來建立備份上儲存的所有邏輯磁區。如果有足夠的空間，則會回復整個備份。否則，安裝會停止，並且系統會提示您選擇更多的目標硬碟。

在目標系統上建立的檔案系統與來源系統上的大小相同，除非在製作備份映像檔之前，將 `image.data` 檔案中的 SHRINK 變數設為 `yes`。`/tmp` 目錄是一個例外，您可以增加該目錄的大小，以配置足夠的空間給 **bosboot** 指令。如需設定變數的相關資訊，請參閱 `image.data` 檔案。

系統完成備份映像檔的安裝後，安裝程式會重新配置目標系統上的 ODM。如果目標系統與來源系統的硬體配置並非完全相同，程式可能會修改下列目標系統檔中的裝置屬性：

· `/etc/objrepos` 中以 Cu 開頭的所有檔案

· `/dev` 目錄中的所有檔案。

相關資訊
安裝系統備份

實作排定的備份
此程序說明如何開發及使用 Script， 用以執行使用者檔案的每週完整備份及每日漸進式備份。

- 使用此 Script 時，排程備份的資料量不能超過一個磁帶的資料量。
- 請確定以 **cron** 指令執行 Script 之前，備份裝置已載入磁帶了。
- 請確定裝置已連接且可供使用，特別是使用 Script 在晚上執行時。請使用 **lsdev -C | pg** 指令檢查裝置可用性。
- 請確定最近已清除過備份裝置，以避免發生錯誤。
- 如果您要備份的檔案系統可能還在使用中，請先卸載它們以防止檔案系統毀損。
- 在製作備份之前，請先檢查檔案系統。使用檔案系統驗證中敘述的程序，或執行 **fsck** 指令。

此程序中所含的 Script 只是參考模型，您必須小心裁定以符合特定站台的需求。

相關概念
備份策略
備份大量資料的方法有兩種。

使用 *cron* 指令來備份檔案系統
此程序說明如何撰寫可以傳送給 **cron** 指令執行的 **crontab** Script。

這個 Script 會在星期一到星期六每天晚上，備份兩個使用者檔案系統 /home/plan 及 /home/run。這兩個檔案系統備份在同一個磁帶上，每天早上需插入一個新磁帶以供當天晚上使用。星期一晚上的備份是完整保存（層次 0）。星期二到星期六的備份是漸進式備份。

1. 建立 **crontab** Script 的第一個步驟，是發出 **crontab**-e 指令。這會開啟一個空的檔案，您可以在其中寫入項目，這些項目會提交給每天晚上執行的 **cron** Script（預設編輯器是 **vi**）。請鍵入：

   ```
   crontab -e
   ```

2. 下面範例會顯示六個 **crontab** 欄位。欄位 1 代表分鐘，欄位 2 代表小時（24 小時表示法）， 欄位 3 代表當月的第幾天，欄位 4 代表月份。欄位 3 及 4 含有 *（星號），顯示 Script 每個月會在 **day/wk** 欄位所指定的日子執行。欄位 5 代表星期幾，也可使用數天範圍來指定該欄位，例如，1-6。欄位 6 代表正在執行的 shell 指令。

   ```
   min hr day/mo mo/yr day/wk       shell command
   0   2    *     *       1         backup -0 -uf /dev/rmt0.1 /home/plan
   ```

   上面顯示的指令行乃假設站台上的人員可在適當時間回應提示。 **backup** 指令的 -0（零）旗標代表層次零，或完整備份。**-u** 旗標會更新 /etc/dumpdates 檔案中的備份記錄，**f** 旗標則指定裝置名稱，在上述範例中即為原始磁帶機 0.1。
3. 針對要在特定日子進行備份的每個檔案系統，鍵入類似步驟 2 的指令行。下列範例顯示出要在兩個檔案系統上執行六天備份的完整 Script：

   ```
   0 2 * * 1 backup -0 -uf/dev/rmt0.1 /home/plan
   0 3 * * 1 backup -0 -uf/dev/rmt0.1 /home/run
   0 2 * * 2 backup -1 -uf/dev/rmt0.1 /home/plan
   0 3 * * 2 backup -1 -uf/dev/rmt0.1 /home/run
   0 2 * * 3 backup -2 -uf/dev/rmt0.1 /home/plan
   0 3 * * 3 backup -2 -uf/dev/rmt0.1 /home/run
   0 2 * * 4 backup -3 -uf/dev/rmt0.1 /home/plan
   0 3 * * 4 backup -3 -uf/dev/rmt0.1 /home/run
   0 2 * * 5 backup -4 -uf/dev/rmt0.1 /home/plan
   0 3 * * 5 backup -4 -uf/dev/rmt0.1 /home/run
   0 2 * * 6 backup -5 -uf/dev/rmt0.1 /home/plan
   0 3 * * 6 backup -5 -uf/dev/rmt0.1 /home/run
   ```

4. 儲存您建立的檔案並結束編輯器。作業系統會傳送 crontab 檔案至 **cron** Script。

相關資訊
rmt 特殊檔案

備份 *DMAPI* 管理的 *JFS2* 檔案系統上的檔案

tar 及 **backbyinode** 指令中的選項容許您備份延伸屬性 (EA)。

在 DMAPI 系統上使用 **backbyinode** 指令時，只會備份發出指令當時存在檔案系統中的資料。**backbyinode** 指令 會檢查中繼資料的當前狀態，再執行其工作。這有利於 DMAPI，因為它會備份受管理系統的狀態。不過，不會備份任何離線資料。

若要備份 DMAPI 檔案系統的所有資料，請使用可讀取所有檔案的指令，例如 **tar** 指令。如此會 使啟用 DMAPI 功能的應用程式還原 **tar** 指令存取的所有檔案資料，並在第二個及第三個儲存體之間來回移動資料，因此可能會影響到效能。

格式化磁片（*format* 或 *fdformat* 指令）

您可以利用 **format** 及 fdformat 指令， 格式化 *Device* 參數（預設裝置為 /dev/rfd0）指定之軟式磁碟機中的磁片。

小心：格式化磁片會摧毀磁片上任何現有的資料。

format 指令將判斷其裝置類型，裝置類型有：

- 5.25 英吋低密度磁片 (360 KB)，包含 40x2 磁軌，每一磁軌 9 磁區
- 5.25 英吋高容量磁片 (1.2 MB)，包含 80x2 磁軌，每一磁軌 15 磁區
- 3.5 英吋低密度磁片 (720 KB)，包含 80x2 磁軌，每一磁軌 9 磁區
- 3.5 英吋高容量磁片 (2.88 MB)，包含 80x2 磁軌，每一磁軌 36 磁區

所有磁片類型的磁區大小為 512 位元組。

使用 **format** 指令會將磁片格式化為高密度，除非 *Device* 參數指定了不同的密度。

使用 **fdformat** 指令會將磁片格式化為低密度，除非指定 **-h** 旗標。*Device* 參數指定包含要格式化之磁片的裝置（如代表磁碟機 0 的 /dev/rfd0 裝置）。

在格式化磁片之前，**format** 以及 **fdformat** 指令將提示驗證。此動作可讓您在必要時俐落地結束作業。

請參閱下列範例：

- 若要格式化 /dev/rfd0 裝置中的磁片，請鍵入：

```
format -d /dev/rfd0
```

- 若要格式化磁片，而不檢查是否有壞軌，請鍵入：

```
format -f
```

- 若要格式化的 360 KB 磁片位於 /dev/rfd1 裝置的 5.25 英吋 1.2 MB 軟碟機內，請鍵入：

```
format -l -d /dev/rfd1
```

- 若要使用 **fdformat** 指令，強制對磁片進行高密度格式化，請鍵入：

```
fdformat -h
```

請參閱 *Commands Reference, Volume 2* 中的 **format** 指令，以取得完整語法。

檢查檔案系統的完整性（*fsck* 指令）

使用 **fsck** 指令來檢查不一致的檔案系統，並交互修復它們。

在每一個檔案系統上執行此指令， 當作系統起始設定的一部分，是很重要的。您必須能夠讀取內含檔案系統的裝置檔案(例如，/dev/hd0 裝置）。一般而言，檔案系統應該是一致的，且 **fsck** 指令僅報告檔案系統中關於檔案、已使用區塊與未使用區塊資訊。如果檔案系統不一致，**fsck** 指令將顯示所發現不一致的相關資訊，並提示您要求同意修復。**fsck** 指令在其修復作業中是很保守的，並避免可能會導致有效的資料流失的動作。然而，在某些情況下，**fsck** 指令將建議摧毀受損的檔案。

小心：請務必在系統故障後，再執行檔案系統中的 **fsck** 指令。更正的動作 會導致某些資料的流失。每一個一致性更正的預設動作是等待操作員鍵入 yes 或 no。如果您對受影響的檔案不具有寫入許可權，則 **fsck** 指令將預設為沒有回應。

請參閱下列範例：

· 若要檢查所有預設的檔案系統，請鍵入：

```
fsck
```

此 **fsck** 指令格式將在變更檔案系統前，尋求您的許可。

· 若要自動修正預設檔案系統的次要問題，請鍵入：

```
fsck -p
```

· 若要檢查 /dev/hd1 檔案系統，請鍵入：

```
fsck /dev/hd1
```

此指令會檢查位於 /dev/hd1 裝置上已解除裝載的檔案系統。

註：fsck 指令並不會更正已裝載的檔案系統。

請參閱 *Commands Reference, Volume 2* 中的 **fsck** 指令，以取得完整語法。

複製到磁片或自磁片複製（*flcopy* 指令）

使用 **flcopy** 指令，可以將磁片（開啟為 /dev/rfd0）複製到在現行目錄中建立的 floppy 檔案。

訊息 Change floppy, hit return when done 會依需求顯示。然後，**flcopy** 指令會將 floppy 檔案複製到磁片。

請參閱下列範例：

· 若要複製 /dev/rfd1 到現行目錄下的 floppy 檔案，請鍵入：

```
flcopy -f /dev/rfd1 -r
```

· 若要複製磁片的前 100 個磁軌，請鍵入：

```
flcopy -f /dev/rfd1 -t 100
```

請參閱 *Commands Reference, Volume 2* 中的 **flcopy** 指令，以取得完整語法。

將檔案複製到磁帶或磁碟（*cpio -o* 指令）

使用 **cpio -o** 指令，自標準輸入讀取檔案路徑名稱， 然後將這些檔案和路徑名稱與狀態資訊一起複製到標準輸出。

路徑名稱不可超過 128 個字元。避免使用許多特殊鏈結檔案來組成 **cpio** 指令的路徑名稱，因為這樣可能會因記憶體不足而無法持續追蹤路徑名稱，而遺失鏈結資訊。

請參閱下列範例：

· 若要複製現行目錄下檔名結尾是 .c 的檔案到磁片，請鍵入：

```
ls *.c | cpio -ov >/dev/rfd0
```

-v 旗標顯示每一檔案名稱。

· 若要複製現行目錄及所有子目錄到磁片，請鍵入：

```
find . -print | cpio -ov >/dev/rfd0
```

此動作將儲存以現行目錄 (.) 開始的目錄樹。並且內含其所有子目錄與檔案。

· 若要使用較簡短的指令字串，請鍵入：

```
find . -cpio /dev/rfd0 -print
```

-print 項目會顯示每一個複製時的檔案名稱。

請參閱 *Commands Reference, Volume 1* 中的 **cpio** 指令，以取得完整語法。

自磁帶或磁碟複製檔案（*cpio -i* 指令）
使用 **cpio -i** 指令可以自標準輸入讀取由 **cpio -o** 指令所建立的保存檔，然後自其中複製名稱符合 *Pattern* 參數的檔案。

這些檔案將複製至現行目錄樹中。您可以使用 **ksh** 指令中說明的檔名表示法，列出一個以上的 *Pattern* 參數。*Pattern* 參數的預設值為星號 (*)，它會選取現行目錄中的所有檔案。在如 [a-z] 的表示式中，根據現行對照順序，連字號 (-) 代表從某處直至另一處。

註：型樣 "*.c" 及 "*.o" 必須含括在引號之內，免得 shell 將星號 (*) 視為型樣相符字元。這是一特殊個案，其中 **cpio** 指令本身將解碼為符合型樣字元。

請參閱下列範例：

· 若要列出使用 **cpio** 指令儲存到磁片的檔案，請鍵入：

```
cpio -itv </dev/rfd0
```

這會顯示先前以 **cpio** 指令格式儲存在 /dev/rfd0 檔案內的資料目錄。此清單類似於由 **ls -l** 指令所產生的長目錄報表。

· 如需僅列出檔案路徑名稱，請僅使用 **-it** 旗標。

· 若要從磁片複製之前使用 **cpio** 指令儲存的檔案，請鍵入：

```
cpio -idmv </dev/rfd0
```

這會將先前以 **cpio** 指令儲存在 /dev/rfd0 檔案上的檔案，複製回檔案系統（指定 **-i** 旗標）。如果目錄樹已儲存，則 **-d** 旗標可讓 **cpio** 指令建立適當目錄。**-m** 旗標將維護最後修改的生效時間。**-v** 旗標會使 **cpio** 指令在複製每一檔案時，顯示其名稱。

· 若要從磁片複製選取的檔案，請鍵入：

```
cpio -i "*.c" "*.o" </dev/rfd0
```

此指令將自磁片複製結尾為 .c 或 .o 的檔案。

請參閱 *Commands Reference, Volume 1* 中的 **cpio** 指令，以取得完整語法。

複製到磁帶或自磁帶複製（*tcopy* 指令）
使用 **tcopy** 指令來複製磁帶。

例如，若要複製某個串流磁帶到 9 軌的磁帶，請鍵入：

```
tcopy /dev/rmt0 /dev/rmt8
```

請參閱 *Commands Reference, Volume 5* 中的 **tcopy** 指令，以取得完整語法。

檢查磁帶的完整性（*tapechk* 指令）
使用 **tapechk** 指令，對連接的串流磁帶機執行基本的一致性檢查。

您可簡單地讀取磁帶，以偵測某些硬體的串流磁帶機故障。 **tapechk** 指令提供一執行磁帶讀取檔案層次的方法。

例如，若要檢查串流磁帶機上的前三個檔案，請鍵入：

```
tapechk 3
```

請參閱 *Commands Reference, Volume 3* 中的 **tapechk** 指令，以取得完整語法。

保存檔（*tar* 指令）

使用保存備份方法來製作一個以上檔案、或整個資料庫的副本，作為未來參考、歷史用途，或在原始資料損壞或遺失時，進行復原之用。

通常，會在特定資料從系統中移除時使用保存檔。

使用 **tar** 指令將檔案寫入保存儲存體中，或自保存儲存體中擷取檔案。**tar** 指令尋找在預設裝置（通常為磁帶）上的保存，除非您指定另一個裝置。

當寫入保存檔時，**tar** 指令會使用暫存檔（/tmp/tar* 檔案），並在記憶體中維護一個具有數個鏈結的檔案表。如果 **tar** 指令不能建立暫用檔，或如果沒有足夠的可用記憶體來保留鏈結表，您會收到一個錯誤訊息。

請參閱下列範例：

· 若要將 file1 和 file2 檔案寫入預設磁帶機上的新保存檔，請鍵入：

```
tar -c file1 file2
```

· 若要擷取 /dev/rmt2 磁帶機上 /tmp 目錄中的所有檔案，並使用擷取時間為修改時間，請鍵入：

```
tar -xm -f/dev/rmt2 /tmp
```

· 若要顯示現行目錄下 out.tar 磁碟保存檔中的檔案名稱，請鍵入：

```
tar -vtf out.tar
```

請參閱 *Commands Reference, Volume 5* 中的 **tar** 指令，以取得詳細資訊及完整的語法。

檔案備份

使用 **backup** 指令或 **smit** 指令，在備份媒體（例如：磁帶或磁片）上建立檔案的副本。

小心：如果您嘗試備份已裝載的檔案系統，則會顯示出訊息。**backup** 指令仍然繼續，但會在檔案系統中發生不一致的情況。此狀況並不適用於根 (/) 檔案系統。

您使用 **backup** 指令或 **smit** 指令建立的副本，具有下列其中一種備份格式：

· 以名稱來備份的特定檔案，使用 **-i** 旗標。

· 以 i-node 號碼來備份的整個檔案系統，使用 *-Level* 及 *FileSystem* 參數。

註：

– 在系統備份期間修改檔案，總是會有發生資料損毀的可能。因此，在系統備份程序期間，請確定系統活動已減至最少。

– 如果是在 8 公釐的磁帶上進行備份，並將其裝置區塊大小設為 0（零），則不可能直接從磁帶還原資料。如果您已經以 0 為設定值來完成備份，您可以使用 **restore** 指令下所說明的特殊程序，從這些備份中還原資料。

小心：請確定您所指定的旗標符合備份媒體。

使用 *backup* 指令來備份檔案

使用 **backup** 指令在備份媒體上建立檔案的副本。

例如，若要備份 **$HOME** 目錄中依名稱選取的檔案，請鍵入：

```
find $HOME -print | backup -i -v
```

-i 旗標會提示系統自標準輸入中讀取要備份之檔案的名稱。**find** 指令會在使用者的目錄中產生檔案清單。此清單可輸送到 **backup** 指令作為標準輸入。每一檔案複製時，**-v** 標示會顯示一份進度報告。這些檔案備份於本端系統的預設備份裝置上。

請參閱下列範例：

· 若要備份根檔案系統，請鍵入：

```
backup -0 -u /
```

0 層次及 / 告知系統要備份 /（根）檔案系統。檔案系統備份到 /dev/rfd0 檔案。**-u** 旗標告知系統要更新 /etc/dumpdates 檔案中的現行備份層次記錄。

· 若要備份從上次 0 層次備份之後 /（根）檔案系統中所有修改過的檔案，請鍵入：

```
backup -1 -u /
```

請參閱 *Commands Reference, Volume 4* 中的 **backup** 指令，以取得完整語法。

使用 *smit* 指令來備份檔案

使用 **smit** 指令來執行 **backup** 指令，此 backup 指令會在備份媒體上建立檔案的副本。

1. 在提示時鍵入：

   ```
   smit backup
   ```

2. 請在 **DIRECTORY 完整路徑名稱**欄位中，鍵入通常用來裝載檔案系統之目錄的路徑名稱：

   ```
   /home/bill
   ```

3. 在 **BACKUP** 裝置或 **FILE** 欄位中， 輸入輸出裝置名稱，如下列原始資料磁帶機範例所示：

   ```
   /dev/rmt0
   ```

4. 如果您要將錯誤訊息列印在畫面上，請使用 Tab 鍵來輪換**報告備份的每個階段**選用欄位。
5. 在系統管理環境中，使用**要寫入備份媒體的最大區塊數**欄位的預設值，因為此欄位不適用於磁帶備份。
6. 按 Enter 鍵來備份所指名的目錄或檔案系統。
7. 執行 **restore -t** 指令。

   如果此指令產生錯誤訊息，您必須重複整個備份。

系統關機

shutdown 指令是停止作業系統最安全、也是最徹底的一種方式。

您可能要在以下情況時關閉系統：

· 安裝新軟體或變更現有軟體的配置之後

· 存在硬體問題時

· 系統死當時

· 系統效能降低時

· 檔案系統可能毀損時。

當您指定適當的旗標時，此指令會通知使用者系統將要關機，請結束所有現有的處理程序、卸載檔案系統並停止系統。如需相關資訊，請參閱 **shutdown**。

複查下列資訊，以取得特定關機狀況的詳細資料：

關閉系統而不重新開機

有兩種關閉系統而不重新開機的方法。

您可以使用兩種方法來關閉系統而不重新開機：SMIT 捷徑或 **shutdown** 指令。

必要條件

您必須具有 root 使用者權限，才能關閉系統。

若要使用 SMIT 關閉系統：

1. 以 root 身分登入。

2. 在指令提示上，請鍵入：

```
smit shutdown
```

若要使用 **shutdown** 指令關閉系統：

1. 以 root 身分登入。
2. 在指令提示上，請鍵入：

```
shutdown
```

關閉系統進入單一使用者模式
在部分情況下，您可能需要關閉系統並進入單一使用者模式，以執行軟體維護與診斷。

1. 鍵入 `cd /`，以切換至根目錄。
   您必須在根目錄中，才能關閉系統進入單一使用者模式，以確定徹底卸載檔案系統。
2. 鍵入 `shutdown -m`。
   系統關機至單一使用者模式。

顯示系統提示，且您可以執行維護活動。

緊急關閉系統
使用 **shutdown** 指令，可以快速停止系統而不通知其他使用者。

在緊急狀況下，您可以使用 **shutdown** 指令來關閉系統。

鍵入 `shutdown -F`。**-F** 旗標指示 **shutdown** 指令略過傳送訊息給其他使用者，並盡快關閉系統。

系統環境

系統環境主要是指定義或控制某些處理程序執行層面的變數設定。

每一次啟動 shell 時即會設定或重設這些變數。從系統管理的角度而言，確定使用者在登入時已使用正確的值來設定是非常重要的。大部分的變數是在系統起始設定期間內設定。可以從 `/etc/profile` 檔案中讀取其定義，或根據預設值來設定定義。

設定檔
當您登入作業系統時，shell 會使用兩種類型的設定檔。

此 shell 會先評估檔案中包含的指令，然後執行這些指令，以設定系統環境。`/etc/profile` 檔案會控制系統上所有使用者的設定檔變數，而 `.profile` 檔案可讓您自訂您自己的環境，除了以上的這點不同之外，這兩個檔案的功能很類似。

提供下列設定檔及系統環境資訊：

- `/etc/profile` 檔案
- `.profile` 檔案
- 系統環境變數設定
- 變更當日訊息
- 第 43 頁的『時間資料操作服務』.

/etc/profile 檔案

登入時作業系統使用的第一個檔案是 `/etc/profile` 檔案。此檔案會控制全系統預設變數，如：

- 匯出變數
- 檔案建立遮罩 (umask)
- 終端機類型
- 郵件訊息，指出新郵件到達的時間。

系統管理者可為系統上的所有使用者配置 `profile` 檔案。僅系統管理者能夠變更此檔案。

.profile 檔案

登入時作業系統使用的第二個檔案是 .profile 檔案。 .profile 檔案存在於您的起始 ($HOME) 目錄，並可讓您自訂個別工作環境。.profile 檔案亦可置換 /etc/profile 檔案中設定的指令及變數。因為 .profile 檔是隱藏檔，所以請使用 **ls -a** 指令來列出它。使用 .profile 檔案來控制下列預設值：

- 要開啟的 shell
- 提示外觀
- 環境變數（例如，搜尋路徑變數）
- 鍵盤聲音

下列範例顯示典型的 .profile 檔案：

```
PATH=/usr/bin:/etc:/home/bin1:/usr/lpp/tps4.0/user:/home/gsc/bin::
epath=/home/gsc/e3:
export PATH epath
csh
```

此範例已定義兩個路徑（PATH 及 epath）、匯出它們，以及開啟 C shell (csh)。

您也可以使用 .profile 檔案（如果不存在，則使用 .profile 檔案）來判斷登入 shell 變數。您亦可自訂其他 shell 環境。例如，使用 .chsrc 及 .kshrc 檔案，以在啟動每種類型的 shell 時，分別修改 C shell 及 Korn shell。

時間資料操作服務

時間函數會存取並重新格式化現行系統日期及時間。

您無需對編譯器指定任何特殊旗標，即可使用時間函數。請將這些函數的標頭檔併入程式。若要併入標頭檔，請使用下列陳述式：

```
#include <time.h>
```

時間服務包括下列內容：

項目	說明
adjtime	更正時間以讓系統計時器同步化。
ctime, localtime, gmtime, mktime, difftime, asctime, tzset	將日期及時間轉換為字串表示。
getinterval, incinterval, absinterval, resinc, resabs, alarm, ualarm, getitimer, setitimer	操作間隔計時器的期限時間。
gettimer, settimer, restimer, stime, time	取得或設定指定之全系統計時器的現行值。
gettimerid	配置前置處理間隔計時器。
gettimeofday、settimeofday、ftime	取得並設定日期及時間。
nsleep、usleep、sleep	暫停執行中的現行處理程序。
reltimerid	釋放先前配置的間隔計時器。

64 位元模式所需的檔案集與硬體

核心以 64 位元模式執行，可用來快速存取大量資料，並有效處理 64 位元資料類型。

基礎作業系統 64 位元執行時期檔案集為 bos.64bit。安裝 bos.64bit 時，也會安裝 /etc/methods/cfg64 檔案。/etc/methods/cfg64 檔案為啟用 64 位元執行時期環境的指令。此指令會在開機處理程序的階段 3 期間由 rc.boot Script 呼叫。

從 AIX 6.1 開始，已經淘汰 32 位元核心。安裝 AIX 6.1 基本作業系統會啟用 64 位元模式。

註： 硬體必須為 64 位元功能，才能執行 AIX 6.1。下列 RS/6000® 機型使用 604e 處理器，而不是 64 位元功能：

· 7025 F50 系列

· 7026 H50 系列

· 9076 H50 系列

· 7043 150 系列

· 7046 B50 系列

若要驗證處理器的功能，請執行下列指令：

```
/usr/sbin/prtconf -c
```

prtconf 指令會傳回 32 或 64，視您的處理器功能而定。如果您的系統沒有 **prtconf** 指令，則可以使用加上 -y 旗標的 **bootinfo** 指令。

64 位元模式所需的硬體

您必須有 64 位元硬體，來執行 64 位元應用程式。

若要判別系統是 32 位元或 64 位元硬體架構：

1. 以 root 使用者的身分登入。
2. 在指令行中，輸入 `bootinfo -y`。

這樣就會根據硬體架構是 32 位元或 64 位元，而產生 **32** 或 **64** 的輸出。此外，如果您在任何版本的 AIX 中輸入 `lsattr -El proc0`，就會顯示伺服器的處理器類型。

32 位元及 64 位元效能比較

大多數情況下，在 64 位元硬體上執行 32 位元應用程式並不會有問題，因為 64 位元硬體 可以執行 64 位元和 32 位元兩種軟體。但是，32 位元硬體無法執行 64 位元軟體。

若要瞭解 執行在系統上的應用程式是否有效能上的問題，請參閱該應用程式的使用手冊中建議的執行環境。

動態處理器取消配置

AIX 可以偵測並自動停止使用故障的處理器。

從機型 7044 型號 270 開始，具有多個處理器之所有系統的硬體能夠偵測可更正的錯誤（由韌體收集）。這些錯誤不是嚴重的，且只要它們仍很少出現，就可以被安全地忽略。不過，當在特定的處理器上似乎顯現出失敗的型樣時，則此型樣可能指示此元件很可能會在不久的將來表現出嚴重的失敗。 此預測是由韌體依據失敗的速率及臨界值分析而做出的。

在這些系統上，AIX 會施行連續的硬體監視，並定期輪詢韌體以尋找硬體錯誤。當處理器錯誤數達到臨界值且韌體辨識到此系統元件很可能失敗時，韌體會傳回錯誤報告。在所有情況下，都會將錯誤記載到系統錯誤日誌。此外，在多重處理器系統上，依失敗的類型，AIX 會嘗試停止使用不可信賴的處理器並將其取消配置。此特性稱為動態處理器取消配置。

此時，韌體還會給該處理器標上旗標，以針對後續的重新開機持續取消配置該處理器，直到維護人員更換該處理器為止。

處理器取消配置對應用程式的影響

處理器取消配置對絕大多數應用程式（包括驅動程式及核心擴充）都是透通的。不過，您可以使用發佈的介面，來判斷應用程式或核心擴充是否在多重處理器機器上執行、找出有多少個處理器，並將執行緒連結到特定處理器。

將處理程序或執行緒連結到處理器的 bindprocessor 介面，是使用連結 CPU 號碼。連結 CPU 號碼的範圍是 [0..*N*-1]，其中 *N* 是指 CPU 的總數。若要避免中斷在 CPU 編號中假設不含「孔」的應用程式或核心擴充，AIX 一定讓應用程式看起來像它是要取消配置的「最後一個」（最高編號的）連結 CPU。例如，在 8 向 SMP 上，連結 CPU 號碼為 [0..7]。如果取消配置一個處理器，則可用 CPU 的總數會變為 7，並會將它們編號為 [0..6]。在外部，不管哪個實體處理器失敗，看起來都像是 CPU 7 消失了。

註： 此說明的其餘部分，術語 *CPU* 用於邏輯實體，且術語處理器用於實際實體。

若當處理器之其中一個需要取消配置時，AIX 無聲地終止它們的連結執行緒，或強制將其移至其他 CPU，則連結處理程序或執行緒的應用程式或核心擴充有可能被中斷。 動態處理器取消配置提供程式設計介面，以便當要發生處理器取消配置時，可通知那些應用程式及核心擴充。當這些應用程式及核心擴充收到通知時，它們會負責將它們的連結執行緒及相關資源（如計時器要求區塊)，從最後一個連結 CPU ID 移走，並讓它們自己適應新的 CPU 配置。

通知之後，如果部分執行緒仍連結到最後一個連結 CPU ID，則會中斷取消配置，中斷的取消配置記載在錯誤日誌中，而 AIX 會繼續使用故障的處理器。當處理器最終失敗時，其會導致整個系統失敗。 因此，應用程式或核心擴充接收即將發生處理器取消配置的通知，並針對此通知採取行動是很重要的。

即使在取消配置無法完成的罕見情況下，動態處理器取消配置仍會給予系統管理者預先警告。藉由將錯誤記載在錯誤日誌中，它會給系統管理者提供機會在系統上排程維護作業，以在發生整體系統失敗之前取代該故障的元件。

處理器取消配置處理程序
AIX 可以藉由取消配置故障的處理器，來停止它。

處理器取消配置的典型事件流程如下：

1. 韌體偵測到其中一個處理器已達到可回復錯誤臨界值。
2. 將韌體錯誤報告記載於系統錯誤日誌中，且當 AIX 在支援處理器取消配置的機器上執行時，AIX 會啟動取消配置處理程序。
3. AIX 會通知連結到最後一個連結 CPU 的非核心處理程序和執行緒。
4. AIX 會等待所有連結執行緒（最多 10 分鐘），以從最後一個連結 CPU 中移走。若執行緒仍保持連結，則 AIX 會中斷取消配置。
5. 若將所有處理程序或執行緒從故障的處理器切斷，則會呼叫先前已登錄的「高可用性事件處理常式 (HAEH)」。HAEH 可能會傳回中斷取消配置的錯誤。
6. 除非中斷，否則取消配置處理程序最後會停止故障的處理器。

若於取消配置的任何時間發生失敗，將記載該失敗及其原因。系統管理者可以查看錯誤日誌，採取更正動作（如果可能），並重新啟動取消配置作業。例如，若取消配置中斷的原因為某應用程式未切斷其連結執行緒，則系統管理者可以停止該應用程式、重新啟動取消配置作業，然後重新啟動應用程式。

啟用動態處理器取消配置
如果您的機器支援動態處理器取消配置，您可以使用 SMIT 或系統指令來**啟用**或**停用**此特性。

在安裝期間，依預設啟用動態處理器取消配置，提供具有正確硬體及韌體的機器來支援它。

SMIT 捷徑程序

1. 以 root 權限，在系統提示中鍵入 `smit system`，然後按 Enter 鍵。
2. 在**系統環境**視窗中，選取**變更/顯示作業系統的性質**。
3. 使用 SMIT 對話框來完成作業。

若要取得完成作業的其他資訊，您可以在 SMIT 對話框中選取「F1 輔助說明鍵」。

指令程序
如果具有 root 權限，則可以使用下列指令來使用動態處理器取消配置：

- 使用 **chdev** 指令以變更指定裝置的性質。如需使用此指令的相關資訊，請參閱 *Commands Reference, Volume 1* 中的 **chdev**。
- 如果處理器取消配置因任何原因而失敗，您可以在修正後使用 **ha_star** 指令來重新啟動它。如需使用此指令的相關資訊，請參閱 *Commands Reference, Volume 2* 中的 **ha_star**。
- 使用 **errpt** 指令以產生記載的錯誤報告。如需使用此指令的相關資訊，請參閱 *Commands Reference, Volume 2* 中的 **errpt**。

開啟及關閉處理器取消配置的方法
您可以藉由變更 ODM 物件 `sys0` 的 `cpuguard` 屬性值，來啟用或停用 動態處理器取消配置。

該屬性的可能值為 enable 及 disable。

預設為 enabled（屬性 cpuguard 有 enable 的值）。想要停用此特性的系統管理者，必須使用系統功能表（SMIT **系統環境**功能表）或 **chdev** 指令。（於前一版 AIX 中，預設值為 disabled）。

註： 若處理器取消配置關閉（停用），仍會記載錯誤。錯誤日誌中會包含諸如 CPU_FAILURE_PREDICTED 之類的錯誤，指出已通知 AIX 發生 CPU 問題。

重新啟動中斷的處理器取消配置
有時，處理器取消配置失敗的原因是應用程式未將其連結執行緒從最後的邏輯 CPU 移走。

一旦藉由切斷（如果這樣做安全的話）或停止應用程式修訂了此問題後，系統管理者就可使用 **ha_star** 指令，來重新啟動處理器取消配置處理程序。

此指令的語法為：

```
ha_star -C
```

其中 **-C** 代表 CPU 預測性失敗事件。

處理器狀態注意事項
您應該考慮數個關於處理器狀態的事項。

實體處理器在 ODM 資料庫中以名為 **proc***n* 的物件來代表，其中 *n* 是代表實體處理器號碼的十進位數。 就如 ODM 資料庫中所代表的所有其他裝置一樣，處理器物件擁有狀態（如已定義/可用）及屬性。

只要對應的處理器存在，**proc** 物件的狀態就一律為「可用」，而不管它是否可用。**proc** 物件的 **state** 屬性會指示處理器是否已使用，如果未使用，還會指示原因。此屬性可具有三個值：

項目	說明
enable	處理器已使用。
disable	處理器已被動態地取消配置。
faulty	處理器在啟動時由韌體宣告為已損毀。

若順利地取消配置故障的處理器，則處理器的狀態會從 **enable** 變更為 **disable**。此處理器也在韌體中標示為毀損，而與 AIX 無關。於重新開機時，取消配置的處理器將不可用，且其狀態將設為 **faulty**。但 ODM **proc** 物件仍標示為「可用」。 您必須實際上從主機板移除毀損的 CPU，或移除 CPU 主機板（如果可能的話），使 **proc** 物件變更為「已定義」。

在下列範例中，處理器 **proc4** 運作正常，且正由作業系統使用，如下列輸出所示：

```
        # lsattr -EH -l proc4
    attribute    value            description        user_settable

        state        enable            Processor state        False
type        PowerPC_RS64-III    Processor type        False
        #
```

處理器 **proc4** 獲得預測性失敗，且由作業系統取消配置，如下列輸出所示：

```
        # lsattr -EH -l proc4
    attribute    value            description        user_settable

        state        disable            Processor state        False
type        PowerPC_RS64-III    Processor type        False
        #
```

於下次系統重新啟動時，韌體會將處理器 **proc4** 報告為已損毀，如下列輸出所示：

```
        # lsattr -EH -l proc4
    attribute    value            description        user_settable

        state        faulty            Processor state        False
type        PowerPC_RS64-III    Processor type        False
        #
```

但處理器 **proc4** 的狀態仍為「可用」，如下列輸出所示：

```
    # lsdev -CH -l proc4
name        status           location        description

    proc4        Available        00-04            Processor
#
```

取消配置錯誤日誌項目

三則不同的錯誤日誌訊息與 CPU 取消配置相關。

範例如下。

errpt 簡短格式 - 摘要

下面是 **errpt** 指令（無選項）顯示的項目範例：

```
# errpt
IDENTIFIER      TIMESTAMP       T    C    RESOURCE_NAME    DESCRIPTION
804E987A        1008161399      I    O    proc4
CPU DEALLOCATED
8470267F        1008161299      T    S    proc4
CPU DEALLOCATION ABORTED
1B963892        1008160299      P    H    proc4
CPU FAILURE PREDICTED
#
```

- 若啟用了處理器取消配置，則 CPU FAILURE PREDICTED 訊息之後 一律跟隨著 CPU DEALLOCATED 訊息或 CPU DEALLOCATION ABORTED 訊息。
- 若未啟用處理器取消配置，則僅會記載 CPU FAILURE PREDICTED 訊息。 於已記載一或多則 CPU FAILURE PREDICTED 訊息之後的任何時間啟用處理器取消配置，均會針對每個被報告失敗的處理器，起始取消配置處理程序並產生成功或失敗的錯誤日誌項目（如上面所述）。

errpt 長格式 - 詳細說明

下列是使用 **errpt -a** 取得的輸出格式：

- CPU_FAIL_PREDICTED

  錯誤說明：預測性處理器失敗

  此錯誤指示硬體偵測到處理器很有可能在不久的將來失敗。無論是否已啟用處理器取消配置，一律會記載之。

  明細資料：實體處理器號碼、位置

  範例錯誤日誌項目 - 長格式

```
        LABEL:            CPU_FAIL_PREDICTED
        IDENTIFIER:        1655419A

        Date/Time:        Thu Sep 30 13:42:11
        Sequence Number:    53
        Machine Id:        00002F0E4C00
        Node Id:        auntbea
        Class:            H
        Type:            PEND
        Resource Name:        proc25
        Resource Class:        processor
        Resource Type:        proc_rspc
        Location:        00-25

      說明
        CPU FAILURE PREDICTED

    Probable Causes
    CPU FAILURE

    Failure Causes
    CPU FAILURE

        Recommended Actions
                ENSURE CPU GARD MODE IS ENABLED
                RUN SYSTEM DIAGNOSTICS.
```

```
Detail Data
PROBLEM DATA
0144    1000    0000    003A    8E00    9100    1842    1100    1999    0930    4019
0000    0000    0000    0000    0000
0000    0000    0000    0000    0000    0000    0000    0000    4942    4D00    5531
2E31    2D50    312D    4332    0000
0002    0000    0000    0000    0000    0000    0000    0000    0000    0000    0000
0000    0000    0000    0000    0000
0000    0000    0000    0000    0000    0000    0000    0000    0000    0000    0000
0000    0000    0000    0000    0000
...    ...    ...    ...    ...
```

- CPU_DEALLOC_SUCCESS

錯誤說明：偵測到預測性處理器失敗之後，已順利取消配置處理器。如果啟用了處理器取消配置，且已順利取消配置 CPU，則會記載此訊息。

明細資料：已取消配置之處理器的邏輯 *CPU* 號碼。

範例：錯誤日誌項目 - 長格式：

```
        LABEL:            CPU_DEALLOC_SUCCESS
        IDENTIFIER:        804E987A

        Date/Time:        Thu Sep 30 13:44:13
        Sequence Number:    63
        Machine Id:        00002F0E4C00
        Node Id:        auntbea
        Class:            O
    Type:            INFO
        Resource Name:        proc24

      說明
    CPU DEALLOCATED


        Recommended Actions
                MAINTENANCE IS REQUIRED BECAUSE OF CPU FAILURE

    Detail Data
        LOGICAL DEALLOCATED CPU NUMBER

        0
```

於此範例中，發生失敗時已順利取消配置 **proc24**，且為邏輯 CPU **0**。

- CPU_DEALLOC_FAIL

錯誤說明：處理器取消配置（由於預測性處理器失敗）未順利完成。如果啟用了 CPU 取消配置，且尚未順利取消配置 CPU，則會記載此訊息。

明細資料：原因碼、邏輯 *CPU* 號碼、視失敗類型而異的其他資訊。

原因碼是數字十六進位值。可能的原因碼為：

項目	說明
2	一或多個處理程序/執行緒仍與最後邏輯 CPU 連結。此時，明細資料會提供導致錯誤之處理程序的 PID。
3	已登錄的驅動程式或核心擴充會在得到通知時傳回錯誤。此時，明細資料欄位包含導致錯誤之驅動程式或核心擴充的名稱（ASCII 編碼）。
4	取消配置處理器會導致機器的可用 CPU 少於兩個。此作業系統不會在 *N* 向機器上取消配置多於 *N*-2 個處理器，以免讓應用程式或核心擴充弄錯，因為應用程式或核心擴充是使用可用處理器的總數來判斷它們是在「單一處理器 (UP)」系統（在此系統上，可以安全地跳過使用多重處理器鎖定）上還是在「對稱多重處理器 (SMP)」）上執行。
200 (0xC8)	會停用處理器取消配置（ODM 屬性 **cpuguard** 的值為 disable）。除非手動啟動 **ha_star**，否則您通常不會看到此錯誤。

範例：錯誤日誌項目 - 長格式

範例 1：

```
        LABEL:            CPU_DEALLOC_ABORTED
        IDENTIFIER:        8470267F
        Date/Time:        Thu Sep 30 13:41:10
        Sequence Number:    50
        Machine Id:        00002F0E4C00
        Node Id:        auntbea
        Class:            S
        Type:            TEMP
        Resource Name:        proc26

  說明
CPU DEALLOCATION ABORTED

Probable Causes
SOFTWARE PROGRAM

Failure Causes
SOFTWARE PROGRAM

    Recommended Actions
            MAINTENANCE IS REQUIRED BECAUSE OF CPU FAILURE
        SEE USER DOCUMENTATION FOR CPU GARD

Detail Data
DEALLOCATION ABORTED CAUSE
0000 0003
DEALLOCATION ABORTED DATA
6676 6861 6568 3200
```

於此範例中，**proc26** 的取消配置失敗。 原因碼 3 表示核心擴充向核心通知常式傳回錯誤。上面的 DEALLOCATION ABORTED DATA 表示 **fvhaeh2**，這是擴充在向核心登錄時使用的名稱。

範例 2：

```
        LABEL:            CPU_DEALLOC_ABORTED
        IDENTIFIER:        8470267F
        Date/Time:        Thu Sep 30 14:00:22
        Sequence Number:    71
        Machine Id:        00002F0E4C00
        Node Id:        auntbea
        Class:            S
        Type:            TEMP
        Resource Name:        proc19

  說明
CPU DEALLOCATION ABORTED

Probable Causes
SOFTWARE PROGRAM

Failure Causes
SOFTWARE PROGRAM

    Recommended Actions
        MAINTENANCE IS REQUIRED BECAUSE OF CPU FAILURE;
        SEE USER DOCUMENTATION FOR CPU GARD

Detail Data
DEALLOCATION ABORTED CAUSE
0000 0002
DEALLOCATION ABORTED DATA
0000 0000 0000 4F4A
```

於此範例中，**proc19** 的取消配置失敗。原因碼 2 指出執行緒連結至最後邏輯處理器，且在接收到 SIGCPUFAIL 信號之後未切斷。DEALLOCATION ABORTED DATA 顯示這些執行緒屬於處理程序 **0x4F4A**。

ps 指令的選項（-o THREAD、-o BND）允許列出所有執行緒或處理程序， 及其所連結的 CPU 號碼（當適用時）。

範例 3：

```
        LABEL:            CPU_DEALLOC_ABORTED
        IDENTIFIER:        8470267F

        Date/Time:        Thu Sep 30 14:37:34
        Sequence Number:    106
        Machine Id:        00002F0E4C00
        Node Id:        auntbea
        Class:            S
        Type:            TEMP
        Resource Name:        proc2

  說明
CPU DEALLOCATION ABORTED

Probable Causes
SOFTWARE PROGRAM

Failure Causes
SOFTWARE PROGRAM

    Recommended Actions
            MAINTENANCE IS REQUIRED BECAUSE OF CPU FAILURE
        SEE USER DOCUMENTATION FOR CPU GARD

Detail Data
DEALLOCATION ABORTED CAUSE
0000 0004
DEALLOCATION ABORTED DATA
0000 0000 0000 0000
```

在此範例中，**proc2** 的取消配置失敗，原因是在失敗時有兩個或少於兩個作用中處理器（原因碼 4）。

系統環境變數設定
系統環境主要是指定義或控制某些處理程序執行層面的變數設定。

每一次啟動 shell 時即會設定或重設這些變數。從系統管理觀點來看，使用者在登入時使用正確的設定值是很重要的一件事。大部分的變數是在系統起始設定期間內設定。可以從 /etc/profile 檔案中讀取其定義，或根據預設值來設定定義。

測試系統電池
如果系統無法顯示時間，則原因可能是電池用完或脫離。

1. 欲判斷系統電池的狀態，請鍵入下列 **diag** 指令：

   ```
   diag -B -c
   ```

2. 出現「診斷程式」主功能表時，選取**問題判斷**選項。
   如果電池脫離或用完了，則在問題功能表中會顯示服務要求號碼 (SRN)。請記錄「問題彙總表格」中「項目 4」的 SRN，並向您的客戶服務組織報告此問題。

如果系統電池是可運作的，則您的系統時間可能因 **date** 或 **setclock** 指令的執行不正確或不成功而未正確地重設。

相關概念
設定系統時鐘
系統計時器會記錄系統事件的時間，可讓您排程系統事件（如在早上三點時執行硬體診斷程式），並告知您第一次建立或最後一次儲存檔案的時間。

設定系統時鐘
系統計時器會記錄系統事件的時間，可讓您排程系統事件（如在早上三點時執行硬體診斷程式），並告知您第一次建立或最後一次儲存檔案的時間。

使用 **date** 指令以設定系統計時器。使用 **setclock** 指令以藉由聯絡時間伺服器來設定時間與日期。

相關工作
測試系統電池
如果系統無法顯示時間，則原因可能是電池用完或脫離。

date 指令
date 指令會顯示或設定日期及時間。

輸入下列指令以判斷系統所辨識的目前日期及時間：

```
/usr/bin/date
```

小心：當執行的系統上有多個使用者時，請勿變更日期。

在使用 *Date* 參數設定日期時，可以使用下列格式：

· *mmddHHMM*[*YYyy*]（預設值）

· *mmddHHMM*[*yy*]

Date 參數的變數定義如下：

項目	說明
mm	指定第幾個月。
dd	指定一個月的第幾天。
HH	指定一天中的小時（使用 24 小時表示法）。
MM	指定分鐘數。
YY	指定四位數年份的前兩位數。
yy	指定年份的最後兩個數字。

在 root 權限下，您可以使用 **date** 指令來設定目前的日期及時間。例如：

```
date 021714252002
```

會將日期設為 2002 年 2 月 17 日，並將時間設為 14:25。如需 **date** 指令的詳細資訊，請參閱 *Commands Reference, Volume 2* 中的相關說明。

setclock 指令
setclock 指令會藉由從網路上的時間伺服器要求目前的時間來顯示或設定日期與時間。

若要顯示系統的日期及時間，請鍵入：

```
/usr/sbin/setclock
```

setclock 指令會採用時間伺服器的第一個回應、轉換在該處發現的行事曆時鐘讀取，以及顯示區域日期與時間。如果沒有時間伺服器有回應，或如果網路無法運作，則 **setclock** 指令會顯示有關該影響的訊息，並將日期及時間設定保持不變。

註：任何執行 **inetd** 常駐程式的主機都可以當成時間伺服器。

在 root 權限下，您可以使用 **setclock** 指令將網際網路 TIME 服務要求傳送到時間伺服器主機，並依此來設定本端日期及時間。例如：

```
setclock
TimeHost
```

其中 *TimeHost* 是時間伺服器的主機名稱或 IP 位址。

相關資訊
setclock 指令

***Olson* 時區支援及設定**
從 AIX 6.1 開始，就提供支援與 Olson 資料庫一致的時區值。

先前 AIX 版次中所支援的 POSIX 時區規格，不會充分地處理時區規則的變更，例如，日光節約時間。Olson 資料庫會維持時區規則的歷程記錄，以便如果規則在特定位置發生變更，則 AIX 會正確解譯現在及過去的日期及時間。

符合 POSIX 規格的時區定義仍由 AIX 支援及辨識。AIX 會檢查 **TZ** 環境變數，以判斷環境變數是否符合 Olson 時區值。如果 **TZ** 環境變數不符合 Olson 時區值，則 AIX 會遵循 POSIX 規格規則。

如需 TZ 環境變數的詳細資料，請參閱環境檔案。

若要使用 Olson 定義的值來設定時區，請使用下列 SMIT 路徑：**系統環境 > 變更/顯示日期、時間及時區 > 使用系統定義的值來變更時區**。

日期設定的訊息
每一次使用者登入系統時，即會顯示日期的訊息。

這對所有使用者而言，是一種資訊通訊的便利方式，如安裝的軟體版本號碼或現行的系統新聞。若要變更日期訊息，請使用您喜好的編輯器來編輯 `/etc/motd` 檔案。

IBM License Metric Tool 的 AIX 使用度量資料（SLM 標籤）

AIX 作業系統產生的「軟體授權度量 (SLM)」標籤用作 IBM License Metric Tool 使用的使用度量資料。使用度量資料會記錄虛擬 CPU （vCPU）資訊，其代表系統中線上虛擬 CPU 的數目。

若要使用 IBM License Metric Tool，您必須安裝在 AIX 作業系統擴充套件中可用的 `slm.rte` 檔案集，並且還可從網站下載檔案集。

IBM License Metric Tool 產生可動態擴展的 `vcpu.slmtag` 軟體授權度量標籤 (SLMTAG) 檔案。`vcpu.slmtag` 檔案位於 `/var/opt/slm` 目錄中。由 BigFix® Agent (BESClient) 使用此檔案與 IBM License Metric Tool Agent 軟體合併，並將檔案傳送至 IBM License Metric Tool 伺服器。

在系統中安裝與配置的 IBM License Metric Tool 可以與 BigFix Agent (BESClient) 一起使用。 BigFix Agent 必須個別從 IBM License Metric Tool 伺服器取得，且安裝在用戶端上。 若在系統上未安裝 BigFix Agent，則會建立 SLMTAG 檔案，但不會將資料傳送至 IBM License Metric Tool 伺服器。

您可以配置 IBM License Metric Tool 及定義在 `/etc/environment` 檔案中的環境變數。安裝時，這些變數會設為預設值。可以配置下列可配置變數。

變數	詳細資料
SLM_VCPU_PERIOD_HOURS	為新增度量項目而指定的時段（以小時為單位）。預設值是 720 小時（30 天）。此值必須是 4 的倍數。如果指定的值不是 4 的倍數，則 IBM License Metric Tool 會視其為下一個 4 的倍數。最小值為 4 小時。
SLM_VCPU_MAX_FSIZE	指定的檔案大小上限（以位元組為單位）為記錄檔案的大小上限（以位元組為單位）。如果檔案大小超出此限制，則會保存檔案。預設值為 2097152 個位元組 (2 MB)。最小值為 10 KB。 如果您指定一個低值，則 IBM License Metric Tool 會將其視為 10 KB。
SLM_VCPU_COUNT_ARCH	工具所保留的保存數目。預設值為 4。最小值為 1。如果您指定一個低值，則 IBM License Metric Tool 會視其為 1。

使用度量資料會顯示在 IBM License Metric Tool Server GUI 中的「**資源使用率**」頁面上。使用度量資料會在主機名稱旁邊顯示為 VCPU 度量類型及 COUNT 子類型。

相關資訊
IBM License Metric Tool 9.2.0

AIX Runtime Expert

AIX Runtime Expert 提供一組可以用於單一合併的簡化動作，以收集、套用及驗證一個以上 AIX 實例的執行時期環境。

AIX 元件提供的工具（例如 Reliability Availability Serviceability (RAS)、Security 或 Kernel）可讓您變更每一個元件層內的設定，以將作業系統調整為特定需要或需求。AIX Runtime Expert 透過使用可擴充的架構來處理 AIX 中現有的許多不同配置方法，以啟用系統整體的配置。

AIX Runtime Expert 使用配置設定檔，將多元件配置指令執行為單一動作。您可以使用此設定檔，將相同的系統設定套用至多個系統。AIX Runtime Expert 提供簡化的替代方案，以管理一個以上系統的執行時期配置，但未防止使用其他方法來變更系統設定。

AIX Runtime Expert 概念

您必須對 AIX Runtime Expert 有基本瞭解之後，才能開始使用它。

AIX Runtime Expert 基本功能支援單一 AIX 系統的配置設定檔管理及應用。若要啟用單一設定檔的多個系統可調整使用，AIX 系統在啟動時或在目標 AIX 端點上受到管理作業指示時，都可以探索到並使用 LDAP 型設定檔說明。只有使用「網路安裝管理程式 (NIM)」元件才能進行 AIX Runtime Expert 的遠端管理。使用現有的 NIM 功能，即可從 NIM 主要伺服器機器，在數個獨立式 NIM 用戶端上遠端執行 AIX Runtime Expert。

AIX Runtime Expert 設定檔

AIX Runtime Expert 設定檔可用來設定執行中系統上的值、擷取執行中系統的值，以及將值與執行中系統或另一個設定檔進行比較。

設定檔說明目標功能範圍的一個以上執行時期配置控制項及其設定。設定檔可以代表一組完整控制項或控制項子集及其值。配置設定檔是標準 XML 檔案。使用 AIX Runtime Expert，您可以在已定義系統上管理並套用設定檔。

設定檔可包含未含任何值的配置參數及調整參數，如同範例設定檔一樣。沒有任何參數的設定檔用途，是要從指定的設定檔中擷取現行系統值。至少包含一個沒有任何值的參數的設定檔，其限制如下：

- 使用 **artexset** 指令失敗，並發生錯誤。
- 使用 **artexdiff** 指令會針對每一個沒有值的參數傳回警告訊息。

設定檔中的參數值可以包含下列項目：

- 沒有值
- 二進位大型物件值，這是與行內文字檔相同的 Base64 編碼二進位資料。二進位大型物件值可用來取代現存檔案，如 `/etc/motd` 或 `/etc/hosts`。
- 非二進位大型物件值，這是指派給系統配置參數的值，如整數或字串。

在 `/etc/security/artex/samples` 目錄中，您可以檢視現有的範例設定檔。範例設定檔只包含與 AIX Runtime Expert 一起安裝的預設值所支援的參數名稱。範例設定檔中的參數沒有任何值。範例設定檔是唯讀檔。您可以使用範例設定檔作為範本，以建立新的配置設定檔。您不可以將現有的範例套用至執行中系統。

下列範例是部分可以透過配置設定檔控制的基本配置指令：

- 網路配置
  - 否
  - mktcpip
- 核心配置
  - ioo
  - schedo
- RAS 配置
  - alog
- 安全配置
  - setsecattr

範例

下列範例顯示已指派不同參數值的不同型錄及子型錄的配置設定檔。您可以使用任何 XML 編輯器或使用 **vi** 指令來編輯此設定檔，以及變更已定義參數的現有值。

```
<?xml version="1.0" encoding="UTF-8" ?>
<Profile origin="get" version="1.0" date="2009-04-25T15:33:37Z">
<Catalog id="vmoParam">
<Parameter name="kernel_heap_psize" value="0" applyType="nextboot" reboot="true" />
<Parameter name="maxfree" value="1088" />
</Catalog>
<Catalog id="noParam">
<SubCat id="tcp_network">
<Parameter name="tcp_recvspace" value="16384" />
<Parameter name="tcp_sendspace" value="16384" />
</SubCat>
<SubCat id="general_network">
<Parameter name="use_sndbufpool" value="1" applyType="nextboot" reboot="true" />
</SubCat>
</Catalog>
<Catalog id="lvmoParam">
<Parameter name="max_vg_pbuf_count" value="0">
<Target class="vg" instance="rootvg" />
</Parameter>
<Parameter name="pv_pbuf_count" value="512">
<Target class="vg" instance="rootvg" />
</Parameter>
</Catalog>
```

相關工作

修改 AIX Runtime Expert 設定檔
AIX Runtime Expert 設定檔是 XML 檔案，而且可以使用任何 XML 編輯器或任何文字編輯器進行修改。

建立 AIX Runtime Expert 設定檔
利用 **artexget** 指令，並使用 /etc/security/artex/samples 目錄中的現有範例，建立新的設定檔。範例設定檔是一個範本，可讓您用來建立可以修改並儲存為自訂檔案的設定檔。

取得 AIX Runtime Expert 設定檔值
使用 **artexget** 指令，可以尋找設定檔的相關資訊。

套用 AIX Runtime Expert 設定檔
若要使用設定檔中的配置及可調整參數來設定系統，請使用 **artexset** 指令來套用設定檔。

***AIX Runtime Expert* 型錄**
型錄是一種機制，定義及指定可以在 AIX Runtime Expert 上作業的配置控制項。

提供 AIX Runtime Expert 目前所支援控制項的型錄。型錄是定義檔，可以將配置設定檔值對映至用來執行指令及配置動作的參數。

AIX Runtime Expert 提供現有的唯讀型錄（位於 /etc/security/artex/catalogs 目錄），以識別可以修改的值。請勿修改這些型錄。

每一個型錄都包含某個元件的參數。不過，部分型錄可以包含多個極為相關元件的參數。型錄的名稱說明型錄中包含的元件。每一個型錄中的 <description> XML 元素都會提供型錄的說明。

AIX Runtime Expert* 及 *LDAP
AIX Runtime Expert 可以從「輕量型目錄存取通訊協定 (LDAP)」伺服器擷取設定檔。

AIX Runtime Expert 設定檔必須儲存為 ibm-artexProfile 物件，並且具有下列必要屬性：

- Ibm-artexProfileName。AIX Runtime Expert 設定檔名稱。
- Ibm-artexProfileXMLData。儲存為 octetString 的 AIX Runtime Expert 設定檔的 XML 內容。

AIX Runtime Expert 綱目必須先安裝於 LDAP 伺服器，然後才能儲存任何 AIX Runtime Expert 設定檔。設定 AIX Runtime Expert 的 LDAP 伺服器，與設定進行使用者鑑別的 LDAP 伺服器類似。如需設定 LDAP 的相關資訊，請參閱設定 ITDS 安全資訊伺服器。

設定 AIX Runtime Expert 的 LDAP 用戶端，與設定進行使用者鑑別的 LDAP 用戶端類似。如需相關資訊，請檢視設定 LDAP 用戶端主題。若要設定 LDAP 用戶端，請使用 **mksecldap -c** 指令正確地配置

secldapclntd 常駐程式。AIX Runtime Expert 是根據 **secldapclntd** 常駐程式來存取 LDAP 伺服器。AIX Runtime Expert 預設會尋找 ID DN: ou=artex,cn=AIXDATA 下的設定檔項目。 更新 /etc/security/ldap/ldap.cfg secldapclntd 配置檔中的 artexbasedn 鍵值，即可自訂此 DN。

上傳 AIX Runtime Expert 設定檔

若要上載 AIX Runtime Expert 設定檔，您可以建立 LDAP 資料交換格式 (LDIF) 檔案並使用 **ldapadd** 指令，或是使用 LDAP 管理工具（如「Tivoli® Directory Server Web 管理工具」）。

下列是以 LDIF 儲存的設定檔範例：

```
dn: ou=artex,cn=AIXDATA
objectClass: organizationalUnit
objectClass: top
ou: artex

dn: ibm-artexProfileName=alogProfile.xml,ou=artex,cn=AIXDATA
objectClass: ibm-artexProfile
objectClass: top
ibm-artexProfileName: alogProfile.xml
ibm-artexProfileXMLData:< file:///etc/security/artex/samples/alogProfile.xml
```

下列範例顯示如何使用 **ldapadd** 指令及範例 LDIF 檔案 sample.ldif 來上傳設定檔：

```
ldapadd -c -h <ldaphost> -D cn=admin -w <password> -f sample.ldif
```

相關工作

建立 AIX Runtime Expert 設定檔
利用 **artexget** 指令，並使用 /etc/security/artex/samples 目錄中的現有範例，建立新的設定檔。範例設定檔是一個範本，可讓您用來建立可以修改並儲存為自訂檔案的設定檔。

相關資訊

IBM Security Directory Server

AIX Runtime Expert 及 *RBAC*

可以使用「角色型存取控制 (RBAC)」來提供非 root 使用者執行 AIX Runtime Expert 指令的能力。

AIX Runtime Expert 授權

安裝 **artex.base.rte** 檔案集時會建立三個系統授權，容許不同的 AIX Runtime Expert 功能存取層次：

- **aix.system.config.artex.read** 授權容許執行 **artexlist** 及 **artexmerge** 指令。 也容許 **artexget** 及 **artexdiff** 指令，但只能取得設定檔值。無法從系統擷取值（即表示 **artexget** 指令無法與 *–r*、*–n* 或 *–p* 旗標一起執行，而 **artexdiff** 指令只能在兩個設定檔之間執行）。
- **aix.system.config.artex.get** 授權容許 **artex.system.config.read** 授權所容許的所有作業，此外還容許 **artexget** 及 **artexdiff** 指令的未限定執行。
- **aix.system.config.artex.set** 授權容許 **artex.system.config.get** 授權所容許的所有作業，此外還容許 **artexset** 指令的執行。

AIX Runtime Expert 角色

AIX Runtime Expert 不會建立任何新角色，然而 **artex.base.rte** 檔案集會將 **aix.system.config.artex** 授權新增至 **SysConfig** 角色。 任何角色為 **SysConfig** 或具有任何含括角色（例如 **isso** 角色）的使用者將可以執行 **artexlist**、**artexmerge**、 **artexdiff**、**artexget** 及 **artexset** 指令。

限制

基於安全理由，**ARTEX_CATALOG_PATH** 環境變數的使用受限於 root 使用者。透過 RBAC 授與 AIX Runtime Expert 指令執行權限的非 root 使用者無法使用 **ARTEX_CATALOG_PATH** 環境變數。

管理 AIX Runtime Expert

AIX Runtime Expert 使用一些簡單的指令，建立設定檔、修改設定檔、合併設定檔，以及套用設定檔。

配置 *AIX Runtime Expert*

AIX Runtime Expert 會使用配置檔 /etc/security/artex/config/artex.conf。

配置檔中的項目由配置選項的名稱所組成，隨後接著一個以上的空格或值。會忽略空白行以及以 # 符號開頭的行。

支援下列選項：

表 *1.* 配置選項

Options	**說明**
ARTEX_CATALOG_PATH	用來搜尋編目檔的目錄清單（以冒號區隔）。此選項會以 **ARTEX_CATALOG_PATH** 環境變數置換。預設路徑是 /etc/security/artex/catalogs。
ARTEX_PROFILE_PATH	如果沒有指定目錄，則為 **artexlist** 指令用來搜尋設定檔的目錄清單（以冒號區隔）。此選項會以 **ARTEX_PROFILE_PATH** 環境變數置換。預設路徑是 /etc/security/artex/samples。
DEBUG_LOG_CATEGORY	日誌檔的除錯種類。可以重複此選項以選取多個除錯種類。
DEBUG_LOG_LEVEL	日誌檔的除錯層次介於 0（無除錯追蹤資料）及 3（最詳細）之間。
MAX_CMDS	並行執行的外部指令數目上限。會將 AIX Runtime Expert 所執行的外部指令排入佇列，因此在所有給定時間都不會有 MAX_CMDS 外部指令之外的指令同步執行。預設值為 10。

建立 *AIX Runtime Expert* 設定檔

利用 **artexget** 指令，並使用 /etc/security/artex/samples 目錄中的現有範例，建立新的設定檔。範例設定檔是一個範本，可讓您用來建立可以修改並儲存為自訂檔案的設定檔。

若要建立內含 AIX Runtime Expert 支援的所有參數的設定檔，請完成下列步驟：

1. 配置並調整系統，使其具有新設定檔想要的設定。
2. 前往 samples 目錄：/etc/security/artex/samples
3. 執行下列指令，以建立新的設定檔 custom_all.xml：

```
artexget -p all.xml > /directory_for_new_profile/custom_all.xml
```

註：custom_all.xml 設定檔可以用來配置其他具有與目前系統配置類似的系統。

若要建立特定元件（例如網路選項）的設定檔，請完成下列步驟：

1. 配置並調整系統，使其具有新設定檔想要的設定。
2. 前往 samples 目錄：/etc/security/artex/samples。
3. 執行下列指令，以透過現有範例設定檔 noProfile.xml 建立新的設定檔 custom_no.xml：

```
artexget -p noProfile.xml > /directory_for_new_profile/custom_no.xml
```

使用 XML 編輯器或任何文字編輯器來變更或移除參數值，即可自訂新建立的設定檔。

自訂設定檔可以上傳至 LDAP 伺服器，以從多個 AIX 系統使用。若要將設定檔上傳至 LDAP 伺服器，請使用 LDAP 所提供的工具。

相關概念

AIX Runtime Expert 及 LDAP
AIX Runtime Expert 可以從「輕量型目錄存取通訊協定 (LDAP)」伺服器擷取設定檔。

AIX Runtime Expert 設定檔
AIX Runtime Expert 設定檔可用來設定執行中系統上的值、擷取執行中系統的值，以及將值與執行中系統或另一個設定檔進行比較。

相關工作

取得 AIX Runtime Expert 設定檔值
使用 **artexget** 指令，可以尋找設定檔的相關資訊。

套用 AIX Runtime Expert 設定檔
若要使用設定檔中的配置及可調整參數來設定系統，請使用 **artexset** 指令來套用設定檔。

相關資訊

artexget 指令

修改 *AIX Runtime Expert* 設定檔

AIX Runtime Expert 設定檔是 XML 檔案，而且可以使用任何 XML 編輯器或任何文字編輯器進行修改。

變更參數值，或是移除部分不再需要修改或監視設定檔的參數，即可自訂使用者使用 **artexget** 指令建立的設定檔。

若要修改 AIX Runtime Expert 設定檔，請完成下列步驟：

1. 從 `custom_all.xml` 所在的目錄，執行下列指令，以儲存設定檔的副本：

   ```
   cp custom_all.xml custom_all_backup.xml
   ```

2. 從 `custom_all.xml` 所在的目錄，執行下列指令，以編輯設定檔：

   ```
   vi custom_all.xml
   ```

   註：您可以使用任何 XML 編輯器或文字編輯器。
3. 修改參數值，或是移除不再需要變更或監視設定檔的參數。
4. 執行下列指令，比較設定檔變更與目前系統設定，來驗證已正確儲存設定檔變更：

   ```
   artexdiff -c -r custom_all.xml custom_all_backup.xml
   ```

   artexdiff 指令會顯示編輯器所修改的參數。`<FirstValue>` 會顯示設定檔的值，而 `<SecondValue>` 則顯示目前系統的值。

相關概念

AIX Runtime Expert 設定檔
AIX Runtime Expert 設定檔可用來設定執行中系統上的值、擷取執行中系統的值，以及將值與執行中系統或另一個設定檔進行比較。

相關工作

取得 AIX Runtime Expert 設定檔值
使用 **artexget** 指令，可以尋找設定檔的相關資訊。

套用 AIX Runtime Expert 設定檔
若要使用設定檔中的配置及可調整參數來設定系統，請使用 **artexset** 指令來套用設定檔。

相關資訊

artexdiff 指令

結合 *AIX Runtime Expert* 設定檔

設定檔可以代表一組完整控制項或任何控制項的子集。其他修改設定檔的有用方式，是使用 **artxmerge** 指令來合併代表控制項子集的設定檔。

您可以使用 **artexmerge** 指令，將一個以上設定檔合併為單一設定檔。

若要合併設定檔，請完成下列步驟：

1. 從設定檔的儲存目錄，執行下列指令：

```
artexmerge profile_name1.xml profile_name2.xml > new_profile_name.xml
```

2. 執行下列指令，以檢視設定檔，並驗證它是有效的設定檔：

```
 artexget new_profile_name.xml
```

註 : 如果合併的設定檔具有重複的參數，則合併設定檔的處理程序會失敗。或者，如果您使用 **-f** 旗標，則會使用最新設定檔的參數值。

相關資訊

artexmerge 指令

尋找 *AIX Runtime Expert* 設定檔

使用 **artexlist** 指令，可以在給定路徑及 LDAP 伺服器中尋找設定檔。

若要尋找設定檔，請完成下列步驟：

1. 如果設定檔位於本端系統，請執行下列指令：

```
artexlist
```

2. 如果設定檔位於 LDAP 伺服器，請執行下列指令：

```
artexlist -l
```

此指令預設會列出 /etc/security/artex/samples 目錄中的設定檔。若要以環境變數置換預設路徑，請將 **ARTEX_PROFILE_PATH** 設定為一個以上以分號區隔的路徑，或是可以作為引數傳遞的路徑。

相關資訊

artexlist 指令

取得 *AIX Runtime Expert* 設定檔值

使用 **artexget** 指令，可以尋找設定檔的相關資訊。

使用設定檔，您可以透過不同的過濾器（例如需要重新開機才能生效的參數，以及需要停止並重新啟動一些服務的參數）來顯示設定檔或系統中不同格式的值（XML、CSV 或文字）。

在下列情況下，取得系統中的值十分實用：

建立系統的 Snapshot

正確配置系統之後，可以透過建立 Snapshot 來儲存系統的配置。之後，如果變更任一參數，但您不記得變更的是哪個參數，即可使用此 Snapshot。Snapshot 設定檔可以用來將系統回復為想要的配置。

若要複製系統的配置，以用於其他系統

在環境中配置並調整系統之後，即可將系統設定擷取至 AIX Runtime Expert 設定檔，並將設定檔套用至其他系統。

若要除錯問題

在正式作業系統上發現問題時，您可以使用設定檔，在測試系統上設定相同的系統設定，然後在測試系統上除錯問題。

若要取得設定檔的相關資訊，請完成下列步驟：

1. 前往想要取得其相關資訊的設定檔的所在目錄。
2. 若要取得設定檔的相關資訊，請執行下列指令：

```
artexget name_of_profile.xml
```

限制 : 當系統已定義多個使用者時，套用至設定檔（例如，chuserProfile.xml、coreProfile.xml 或 all.xml 設定檔）的 AIX Runtime Expert 指令 **artexget**、**artexset** 及 **artexdiff**，需要比平常更多的時間才能完成。

相關概念

AIX Runtime Expert 設定檔

AIX Runtime Expert 設定檔可用來設定執行中系統上的值、擷取執行中系統的值，以及將值與執行中系統或另一個設定檔進行比較。

相關工作

建立 AIX Runtime Expert 設定檔
利用 **artexget** 指令，並使用 /etc/security/artex/samples 目錄中的現有範例，建立新的設定檔。範例設定檔是一個範本，可讓您用來建立可以修改並儲存為自訂檔案的設定檔。

修改 AIX Runtime Expert 設定檔
AIX Runtime Expert 設定檔是 XML 檔案，而且可以使用任何 XML 編輯器或任何文字編輯器進行修改。

相關資訊

artexget 指令

套用 *AIX Runtime Expert* 設定檔

若要使用設定檔中的配置及可調整參數來設定系統，請使用 **artexset** 指令來套用設定檔。

若要套用使用者建立的設定檔，請完成下列步驟：

1. 前往想要套用的設定檔的儲存目錄。
2. 若要將設定檔套用至系統，請執行下列指令：

   ```
   artexset -c name_of_profile.xml
   ```

3. 選擇性的: 如果您想要在每次系統重新啟動時套用設定檔，以維護一致的配置，請執行下列指令：

   ```
   artexset -b name_of_profile.xml
   ```

   註 : 支援受限參數作為唯讀參數。 因此，這些參數的值可以使用 **artexget** 指令進行擷取，但無法使用 **artexset** 指令進行設定。

相關概念

AIX Runtime Expert 設定檔
AIX Runtime Expert 設定檔可用來設定執行中系統上的值、擷取執行中系統的值，以及將值與執行中系統或另一個設定檔進行比較。

相關工作

建立 AIX Runtime Expert 設定檔
利用 **artexget** 指令，並使用 /etc/security/artex/samples 目錄中的現有範例，建立新的設定檔。範例設定檔是一個範本，可讓您用來建立可以修改並儲存為自訂檔案的設定檔。

修改 AIX Runtime Expert 設定檔
AIX Runtime Expert 設定檔是 XML 檔案，而且可以使用任何 XML 編輯器或任何文字編輯器進行修改。

相關資訊

artexset 指令

回復 *AIX Runtime Expert* 設定檔

使用 **artexset -u** 指令，可以將配置設定重設為系統的先前配置設定。您可以套用在套用設定檔之前所使用的系統設定。

如果您在現行階段作業期間都未變更過系統設定，則無法使用 **rollback** 指令。

回復並不會視為重新建立作業系統映像檔。當您使用 **rollback** 指令時，並不會刪除或建立資源，而是改為將執行時期配置值回復為系統先前的設定。利用 **rollback** 指令，無法回復至特定時間或日期的設定。您只能回復至進行變更之前的系統先前設定。

rollback 指令可用於下列情況：

- 測試系統的配置變更。如果新配置運作欠佳，則可以快速恢復至先前信任的配置。
- 除錯系統。如果系統執行效能開始變差，則回復可以確認配置變更是否影響到最近偵測到的問題。
- 實作新的設定檔，以滿足部分特殊例外狀況。例如，指定的動作在系統上一個月只會發生一次，以及在將它套用至您的系統之後，想要將系統還原回其先前的配置。

若要回復至先前的系統設定，請完成下列步驟：

1. 若要回復設定檔，請執行下列指令：

```
artexset -u
```

2. 若要驗證回復已正確完成，請執行下列指令以比較系統設定：

```
artexdiff -f txt -r -profile_name.xml
```

註： *profile_name.xml* 是套用至系統的最新設定檔名稱。

即會顯示系統與設定檔之間的差異。

相關資訊
artexget 指令
artexlist 指令

比較 *AIX Runtime Expert* 設定檔

使用 **artexdiff** 指令，可以比較兩個設定檔，或是比較設定檔值與系統值。

若要比較兩個不同系統的設定檔，請完成下列步驟：

1. 從系統 1，執行下列指令：

```
artexget -p all.xml > all_system1.xml
```

2. 從系統 2，執行下列指令：

```
artexget -p all.xml > all_system2.xml
```

一段時間之後，若要驗證是否變更系統上的任何配置參數（例如，如果您休假，而且想要驗證您離開時進行的所有變更），請執行下列指令：

· 在您休假回來之後，請執行下列指令：

```
$ artexget -p all.xml > all_before_vacation.xml
```

· 若要檢視休假期間進行的所有配置變更，請執行下列指令：

```
$ artexdiff -c -p all_before_vacation.xml
```

相關資訊
artexget 指令
artexlist 指令

撰寫 AIX Runtime Expert 設定檔

您可以藉由新增程式可以使用的型錄和設定檔，來展開 AIX Runtime Expert 的範圍。在您嘗試寫入新型錄之前，必須先熟悉 AIX Runtime Expert 概念。

由 AIX Runtime Expert 所處理的最小的資訊片段即為參數。參數可以是可調整的配置檔、環境變數、物件內容（例如，使用者、裝置或子系統，這類物件在 AIX Runtime Expert 環境定義中稱為目標）。

參數會根據活動的網域（例如使用者、tcpi）聚集至設定檔中。設定檔是使用者和 AIX Runtime Expert 架構之間所預期的互動方式。設定檔是 **artexget** 指令的輸入，以在系統上擷取參數值並傳回設定檔。設定檔（包括值）是 **artexset** 指令的輸入，以將參數設為讀取至設定檔的值。

撰寫 *AIX Runtime Expert* 設定檔的概念

AIX Runtime Expert 設定檔為包含配置參數清單及（選擇性地）參數值和用法旗標的 XML 檔案。

直接在指令行上使用 AIX Runtime Expert 指令時，可以在所調整的系統上找到設定檔。

設定檔位置
AIX Runtime Expert 範例設定檔位於目錄 `/etc/security/artex/samples` 中。

當您寫入要讓 AIX Runtime Expert 支援的新型錄時，建議您也寫入範例設定檔，其可用來作為 **artexget** 指令的項目。範例設定檔是沒有指派任何值給參數的唯讀設定檔。 現有範例設定檔位於 /etc/security/artex/samples 目錄中。 依預設，**artexlist** 指令只會列出位於預設目錄的設定檔，但預設目錄可藉由設定 **ARTEX_PROFILE_PATH** 環境變數來修改。可藉由使用 : 分隔字元在此環境變數中指定多個目錄。

範例目錄的所有設定檔在 **artex.base.samples** 檔案集安裝期間會進行合併，以形成由 **snap** 指令所使用的 **default.xml** 設定檔。不應該在範例目錄中遞送不應為 **default.xml** 設定檔一部分的設定檔。不應該併入在 **default.xml** 設定檔的設定檔範例為具有潛力併入數千個參數（例如，如果其以使用者作為目標類別）的設定檔，以及應該只在特定系統（例如，**vios** 屬性設定檔）上執行的設定檔。

設定檔命名
AIX Runtime Expert 設定檔根據指令來命名。

設定檔的建置通常是根據單一指令或指令集。如果型錄密切相關，則設定檔可能會包含數個型錄。慣例是根據指令來命名檔案，例如，針對範例設定檔為 **commandProfile.xml**，而針對型錄為 **commandParam.xml**，不過這不是必要項目。只有 **.xml** 副檔名是必要項目。

設定檔處理程序
討論寫入新 AIX Runtime Expert 設定檔的處理程序。

寫入新 AIX Runtime Expert 設定檔時，需要執行下列步驟：

1. 在設定檔中製作您所要的參數清單。
2. 針對每一個參數，建立 **<Parameter name="..." >** 元素，並將 *name* 屬性設為編目檔 **<ParameterDef>** 元素中的名稱。
3. 將相同 **<Catalog id="..." >** 元素內相同編目檔中所定義的所有參數進行分組，並將 *id* 屬性設為編目檔 **<Catalog>** 元素中使用的相同 ID。
4. 針對每一個 **<Parameter>** 元素，請執行下列動作：
   a. 如果在編目檔中使用 *reboot*=true 定義參數，請新增 *reboot*=true 及 *applyType*=nextboot 屬性。
   b. 如果只能擷取而無法設定參數，請新增 readOnly=true 屬性。
   c. 如果在編目檔中使用非空的 *targetClass* 屬性定義參數，請執行下列動作：
      1) 如果此參數需要進行目標探索，則請針對此參數定義單一 **<Parameter>** 元素，並針對此元素使用特殊 **<Target class="" instance="" >** 目標。
      2) 如果需要針對此參數定義特殊目標，則針對每一個目標定義一個 **<Parameter>** 元素。在每一個 **<Parameter>** 元素下，定義適當的 **<Target class="..." instance="..." />** 元素，以完整地指定目標。
5. 執行 **artexget –r** 指令來測試設定檔。

AIX Runtime Expert 設定檔元素

<Profile> 元素
<Profile> 元素是所有設定檔的根元素。

語法

支援下列屬性：

表 *2.* 屬性

屬性	必要	類型	說明
origin	否	字串	設定檔的原點。
date	否	日期時間	設定檔的建立日期或前次修改日期。格式為 YYYY-MM-DDThh:mm:ss。

表 2. 屬性 (繼續)			
屬性	**必要**	**類型**	**說明**
readOnly	否	布林	告知此設定檔是否可以用於 set 作業。預設值為 `false`。
version	否	字串	設定檔的版本號碼。

支援下列子元素：

表 3. 子元素			
子元素	**必要**	**號碼**	**說明**
<ShortDescription>	否	0 - 1	型錄的簡短文字說明。
<Description>	否	0 – 1	型錄的詳細文字說明。
<Comments>	否	0 – 1	使用者提供的註解。
<Catalog>	否	0 – 任何	在設定檔上處理作業所需的型錄。

屬性

origin 屬性

origin 屬性是可以指派下列值的參考資訊屬性：

· 建立範例設定檔時，*origin* 屬性必須設為 `reference`。

· 使用 **artexget** 指令建立設定檔時，*origin* 屬性會自動設為 `get`。

子元素

<Comments> 元素是選用字串，保留作為其他目的使用。手動建立設定檔時不得使用此元素，而且基本 AIX Runtime Expert 指令不會使用此元素。

範例

1. 空的範例設定檔如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<Profile origin="reference" version="2.0.0" readOnly="true">
</Profile>
```

2. **artexget -r /etc/security/artex/samples/smtctmProfile.xml** 指令會輸出類似於下列範例的設定檔：

```
<?xml version="1.0" encoding="UTF-8"?>
<Profile origin="get" version="2.0.1" date="2010-09-29T07:50:56Z">
  <Catalog id="smtctlParam" version="2.0">
    <Parameter name="enableSMT" value="1"/>
  </Catalog>
</Profile>
```

相關資訊

<Catalog> 元素

<Description> 及 <ShortDescription> 元素。

<Description> 及 *<ShortDescription>* 元素
可以使用 **<Description>** 及 **<ShortDescription>** 元素來提供設定檔及參數的文字說明。

語法

<ShortDescription> 元素的母元素是：

· **<Profile>** 元素

<Description> 元素的母元素可以是：

· **<Profile>** 元素

· **<Parameter>** 元素

<Description> 及 **<ShortDescription>** 元素具有相同的格式。**<Description>** 元素所包含的文字是 XML 標籤的字串內容。

用法

AIX Runtime Expert 架構目前未使用設定檔中的說明。AIX Runtime Expert 指令會忽略任何併入輸入設定檔中的註解。

範例

下列是 **<Description>** 及 **<ShortDescription>** 元素的範例：

```
<ShortDescription>
   Short summary of field contents.
</ShortDescription>
<Description>
This text field can be used to display in full detail the use of the parent element.
</Description>
```

相關資訊

<Profile> 元素。

<Parameter> 元素。

<Catalog> 元素
<Catalog> 元素指出包含子 **<Parameter>** 元素定義的編目檔名稱。

語法
母元素：**<Profile>**

支援下列屬性：

表 *4.* 屬性

屬性	**必要**	**類型**	**說明**
id	是	字串	指定型錄 ID。此名稱在系統上應該是唯一的。
version	否	字串	指定用來建置此設定檔的型錄版本。

支援下列子元素：

表 *5.* 子元素

子元素	**必要**	**號碼**	**說明**
<Parameter>	否	0 – 任何	併入此型錄的參數。
<SubCat>	否	0 – 任何	併入此型錄的子種類。
<Seed>	否	0 – 任何	併入此型錄的種子。

屬性

id 屬性

id 屬性必須設為型錄名稱，其定義列於 **<Catalog>** 元素下的參數。*id* 屬性是磁碟上編目檔的基本名稱（已移除 **.xml** 副檔名）。例如，設定檔會使用 **<Catalog id=“commandParam”>** 元素來參照 `commandParam.xml` 編目檔。

依預設，會在 `/etc/security/artex/catalogs` 目錄下搜尋編目檔。然而，可藉由設定 **ARTEX_CATALOG_PATH** 環境變數來搜尋其他目錄（僅適用於 root 使用者）。可使用 : 分隔字元在此環境變數中指定多個目錄。

version 屬性

version 屬性以 *MM.mm* 的形式寫入，其中 *MM* 為主要號碼，而 *mm* 為次要號碼。

version 屬性必須與所參照的編目檔版本相符（請參閱撰寫 AIX Runtime Expert 型錄一節中的<Catalog>元素）。如果 AIX Runtime Expert 指令在參照型錄的設定檔上執行，而且具有不正確的版本，則會顯示下列警告訊息：

```
0590-218  型錄版本與設定檔中所參照的版本不同
本端型錄版本為 '2.1'。用來建置設定檔的版本為 '2.0'
```

用法

<Catalog> 元素會識別編目檔，此檔案包含了已列出的種子和參數的定義。設定檔中的所有種子和參數元素都必須位於適當的 **<Catalog>** 元素中。

設定檔可能會參照數個型錄。例如，**default.xml** 設定檔是透過合併其他範例設定檔的選取集，在 `artex.base.sample` 檔案集安裝期間所建置的。

範例

安全屬性設定檔 `secattrProfile.xml` 會使用三個型錄，每一個型錄處理其中一個安全表格：

```
<Profile origin="reference" readOnly="true" version="2.0.0">
  <Catalog id="privcmdParam" version="2.0"
    <Parameter name="privatecommands" />
  </Catalog>
  <Catalog id="privdevParam" version="2.0">
    <Parameter name="privatedevices"/>
  </Catalog>
  <Catalog id="privfileParam" version="2.0">
    <Parameter name="privatefiles" />
  </Catalog>
</Profile>
```

相關資訊

<Catalog> 元素（在編目檔中）。

<SubCat> 元素

<SubCat> 元素提供在 **<Catalog>** 元素下建立邏輯子種類的方法。

語法

母元素：**<Catalog>**、**<SubCat>**

支援下列屬性：

表 6. 屬性			
屬性	**必要**	**類型**	**說明**
id	是	字串	指定子種類名稱。此名稱在相同 **<Catalog>** 元素中應該是唯一的。

支援下列子元素

表 7. 子元素			
子元素	**必要**	**號碼**	**說明**
<Parameter>	否	0 – 任何	包含參數名稱。
<SubCat>	否	0 – 任何	巢狀子種類

屬性

id 屬性可唯一識別型錄內的子種類。設定檔可能會包含數個具有相同 ID 的子種類，但前提是在相同的 **<Catalog>** 元素下未使用這些子種類。

子元素

<SubCat> 元素可能具有另一個 **<SubCat>** 作為子元素。您可以定義的巢狀子種類數目沒有限制。

用法

子種類只會基於可讀性而進行併入。它們不會影響處理參數的方式。

範例

noProfile.xml 設定檔包含數個子種類。下列為範例：

```
<Profile origin="reference" readOnly="true" version="2.0.0">
  <Catalog id="noParam" version="2.0">
    <SubCat id="general_network"
      <Parameter name="fasttimo"/>
      <Parameter name="nbc_limit"/>
    </SubCat>
    <SubCat id="tcp_network">
      <Parameter name="clean_partial_conns"/>
      <Parameter name="delayack"/>
    </SubCat>
    <SubCat id="restricted">
      <Parameter name="extendednetstats" readOnly="true"/>
      <Parameter name="inet_stack_size" readOnly="true"/>
    </SubCat>
  </Catalog>
</Profile>
```

相關資訊

<Parameter> 元素。

<Parameter> 元素
<Parameter> 元素會定義配置參數。

語法

支援下列屬性：

表 *8.* 屬性

屬性	**必要**	**類型**	**說明**
名稱	是	字串	指定參數名稱。此名稱在型錄內必須是唯一的。
value	否	字串	如果已定義，則為參數值。
applyType	否	字串	指定是否必須擷取或設定參數的執行時期或下次開機值（如果沒有指定旗標）。 預設值為 `runtime`。
reboot	否	布林	若為 `true`，指出在值變更生效之前需要重新開機。預設值為 `false`。
readOnly	否	布林	指定是否無法設定參數值。預設值：值為 `false`。
disruptive	否	布林	指定用來設定參數的方法是否默示干擾限制。預設值為 `false`。
setDiscover	否	布林	指定是否必須針對所有目標類別的已探索實例，將 set 方法設為值。預設值為 `false`。

支援下列子元素

表 *9.* 子元素

子元素	**必要**	**號碼**	**說明**
<Value>	否	0 - 1	參數值。
<Target>	否	0 - 任何	參數套用至的目標。
<Description>	否	0 - 1	參數的說明。
<Property>	否	0 - 任何	參數的內容。

屬性

表 *10.* 屬性

屬性	**說明**
名稱	參數名稱是 **<Parameter>** 元素的唯一必要屬性。與母項 **<Catalog>** 元素中所指定的型錄名稱相同，參數名稱可唯一識別編目檔中的參數定義。
value	可以指定參數值作為屬性或子元素。

表 *10.* 屬性 *(*繼續*)*

屬性	說明
applyType	*applyType* 屬性可採用 `runtime`（預設值）或 `nextboot` 值。 此屬性可決定用來處理參數的指令。 若為 set 作業，*applyType*=`runtime` 指出應該使用編目檔中的 **<Set type=“permanent”>** 指令來設定參數。 *applyType*=`nextboot` 指出應改用 **<Set type=“nextboot”>** 指令。 若為 get 作業，使用 *–p* 旗標時，*applyType*=`runtime` 指出必須使用編目檔中的 **<Get type=“current”>** 指令來取得參數。 <*applyType*>=`nextboot` 指出應改用 **<Get type=“nextboot”>** 指令。 如果 *reboot* 屬性已設為 true，則 *applyType* 屬性必須設為 `nextboot`。

reboot	此屬性的預設值為 false。 設為 true 時，表示系統必須在參數的任何變更生效之前重新開機。此屬性必須符合編目檔的相對應 **<ParameterDef>** 元素中定義的 *reboot* 屬性。
	
	當此屬性設為 true 時，*applyType* 屬性必須設為 `nextboot`。
	
	當您設定具有 *reboot* 屬性集的參數時，會向使用者顯示警告，告知他們需要進行重新開機作業：
	
	`0590-206 需要手動後置作業，變更才會生效`
	`請將系統重新開機`
	
	當參數值的變更在下次重新開機後才會生效時，只將 *reboot* 屬性設為 true。

readOnly	此屬性指出參數值可以由 **artexget** 指令讀取， 但是無法使用 **artexset** 指令來設定，也無法使用 **artexdiff** 指令，將其列入考慮以進行使用中的值的比較作業。預設值為 false。 部分可以授權將 *readOnly* 屬性設為 true 的狀況如下： · 參數為靜態，而且無法變更其值（例如，**vmo** 指令中的 *memory_frames* 參數）。 · 參數的存取受限，但不建議使用者在自動程序中修改該參數。在此情況下，應於編目檔中定義此參數的 set 配置方法，但系統管理者必須從可以設定參數的設定檔中手動移除 *readOnly* 屬性。

表 *10.* 屬性 *(*繼續*)*	
屬性	**說明**
setDiscover	當 *setDiscover* 屬性設為 true 時，表示如果呼叫 **artexset** 指令及 *–d* 旗標，則必須呼叫 **discover** 指令，以探索目標的所有實例，並設定儲存在設定檔中的所有值。 *setDiscover* 預設值為 false。只有在參數具有編目檔中定義的目標類別時，值才能為 true。 建立範例設定檔時，請勿指定此屬性。進階的使用者將會視需要手動新增此屬性。

其他屬性

type 及 *disruptive* 屬性為呼叫 **artextget** 指令與 **-i** 旗標時，該指令所自動設定的資訊屬性。建立範例設定檔時，請勿併入這些屬性。

範例

1. 下列範例摘錄自 vmoProfile.xml 範例設定檔，並說明各種選用屬性的使用：

```
<Profile origin="reference" readOnly="true" version="2.0.0">
  <Catalog id="vmoParam" version="2.1">
    <Parameter name="nokilluid"/>
    <Parameter name="memory_frames" readOnly="true"/>
    <Parameter name="kernel_heap_psize" reboot="true" applyType="nextboot"/>
  </Catalog>
</Profile>
```

2. 如果您在範例 1 中的設定檔上執行 **artexget –r** 指令，則會顯示下列設定檔：

```
<Profile origin="get" version="2.0.1" date="2011-03-24T13:41:01Z">
  <Catalog id="vmoParam" version="2.1">
    <Parameter name="nokilluid" value="0"/>
    <Parameter name="memory_frames" value="393216" readOnly="true"/>
    <Parameter name="kernel_heap_psize" value="4096" applyType="nextboot" reboot="true"/>
  </Catalog>
</Profile>
```

相關資訊

參數值主題。

<ParameterDef> 元素。

參數值
參數值可以在設定檔中設定為屬性（如果參數值夠短），或設定為 **<Parameter>** 元素的子元素。

使用

寫入範例設定檔時，不得將值指派給參數。參數值（如果存在的話）會自動併入由執行中 **artexget** 指令取得的設定檔中。

runtime 及 nextboot 值

runtime 及 nextboot 值的概念是 AIX Runtime Expert 架構很重要的一部分。

參數的 runtime 值是其在 **artexget** 指令執行於系統上擷取的現行值。 nextboot 值是參數在下次系統重新開機之後將具有的值。

例如，在設定檔 sysdumpdevProfile.xml 中的參數 *type_of_dump*。 此參數的現行 (runtime) 值可能為 traditional 或 firmware-assisted。 如果此值已變更（使用 **artexset** 指令或直接使用

sysdumpdev 指令），則在下次重新開機之前此值都是無效的。此參數的 nextboot 值即會成為已修改的值。

```
<Parameter name="type_of_dump" applyType="nextboot" reboot="true" />
```

範例

下列範例顯示值指定為屬性的參數，以及另一個值指定為子元素的參數：

```
<Profile origin="get" version="2.0.1" date="2010-09-28T12:30:03Z">
<Catalog id="login.cfgParam" version="2.0">
<Parameter name="shells">
<Value>
/bin/sh,/bin/bsh,/bin/csh,/bin/ksh,/bin/tsh,
/bin/ksh93,/usr/bin/sh,/usr/bin/bsh,/usr/bin/csh,
/usr/bin/ksh,/usr/bin/tsh,/usr/bin/ksh93,
/usr/bin/rksh,/usr/bin/rksh93,
/usr/sbin/uucp/uucico,/usr/sbin/sliplogin,
/usr/sbin/snappd
</Value>
</Parameter>
<Parameter name="maxlogins" value="32767"/>
</Catalog>
</Profile>
```

<Property> 元素
<Property> 元素會將值指派給參數內容。

語法
母元素：**<Parameter>**

支援下列屬性：

表 *11.* 屬性

屬性	**必要**	**類型**	**說明**
名稱	是	字串	指定內容的名稱。
value	否	字串	指定內容的值。

用法

<Property> 元素會將值指派給母元素的內容名稱。在產生指令行期間展開 **%p***[name]* 序列時，會使用此值。

通常不會以手動方式將 **<Property>** 元素新增至設定檔。會根據編目檔的相對應 **<Property>** 元素下所定義的指令，在執行 **artexget -r** 及 **artexget -n** 指令時，自動將此元素插入到輸出設定檔中。

範例

下列範例會設定 netaddr 參數的 **nodeId** 內容。內容值是透過 **artexget -r** 指令來擷取，而且是 **uname -f** 指令的輸出：

```
<Parameter name="netaddr" value="172.16.128.13">
 <Target class="device" instance="en0"/>
 <Property name="nodeId" value="8000108390E00009"/>
</Parameter>
```

<Seed> 元素
<Seed> 元素會定義一個種子，而該種子會在 **<Get>** 作業期間展開至一個以上的 **<ParameterDef>** 元素中。

語法
母元素：**<Catalog>**

支援下列屬性：

表 *12.* 屬性			
屬性	**必要**	**類型**	**說明**
名稱	是	字串	指定符合編目檔中的 **SeedDef** 元素的種子元素名稱。

支援下列子元素：

表 *13.* 子元素			
子元素	**必要**	**號碼**	**說明**
<Parameter>	否	0 – 任何	根據參數的名稱，過濾探索到的參數。
<Target>	否	0 – 任何	根據其目標，過濾探索到的參數。

用法

<Seed> 元素會在 **<Get>** 作業期間動態地探索參數。

發出 **artexget** 指令時，會將輸入設定檔中的每一個 **<Seed>** 元素展開至一個以上的 **<Parameter>** 元素中。 會根據編目檔的相符 **<SeedDef>** 元素中所定義的規則來展開設定檔。此處理程序稱為參數探索。在完成參數探索處理程序之後，**artexget** 指令會使用展開的設定檔如常處理。

選用的 **<Parameter>** 及 **<Target>** 子元素是用來過濾探索到的參數。如果探索到的參數與 **<Parameter>** 子元素中所定義的準則不符，則會捨棄這些參數。 此外，如果適用於目標的那些參數與 **<Target>** 子元素中定義的準則不符，也會捨棄那些參數。

範例

此範例會使用 **devSeed** 型錄來定義一個種子，而該種子可用來探索所有裝置的所有屬性：

```
<?xml version="1.0" encoding="UTF-8"?>
<Catalog id="devSeed" version="3.0">
  <SeedDef name="devAttr">
    <Discover>
      <Command>
        /usr/sbin/lsdev -F 'name class subclass type' |
        while read DEV CLASS SUBCLASS TYPE
                    do
          CAT=devParam.$CLASS.$SUBCLASS.$TYPE
          /usr/sbin/lsattr -F attribute -l $DEV |
          while read PAR
                      do
            echo "device=$DEV $CAT $PAR"
          done
                done
              </Command>
      Mask target="1" catalog="2" name="3">(.*) (.*) (.*)</Mask>
    </Discover>
  </SeedDef>
</Catalog>
```

下列設定檔可用來探索所有支援的裝置的所有支援的屬性：

```
<?xml version="1.0" encoding="UTF-8"?>
<Profile>
  <Catalog id="devSeed" version="3.0">
    <Seed name="devAttr"/>
  </Catalog>
</Profile>
```

2. 使用相同的型錄，**<Target>** 過濾器可用來探索所有乙太網路配接卡的所有支援的屬性：

```
<?xml version="1.0" encoding="UTF-8"?>
<Profile>
  <Catalog id="devSeed" version="3.0">
    <Seed name="devAttr">
      <Target class="device" match="^en[0-9]+$"/>
    </Seed>
  </Catalog>
</Profile>
```

3. 您可以新增 **<Parameter>** 過濾器，只擷取所有乙太網路配接卡的 **netaddr**、**netaddr6**、**alias** 及 **alias6** 屬性：

```
<?xml version="1.0" encoding="UTF-8"?>
<Profile>
  <Catalog id="devSeed" version="3.0">
    <Seed name="devattr">
      <Parameter match="^(netaddr|alias)6?$"/>
      <Target class="device" match="^en[0-9]+$"/>
    </Seed>
  </Catalog>
</Profile>
```

<Target> 元素
<Target> 元素會定義參數可套用至的目標類別實例。

語法

母元素：**<Parameter>**

只有在參數可套用至其目標的不同實例時，才容許相同型錄中多次出現相同的參數。

支援下列屬性：

表 *14.* 屬性

屬性	必要	類型	說明
class	是	字串	指定目標類別的名稱。
instance	否*	字串	指定類別實例的名稱。
match	否*	字串	指定如套用至已探索實例名稱的正規表示式。

* 必須只指定 *instance* 及 *match* 屬性其中一個。

使用

部分參數無法套用至整個系統，不過可套用至特定物件。此範例是 `chuserProfile.xml` 設定檔中所指定使用者的 home 目錄；此參數可套用至特定可載入模組中（檔案、LDAP）的特定的使用者。 在此範例中，使用者及模組為兩個目標類別。*home* 參數可套用至這些目標類別的特定實例。例如，使用者類別的訪客實例，以及模組類別的檔案實例。

如果 *class* 及 *instance* 屬性都設為空字串，則會在這類設定檔上執行 **artexget** 指令時針對此參數執行探索、執行相對應編目檔中宣告的 discovery 方法，以及在參數的每一個已探索實例的輸出設定檔中建立參數。 請參閱範例 1。

如果已指定 *class* 及 *instance* 屬性，則目標是完整的，而且參數只會套用至目標類別的特定實例。請參閱範例 2。

如果已指定 *class* 及 *match* 屬性，則探索會如上述所執行，但是只會探索所含名稱符合 *match* 屬性中所指定正規表示式的目標實例。請參閱範例 3。

撰寫範例設定檔時，*class* 及 *instance* 屬性必須保留為空的。 這表示遇到空的目標類型時，**artexget** 指令會在擷取值之前，探索該目標類別的實例清單（系統上所有使用者或子系統）。

在未探索目標類別上執行 **artexset** 指令會顯示一則警告：

```
0590-216 設定檔中的部分參數需要進行目標探索，將忽略那些參數
```

範例

1. 探索前具有目標的設定檔範例為定義使用者起始目錄的 chuserProfile.xml 設定檔。下列範例顯示範例設定檔：

```
<Profile version="2.0.0" origin="reference" readOnly="true">
  <Catalog id="chuserParam" version="2.0">
    <Parameter name="home">
      <Target class="" instance=""/>
    </Parameter>
  </Catalog>
</Profile>
```

2. 探索之後，chuserProfile.xml 設定檔會在每一個已探索的可載入模組中，包含每一個已探索使用者的起始參數副本：

```
<Profile version="2.0.0" origin="get">
  <Catalog id="chuserParam" version="2.0">

    <Parameter name="home" value="/">
      <Target class="user" instance="root"/>
      <Target class="module" instance="files"/>
    </Parameter>

    <Parameter name="home" value="/etc">
      <Target class="user" instance="daemon"/>
      <Target class="module" instance="files"/>
    </Parameter>

    ...

  </Catalog>
</Profile>
```

3. 下列設定檔使用 *match* 屬性來探索檔案模組中以 *u* 為名稱開頭的所有使用者起始目錄：

```
<Profile version="2.0.0" origin="reference" readOnly="true"
  <Catalog id="chuserParam" version="2.0">
    <Parameter name="home">
      <Target class="user" match="^u"/>
      <Target class="module" instance="files"/>
    </Parameter>
</Catalog>
</Profile>
```

相關資訊

<Discover> 元素（在編目檔中）。

撰寫 AIX Runtime Expert 型錄

AIX Runtime Expert 架構在內部使用編目檔。

編目檔包含配置方法的參數定義和連結資訊，該配置方法說明用來擷取或設定參數值的指令。編目檔為所調整及所配置系統的本端檔案。

AIX Runtime Expert 型錄概念

編目檔包含在參數上執行作業所需的所有資訊，包括定義、使用條件及配置方法。編目檔不得由使用者直接操作，而應該只透過 AIX Runtime Expert 核心引擎使用。

型錄與 AIX Runtime Expert 核心引擎同時安裝在系統上。當新的型錄鏈結至系統上安裝的元件或協力廠商應用程式時，請務必確保其位於已安裝 AIX Runtime Expert 核心引擎的層次中。

型錄位置

AIX Runtime Expert 編目檔儲存在 /etc/security/artex/catalogs 目錄中。

編目檔名稱必須完全符合其 *id* 屬性，而且字尾為 .xml 副檔名。例如，名稱為 commandParam.xml 的型錄必須具有 *id* 屬性值 commandParam。

為了讓參照型錄的設定檔找到其位置，型錄必須在型錄 XML 檔案及設定檔 XML 檔案的 **<Catalog>** 元素中具有相同名稱。依預設，AIX Runtime Expert 核心引擎會在預設目錄 /etc/security/artex/catalogs 中尋找型錄。 可以變更此行為（只適用於 root 使用者），方法是設定 **ARTEX_CATALOG_PATH** 環境變數。可使用 : 分隔字元在此環境變數中指定多個目錄。

型錄處理程序
撰寫新的 AIX Runtime Expert 型錄的步驟。

撰寫新的 AIX Runtime Expert 型錄時，需要執行下列步驟：

1. 在編目檔中製作您所要的參數清單。
2. 針對每一個參數建立 **<ParameterDef>** 元素
3. 如果數個參數針對 **<Get>**、**<Set>**、**<Discover>** 或 **<Diff>** 作業使用相同指令：
   - 在型錄頂端定義 **<CfgMethod>** 元素。
   - 使用 *cfgmethod* 屬性繼承自配置方法。
4. 如果數個參數都需遵循相同限制，請在型錄頂端定義 **<ConstraintDef>** 元素。
5. 針對每一個參數：
   a. 針對每一個參數定義 **<Get type=“current”>** 及 **<Get type=“nextboot”>** 作業，可直接定義於 **<ParameterDef>** 元素下或在 **<CfgMethod>** 元素下參照，或者使用任何一種組合。
   b. 針對每一個參數定義所有支援的 **<Set>** 作業，可直接定義於 **<ParameterDef>** 元素或參照的 **<CfgMethod>** 元素下，或者使用這些可能性的任何組合。
   c. 如果參數需要目標：
      1) 使用 *targetClass* 屬性定義支援的目標類別
      2) 定義 discover 作業，可直接定義於 **<ParameterDef>** 元素或參照的 **<CfgMethod>** 元素下，或者使用這些可能性的任何組合。在大部分的狀況下，discover 方法會定義於配置方法中。
   d. 如果參數需要重新開機以讓變更生效，請新增 *reboot* =true 屬性。
   e. 如果參數需遵循限制，請於 **<ParameterDef>** 元素下定義 **<ConstraintDef>** 元素，或使用 constraint 屬性來參照現有限制。
6. 若要測試編目檔，請執行下列動作：
   a. 以編目檔中定義的所有參數來建立設定檔。
   b. 使用 **artexget -r** 指令來擷取值，並測試 **<Discover>** 及 **<Get>** 作業。
   c. 對結果設定檔使用 **artexset -c -F -R -l all** 指令，來測試 **<Set>** 及 **<Diff>** 作業。
   d. 此外，可以將 **-g 3 -g COMMANDS** 旗標新增至那兩個指令，以取得產生來執行所要求作業的指令行相關資訊。

相關資訊

請參閱 **<Catalog>** 根元素的相關主題。

***AIX Runtime Expert* 型錄元素**

<Catalog> 元素
<Catalog> 元素是所有編目檔的根元素。

語法

支援下列屬性：

表 15. 屬性			
屬性	**必要**	**類型**	**說明**
id	是	字串	指定型錄名稱。此名稱在系統上必須是唯一的。
version	否	字串	指定型錄的版本號碼。
date	否	日期時間	指定建立日期。格式為 *YYYY-MM-DDThh:mm:ss*。
priority	否	整數	指定在 set 方法中與其他型錄相關的型錄執行順序。預設值為 0。
inherit	否	字串	指定要繼承的型錄名稱。

支援下列子元素。*number* 直欄定義容許子項出現多少次：

表 16. 子元素			
子元素	**必要**	**號碼**	**說明**
<ShortDescription>	否	0 – 1	型錄的簡短文字說明。
<Description>	否	0 – 1	型錄的詳細文字說明。
<SubCat>	否	0 – 任何	子種類
<ParameterDef>	否	0 – 任何	包含參數的內容。
<ConstraintDef>	否	0 – 任何	參數限制定義（條件和 disruptive 指令）。
<CfgMethod>	否	0 – 任何	配置方法定義
<PrereqDef>	否	0 – 任何	定義必要條件。
<PropertyDef>	否	0 – 任何	定義內容。
<SeedDef>	否	0 – 任何	定義種子。

屬性

表 17. 屬性	
屬性	**說明**
id	*id* 屬性應該符合編目檔名稱（除去其 `.xml` 副檔名）。 型錄 ID 在設定檔中使用 **<Catalog>** 元素來參照。
priority	需要在其他型錄的 set 方法併入於相同設定檔（例如，複合設定檔 **default.xml**）之前或之後執行特定型錄的 set 方法時，會使用 *priority* 屬性。 型錄的預設優先順序為 0。 規則為當兩個型錄共用相同的優先順序時，其 set 方法會以未定義的順序來執行。如果型錄具有正數值的優先順序，其 set 方法會以遞減優先順序在其他方法之前執行。如果型錄具有負數值的優先順序，其 set 方法會以遞減優先順序在其他方法之後執行。
version	設定檔及型錄中皆具有 *version* 屬性。version 可協助識別設定檔和型錄是否與 AIX Runtime Expert 核心引擎相容，以及是否彼此相容。如需詳細資料，請參閱 *version* 屬性。

表 17. 屬性 (繼續)	
屬性	**說明**
date	*date* 屬性目前不用於 **<Catalog>** 元素。會依將來使用與維護狀況將其併入。
inherit	*inherit* 屬性可指定要繼承的型錄名稱，但不含 .xml 副檔名。繼承的型錄中所定義的所有元素都可在主要型錄中找到，如同已在本端定義它們。

範例

下列是使用 *priority* 屬性的型錄範例。`aixpertParam.xml` 型錄會設定安全選項，而且必須在已設定其他所有型錄之後進行設定。因此，其優先順序會設為高負數值。

```
<Catalog id="aixpertParam" version="2.0" priority="-1000">
```

相關資訊

<ConstraintDef> 元素。

<CfgMethod> 元素。

<Description> 及 <ShortDescription> 元素。

<ParameterDef> 元素。

<SubCat> 元素。

version 屬性

語法

型錄版本以 *MM.mm* 的格式寫入為屬性，其中 *MM* 為主要號碼，而 *mm* 為次要號碼。

```
<Catalog id="commandParam" version="2.0">
```

主要版本號碼

安裝於系統上的所有 AIX Runtime Expert 型錄及整個 AIX Runtime Expert 架構（其參照位置）的主要版本號碼都一樣。此主要號碼會隨著設定檔和型錄的 XML 綱目每一次的重大變更而增加。

建立新型錄時，請將主要版本號碼設為現行 AIX Runtime Expert 核心引擎版本號碼，您可以在隨附於 `artex.base.rte` 檔案集的任何標準編目檔內找到該版本號碼。

如果在設定檔上呼叫 **artexget** 指令，而該設定檔的主要版本號碼不同於 AIX Runtime Expert 核心引擎中所參照的號碼，則指令會失敗並顯示下列錯誤：

```
0590-117 版本錯誤
此設定檔是在不受 ARTEX 支援的版本上建立
```

也建議您讓設定檔和型錄共用相同的主要版本號碼，以使其相容。設定檔會以特定的版本號碼來參照型錄。如果設定檔的主要版本號碼與型錄主要版本號碼不同，則任何 AIX Runtime Expert 指令都會顯示警告，以通知使用者結果可能會無法預期：

```
0590-218  型錄版本與設定檔中所參照的版本不同
```

次要版本號碼

次要版本號碼是每一個型錄的特定號碼，每一次型錄中的重大變更使型錄與舊版不相容時，便會增加該號碼。設定檔會以特定的版本號碼來參照型錄。如果設定檔的次要版本號碼與型錄次要版本號碼不同，則任何 AIX Runtime Expert 指令都會發出警告，以通知使用者結果可能會無法預期：

```
0590-218  型錄版本與設定檔中所參照的版本不同
```

建立新範例設定檔或型錄時，請將次要版本號碼設為 0。

<Description> 及 *<ShortDescription>* 元素
說明是可新增至編目檔中不同元素的選用資訊文字欄位。這些欄位為選用欄位，但建議型錄撰寫者使用它們來記載母元素。

語法

<ShortDescription> 元素的母元素可以是下列其中一項：

- **<Catalog>**
- **<SubCat>**

<Description> 元素的母元素可以是下列其中一項：

- **<Catalog>**
- **<SubCat>**
- **<ParameterDef>**
- **<ConstraintDef>**

<Description> 及 **<ShortDescription>** 元素的內容可為簡式字串，或由 **<NLSCatalog>**、**<NLSSmitHelp>** 或 **<NLSCommand>** 其中一個元素所定義的已轉換訊息。如需相關資訊，請參閱全球化支援主題。

用法

目前只有**<ParameterDef>** 元素的說明是由 **artexget** 指令及 **-i** 旗標來擷取及顯示。建議您針對那些說明欄位中所包括的文字提供全球化的內容。

AIX Runtime Expert 架構目前不使用其他元素的說明欄位，但是為了供日後使用以及基於文件用途，仍應提供這些欄位。

範例

1. 以下是說明欄位的簡式範例：

```
<ShortDescription>
   chuser parameters
</ShortDescription>
<Description>
   Parameter definition for the chuser command
</Description>
```

2. 從 artexcat.cat 訊息檔案使用已轉換訊息的相同範例：

```
<ShortDescription>
<NLSCatalog catalog="artexcat.cat" setNum="12" msgNum="1">
  chuser parameters
</NLSCatalog>
</ShortDescription>
<Description>
<NLSCatalog catalog="artexcat.cat" setNum="12" msgNum="2">
  Parameter definition for the chuser command
</NLSCatalog>
</Description>
```

全球化支援
本節說明如何在 AIX Runtime Expert 型錄的敘述性欄位中實作全球化。

語法

母元素：**<Description>**、**<ShortDescription>**

母元素可能包含下列其中一個（而且是唯一的）子元素：

表 *18.* 子元素			
子元素	**必要**	**號碼**	**說明**
<NLSCatalog>	否	0 – 1	訊息型錄中併入的字串
<NLSSmitHelp>	否	0 – 1	SMIT 說明 HTML 檔案中併入的字串
<NLSCommand>	否	0 – 1	AIX 指令所發出的字串

NLS 型錄

在現有訊息型錄中以 **catgets()** 格式併入要顯示的本地化訊息時，會使用「NLS 型錄」全球化格式。

<NLSCatalog> 元素包含下列屬性：

表 *19.* 屬性			
屬性	**必要**	**類型**	**說明**
catalog	是	字串	訊息所在的型錄名稱
setNum	是	整數	訊息所在的訊息集數目
msgNum	是	整數	訊息集中的訊息數目

如果不存在已本地化的訊息型錄，則會改為顯示預設訊息。會選擇性地併入預設訊息，作為 **<NLSCatalog>** 元素的內容。這是提供預設訊息的建議作法。

NLS SMIT 說明

SMIT 說明 HTML 檔案中已存在要顯示的本地化訊息時，會使用「NLS Smit 說明」全球化格式。

<NLSSmitHelp> 元素包含下列屬性：

表 *20.* 屬性			
屬性	**必要**	**類型**	**說明**
msgId	是	整數	SMIT 段落中提供的 **help_msg_id** 欄位

如果不存在已本地化的說明檔，則會改為顯示預設訊息。會選擇性地併入預設訊息，作為 **<NLSSmitHelp>** 元素的內容。這是提供預設訊息的建議作法。

NLS 指令

AIX 指令發出要顯示的本地化訊息時，會使用「NLS 指令」全球化格式。此案例適用於所有提供 *–h* 旗標，以顯示特定參數說明文字的調整指令（如 **no**、**vmo**）。

<NLSCommand> 元素包含下列屬性：

表 *21.* 屬性			
屬性	**必要**	**類型**	**說明**
command	指令	字串	要執行的 Shell 表示式

範例

1. chssysParam.xml AIX Runtime Expert 型錄的 **<NLSCatalog>** 元素範例（包括預設訊息）。

```
<Description>
  <NLSCatalog catalog="artexcat.cat" setNum="10" msgNum="2">
    Changes a subsystem definition in the subsystem object class.
  </NLSCatalog>
</Description>
```

2. **<NLSSmitHelp>** 元素的範例：

```
<Description>
  <NLSSmitHelp msgId="055136"/>
</Description>
```

3. schedoParam.xml 型錄的 **<NLSCommand>** 元素範例：

```
<Description>
  <NLSCommand command="/usr/sbin/schedo -h maxspin | /usr/bin/tail -n +2"/>
</Description>
```

<SubCat> 元素
可以使用編目檔內的 **<SubCat>** 元素來指定型錄中的子種類、選用參數及子集。

語法

母元素：**<Catalog>**、**<SubCat>**

支援下列屬性：

表 *22.* 屬性

屬性	**必要**	**類型**	**說明**
id	是	字串	指定型錄子種類的名稱。每個編目檔的此名稱應該是唯一的。

支援下列子元素：

表 *23.* 子元素

子元素	**必要**	**說明**
<ShortDescription>	否	子種類的簡短文字說明。
<Description>	否	子種類的詳細文字說明。
<SubCat>	否	巢狀子種類。此元素可能會出現數次。
<ParameterDef>	否	包含參數的內容。此元素可能會出現數次。

屬性

子種類為型錄的本端項目：

· 子種類 ID 在編目檔內是唯一的。
· 多個型錄可充分使用相同的子種類 ID。

定義於型錄中的子種類必須與相關聯範例設定檔中所報告的子種類完全相符。

相關資訊

<Description> 及 <ShortDescription> 元素。

<SubCat> 元素。

<ParameterDef> 元素。

<ParameterDef> 元素
AIX Runtime Expert 使用 **<ParameterDef>** 元素定義於編目檔中。

語法

母元素：**<Catalog>**、**<ParameterDef>**

支援下列屬性：

表 *24.* 屬性

屬性	**必要**	**類型**	**說明**
名稱	是	字串	指定參數名稱。每個型錄中的此名稱必須是唯一的。
type	是	字串	指定如核心引擎中所見的參數類型。
targetClass	否	字串	指定參數的目標類別（如果有的話）。
reboot	否	布林	若為 true，指出需要重新開機。預設值為 false。
cfgmethod	否	字串	指定配置方法的 *id*，該方法定義於包含此參數使用方法的 **<Catalog>** 層次。
constraint	否	字串	指定針對現行編目檔而定義於 <Catalog> 元素層次的限制 ID。
priority	否	整數	相對於此型錄中的其他參數，set 方法中此參數的執行等級。預設值為 0。

支援下列子元素：

表 *25.* 子元素

子元素	**必要**	**說明**
<Description>	否	參數的文字說明。
<ConstraintDef>	否	參數限制定義（disruptive 指令）。
<Get>	否	get 作業的配置方法定義。此元素可能會出現數次。
<Set>	否	set 作業的配置方法定義。此元素可能會出現數次。
<Diff>	否	**diff** 作業的配置方法定義。

表 25. 子元素 (繼續)		
子元素	**必要**	**說明**
<Discover>	否	目標探索的配置方法定義。

屬性

表 26. 屬性	
屬性	**說明**
名稱	name 屬性能唯一識別編目檔內的參數。如需相關資訊，請參閱參數 name 屬性主題。
type	必要的 type 屬性會指出參數的值類型。支援的值如下： · string，適用於英數字串； · integer，適用於數值； · integer-bi，適用於數值，而這些數值具有 “kilo”、“mega”、“giga”、“tera”、“peta” 及 “exa” 的選用大寫或小寫 K、M、G、T、P 或 E 字尾。 會將這些字尾解譯為 1024 的次方； · integer-si，適用於具有選用 SI 字尾的數值。除了字尾是解譯為 1000 的次方之外，其他部分與 integer-bi 類型相同。 · boolean，適用於布林值。支援的值是 0 及 1。 · binary，適用於設定檔中以 base-64 字串編碼的二進位值。

reboot	布林 ‘reboot’ 屬性的預設值為 “false”。如果參數變更需要重新開機才能生效，則此參數必須將其 ‘reboot’ 屬性設為 “true”。
	AIX Runtime Expert 本身永遠不會將系統重新開機。依預設，**artexset** 指令將不會強制執行重新開機參數的設定。如果設定檔包含重新開機參數，則指令會失敗：
	`0590-502: 設定檔中有`
	`參數需要重新開機。`
	`尚未設定設定檔。請使用 -l all`
	`旗標來強制執行所有參數集`
	如果使用適當的 *–l* 旗標來呼叫，則 **artexset** 指令會設定值，並警告使用者需要重新開機才能讓變更生效：
	`0590-206 需要手動後置作業，`
	`變更才會生效`
	`請將系統重新開機`

表 *26.* 屬性 *(*繼續*)*	
屬性	**說明**
priority	依預設，**artexset** 指令會依照沒有定義的順序來設定參數。可以使用 *priority* 屬性來變更此行為，並在其他參數的前後強制設定參數。 預設優先順序為 0。priority 屬性可以用來將此預設優先順序變更為任何介於 -2147483648 及 2147483647 之間的整數值。具有較高優先順序的參數會比較低優先順序的參數先執行。未定義具有相同優先順序的參數設定順序。
targetClass	必須將部分參數與目標相關聯（如設定檔的「Target 元素」相關章節所說明）。必須將那些參數的 *targetClass* 屬性設為其支援目標類別的清單（以逗點區隔）。
cfgmethod	**<ParameterDef>** 元素可藉由使用 *cfgmethod* 屬性參照此配置方法 *id* 屬性，從 **<CfgMethod>** 元素繼承指令行元素。如需配置方法的相關資訊，請參閱 <CfgMethod> 元素的相關章節。
constraint	**<ParameterDef>** 元素可以使用 constraint 屬性來參照 **<ConstraintDef>** 元素的 id 屬性，這指出參數需遵循限制。 如需限制的相關資訊，請參閱「**<ConstraintDef>** 元素」的相關章節。

範例

1. 下列為具有替代整數類型的參數定義範例：來自 vmoParam.xml 編目檔的 *kernel_heap_size*：

   ```
   <ParameterDef name="kernel_heap_psize" type="integer-bi">
   ```

   透過 **artexget** 指令擷取此參數值時，結果與下列內容類似（摘錄自結果設定檔）。

   ```
   <Parameter name="kernel_heap_psize" value="16M"... />
   ```

   參數值會視類型而定，以不同方式解譯：

   · 因為宣告類型為 integer-bi，所以值為 16M=16,777,216。

   · 如果類型為 integer-si，則值為 “16M”=16,000,000。
2. 二進位參數的範例：tsdParam.xml 型錄中的授信簽章資料庫 tsd.dat：

   ```
   <ParameterDef name="tsdatabase" type="binary">
   ```
3. 具有 *reboot* 屬性的參數範例。sysdumpdevParam.xml 型錄中的傾出參數類型：

   ```
   <ParameterDef name="type_of_dump" type="string" reboot="true">
   ```
4. 具有一個目標類別的參數範例：mktcpipParam.xml 型錄中的 *addr* 參數適用於特定的網路介面：

   ```
   <ParameterDef name="addr" type="string" cfgmethod="mktcpip" targetClass="interface">
   ```

5. 具有數個目標類別的參數範例：coreParam.xml 中的命名規格參數適用於特定登錄（檔案、LDAP）中的特定使用者（root、管理、訪客等）。

```
<ParameterDef name="namingspecification" type="string" reboot="true"
targetClass="user,registry"
cfgmethod="coremgt">
```

6. *cfgmethod* 屬性的使用範例：若為 **<Get type="current">** 作業，chlicenseParam.xml 型錄的固定參數會從 *chlicense* 配置方法繼承 **<Command>** 元素， 但它也會在本端針對此相同作業定義其本身的 **<Filter>** 及 **<Mask>**：

```
<CfgMethod id="chlicense">
  <Get type="current">
    <Command>lslicense -c -A</Command>
  </Get>
</CfgMethod>
<ParameterDef name="fixed" cfgmethod="chlicense" type="integer">
  <Get type="current">
    <Filter>tail -n 1 | cut -d: -f3</Filter>
    <Mask value="1">(.*)</Mask>
  </Get>
</ParameterDef>
```

7. constraint 屬性的用法範例：authParam.xml 型錄的授權參數需遵循先前定義於 **<ConstraintDef>** 元素的 **setkst** 限制：

```
<ParameterDef name="authorizations" cfgmethod="cat" constraint="setkst" type="string">
```

name 屬性
參數的名稱通常是由用來取得或設定參數的指令所指定。

參數名稱在編目檔中必須是唯一的。為了要確保設定檔中的 **<Parameter>** 元素與編目檔中唯一的 **<ParameterDef>** 元素可以產生關聯，這是必要動作。

- 如果 **get** 指令顯示數個參數值配對，則 **<Mask>** 元素可用來從單一指令輸出擷取數個參數。只有在參數名稱符合 **get** 指令輸出中所使用的名稱時，才能執行此動作。
- 如果 **set** 指令接受數個參數值配對，則可以在 **<Argument>** 元素中使用 %n 及 %v1 序列，以使用單一指令設定多個參數。只有在參數名稱符合 **set** 指令所使用的名稱時，才能執行此動作。

範例

1. 範例：在 rasoParam.xml 型錄中使用的 **raso -a** 指令會在顯示畫面的每一行顯示一個參數：

```
kern_heap_noexec = 0
kernel_noexec = 1
mbuf_heap_noexec = 0
mtrc_commonbufsize = 485
```

   在此簡易的案例中，參數名稱將會是 *kernel_heap_noexec*、*kernel_noexec* 等等。

2. 範例：acctctlParam.xml 型錄的 **get** 配置方法中所使用的指令會顯示更難以剖析的結果。不僅參數名稱會整合至非格式化的句子，而且會本地化參數名稱及其值。get 配置方法必須執行指令，同時設定環境變數 **LANG=C** 並在每一行將關鍵字取代成適當的參數名稱：

```
Advanced Accounting is not running.
Email notification is off.
The current email address to be used is not set.
Recover CPU accounting time in turbo mode is False.
```

   上述範例中，所選擇的參數名稱為 accounting、email、email_adddr 及 turacct。

相關資訊

- **<Parameter>** 元素
- **<Mask>** 元素

· 展開指令行元素

<ConstraintDef> 元素

語法

母元素：**<Catalog>**、**<ParameterDef>**

支援下列屬性：

表 *27.* 屬性			
屬性	**必要**	**類型**	**說明**
id	否*	字串	指定參數限制的名稱。

*必須針對在型錄層次定義的 **<Constraint>** 元素指定此參數。

支援下列子元素：

表 *28.* 子元素		
子元素	**必要**	**說明**
<Description>	否	disruptive 指令的文字說明。
<PreOp>	否	設定參數值之前要執行的干擾作業。
<PostOp>	否	設定參數值之後要執行的干擾作業。
<BuiltIn>	否	內建干擾作業。此元素可能會出現數次。

用法

部分調整和配置參數可能需要干擾作業，才能讓值變更生效。 干擾作業是可能會暫時岔斷服務或裝置存取的任何作業。一般干擾作業會重新啟動常駐程式、取消裝載或裝載檔案系統，以及讓網路配接卡離線或連線。AIX Runtime Expert 程式使用限制來顯示參數需要干擾作業，以讓變更生效。會使用 **<ConstraintDef>** 元素來定義這類限制。

限制可定義如下：

· 如果限制只能套用至單一參數，則是在 **<ParameterDef>** 元素內。

· 在型錄層次中，**<ConstraintDef>** 元素必須具有 *id* 屬性，以容許日後可在 **<ParameterDef>** 元素中參照限制。

內建的限制

<BuiltIn> 元素不包含任何屬性或子元素。

內建限制會定義核心引擎中寫在程式中的作業。 目前只有一個已定義的內建限制：*bosboot*。 內建限制與其他干擾作業不同之處在於 AIX Runtime Expert 永不執行 *bosboot* 指令。核心引擎將只會警告需要 *bosboot* 以讓變更生效。

```
0590-206 需要手動後置作業，變更才會生效
請執行 bosboot
```

PreOp 及 PostOp 限制

<PreOp> 元素會定義在 set 配置方法設定參數值之前，所要執行的必要指令（Shell 表示式）。**<PostOp>** 元素會定義在 set 配置方法執行之後，所要執行的必要指令。

<ConstraintDef> 元素必須包含 0 個或一個 **<PreOp>** 子元素，以及 0 個或一個 **<PostOp>** 子元素。

範例

1. 內建限制的範例（在型錄層次上）

```
<ConstraintDef id="bosboot">
  <Description>
<NLSCatalog catalog="artexcat.cat" setNum="51" msgNum="3">
    bosboot
</NLSCatalog>
  </Description>
  <Built>Inbosboot</BuiltIn>
</ConstraintDef>
```

2. **<PreOp>** 限制的範例：`trustchkParam.xml` 型錄中的 *clic* 限制。請注意在此範例中，**preop** 指令不會執行任何內容，而只會檢查 **set** 指令所需的核心延伸是否存在。 如果未安裝核心延伸，則在 **<PreOp>** 元素中定義的限制將會失敗，而且不會執行 **set** 指令：

```
<ConstraintDef id="clic">
  <Description>
  <NLSCatalog catalog="artexcat.cat" setNum="48" msgNum="3">
     Check that the clic.rte kernel extension is installed.
  </NLSCatalog>
  </Description>
  <Pre>Oplslpp -l "clic*"</PreOp>
 </ConstraintDef>
```

3. **<PostOp>** 限制的範例：`authParam.xml` 型錄中的設定「核心安全表格」限制。 完成所有修改後，只需要在核心中載入一次已修改的資料庫。

```
<ConstraintDef id="setkst">
  <Description>
<NLSCatalog catalog="artexcat.cat" setNum="5" msgNum="3">
Send the authorizations database to the KST (Kernel Security Tables)
    </NLSCatalog></Description>
  <PostOp>/usr/sbin/setkst -t auth &gt;/dev/null</PostOp>
  </ConstraintDef>
```

<CfgMethod> 元素

語法

母元素：**<Catalog>**

支援下列屬性：

表 *29.* 屬性

屬性	**必要**	**類型**	**說明**
id	是	字串	指定配置方法的名稱。

支援下列子元素：

表 *30.* 子元素

子元素	**必要**	**號碼**	**說明**
<Get>	否	0 – 1	get 作業的配置方法定義。此元素可能會出現數次。
<Set>	否	0 – 1	set 作業的配置方法定義。此元素可能會出現數次。

表 30. 子元素 (繼續)			
子元素	**必要**	**號碼**	**說明**
<Diff>	否	0 – 1	**diff** 作業的配置方法定義。
<Discover>	否	0 – 1	目標探索的配置方法定義。
<Property>	否	0 – 任何	使用配置方法將內容指派給參數。

用法

<CfgMethod> 元素會使用 **<ParameterDef>** 元素的 *cfgmethod* 屬性，定義日後可由參數參照的配置方法。然後參數會繼承已參照配置方法下定義的所有元素。

視參數而定，使用配置可能在本端定義方面有幾項優點：

· 它可簡化編目檔，並避免數個參數產生重複的相同指令行元素。

· 它可容許由單一指令處理多個參數。

範例

vmoParam.xml 型錄可定義所有使用相同配置方法的許多參數。以下是此型錄的簡化版本：

```
<Catalog id="vmoParam" version="2.1">
 <CfgMethod id="vmo">
    <Get type="current">
      <Command>/usr/sbin/vmo -a</Command>
      <Mask name="1" value="2">[[:space:]]*(.*) = (.*)</Mask>
    </Get>

    <Get type="nextboot">
      <Command>/usr/sbin/vmo -r -a</Command>
      <Mask name="1" value="2">[[:space:]]*(.*) = (.*)</Mask>
    </Get>

    <Set type="permanent">
      <Command>/usr/sbin/vmo -p%a</Command>
   <Argument>%n=%v1</Argument>
    </Set>

    <Set type="nextboot">
      <Command>/usr/sbin/vmo -r%a</Command>
      <Argument>%n=%v1</Argument>
    </Set>
  </CfgMethod>

  <ParameterDef name="ame_maxfree_mem" cfgmethod="vmo" type="integer" />
  <ParameterDef name="ame_min_ucpool_size" cfgmethod="vmo" type="integer" />
  <ParameterDef name="ame_minfree_mem" cfgmethod="vmo" type="integer" />
  ...

</Catalog>
```

相關資訊

· 產生指令行

· **<Get>** 元素

· **<Set>** 元素

<Get> 元素

語法

母元素：**<CfgMethod>**、**<ParameterDef>**

支援下列屬性：

表 *31.* 屬性			
屬性	**必要**	**類型**	**說明**
type	是	字串	指定 **get** 指令的類型（current 或 *nextboot*）。

支援下列子元素：

表 *32.* 子元素			
子元素	**必要**	**號碼**	**說明**
<Command>	否	0 – 1	指令
<Argument>	否	0 – 1	指令行引數
<Stdin>	否	0 – 1	**<Stdin>** 元素所支援的引數
<Filter>	否	0 – 1	過濾器
<Mask>	否	0 – 1	輸出擷取遮罩
<Prereq>	否	0 – 任何	將必要條件指派給 get 作業

必須於 **<CfgMethod>** 層次，或直接於 **<ParameterDef>** 層次，針對每一個參數定義 **<Command>** 元素。

用法

<Get> 元素會說明如何擷取特定參數的值。它可直接使用於 **<ParameterDef>** 元素下，或在 **<ParameterDef>** 元素中參照的 **<CfgMethod>** 元素下使用 *cfgmethod* 屬性，或者使用這兩種可能性的組合。

應針對每一個參數定義兩個 Get 元素，每一個都需定義 *type* 屬性的支援值：

- Get **type=“current”** 會識別將要執行的方法，以擷取參數的執行時期值。
- Get **type=“nextboot”** 會識別將要執行的方法，以擷取參數在下次系統重新開機之後將具有的值。
- 要執行的 get 方法是視所執行的作業而定：
  - 如果 **artexget** 指令使用 *–r* 旗標來呼叫，則會使用 current get 方法。
  - 如果 **artexget** 指令使用 *–n* 旗標來呼叫，則會使用 *nextboot* get 方法。
  - 如果 **artexget** 指令使用 *–p* 旗標來呼叫，則執行的方法會視 *applyType* 屬性的參數輸入而定。 current get 方法是用於將其 *applyType* 屬性設為 runtime 的參數，而 *nextboot* get 方法是用於 *applyType* 屬性為 reboot 的參數。

相關資訊

產生指令行

<Mask> 元素。

<Set> 元素
<Set> 元素會定義如何建置指令行以設定參數值。

語法

母元素：**<CfgMethod>**、**<ParameterDef>**

支援下列屬性：

表 *33.* 屬性			
屬性	**必要**	**類型**	**說明**
type	是	字串	將 **set** 指令的類型指定為 current 或 nextboot。

支援下列子元素：

表 *34.* 子元素			
子元素	**必要**	**號碼**	**說明**
<Command>	否	0 – 1	指令
<Argument>	否	0 – 1	指令行引數
<Stdin>	否	0 – 1	Stdin 引數
<Prereq>	否	0 – 任何	將必要條件指派給 **<Set>** 作業

註 : 必須於 **<CfgMethod>** 層次，或直接於 **<ParameterDef>** 層次，針對每一個參數定義 **<Command>** 元素。

用法

可以針對每一個參數定義三種類型的 **<Set>** 元素（由其必要的 *type* 屬性識別）：

- Set **type=“current”** 可定義 set 作業，其只變更現行階段作業的參數值。會在系統重新開機之後遺失使用 set 作業進行的任何變更。
- Set **type=“nextboot”** 可定義 set 作業，其只變更參數在下次系統重新開機之後採用的值。不會修改現行值。
- Set **type=“permanent”** 可定義 set 作業，其變更參數的 current 值和 nextboot 值。

set 作業執行的類型是根據 **artexset** 指令執行時所併入的參數，以及設定檔中參數 *applyType* 屬性來決定。下表視參數的 *applyType* 屬性上編目檔中所定義的 set 方法而定，來彙總所執行的 set 方法：

表 *35. Set* 方法 - 定義的 *Set* 方法類型和參數 *applyType* 屬性。				
current	**nextboot**	**permanent**	**runtime**	**nextboot**
0	0	0	未設定（發生錯誤）	未設定（發生錯誤）
0	0	1	設為 permanent	未設定（發生錯誤）
0	1	0	設為 nextboot 並顯示警告	設為 nextboot
0	1	1	設為 permanent	未設定（發生錯誤）
1	0	0	設為 current 並顯示警告	設為 nextboot
1	0	1	設為 permanent	未設定（發生錯誤）
1	1	0	設為 current 並設為 nextboot	設為 nextboot
1	1	1	設為 permanent	設為 nextboot

相關資訊

產生指令行。

<Diff> 元素
<Diff> 元素會定義如何建置指令行以比較參數的兩個值。

語法

母元素：**<CfgMethod>**、**<ParameterDef>**

支援下列子元素：

表 *36.* 子元素		
子元素	**必要**	**說明**
<Command>	否	指令
<Argument>	否	指令行引數
<Stdin>	否	Stdin 引數
<Filter>	否	過濾器
<Mask>	否	輸出擷取遮罩

註 : 必須於 **<CfgMethod>** 層次，或直接於 **<ParameterDef>** 層次，針對每一個參數定義 **<Command>** 元素。

用法

通常不需要 **<Diff>** 元素，因為從架構知道如何在內部根據類型（string、integer、integer-bi、binary 等等）比較兩個參數值。然而，若為內部比較不適用於特定參數的狀況，可能會改用外部指令。

範例

下列 **<Diff>** 元素可用於大部分的參數，即使使用內部比較功能也較為有效率。**<Diff>** 元素使用 **diff** 指令來比較包含兩個值的兩個檔案：

```
<Diff>
  <Command>/usr/bin/diff %f1 %f2; echo $?</Command>
</Diff>
```

相關資訊

產生指令行。

<Mask> 元素。

<Discover> 元素
<Discover> 元素會定義如何針對支援目標的參數，建置指令行以探索目標。

語法

母元素：**<CfgMethod>**、**<ParameterDef>**

支援下列子元素：

表 *37.* 子元素			
子元素	**必要**	**號碼**	**說明**
<Command>	否	0 – 1	指令
<Prereq>	否	0 – 任何	將必要條件指派給 discover 作業

註 : 必須於 **<CfgMethod>** 層次，或直接於 **<ParameterDef>** 層次，針對每一個參數定義 **<Command>** 元素。

用法

會使用 discover 指令來取得給定參數的目標實例清單。

對於支援 N 個目標類別的參數，其 discover 指令輸出格式如下：

```
class_1=inst_1_1;class_2=inst_2_1;...;class_N=inst_N_1
class_1=inst_1_2;class_2=inst_2_2;...;
class_N=inst_N_2class_1=inst_1_3;
class_2=inst_2_3;...;class_N=inst_N_3
...
```

artexget 指令會針對滿足下列其中一項準則的參數，產生並執行 discover 指令：

· 包含具有空的 *class* 及 *instance* 屬性的 **<Target>** 元素。**<Target class="" instance="" />**

· 包含至少一個具有 *match* 屬性的 **<Target>** 元素：**<Target class="..." match="..." />**

此外，**artexset** 指令還需要滿足下列兩個準則：

· 使用 *–d* 旗標來呼叫 **artexset** 指令。

· 設定檔中的 **<Parameter>** 元素已將 *setDiscover* 屬性設為 true。

範例

1. mktcpipParam.xml 型錄會使用下列 discover 指令，來取得系統上所定義的網路介面清單：

   ```
   <Discover>
     <Command>
      /usr/sbin/lsdev -C -c if -F "name" | /usr/bin/sed -e 's/^/interface=/'
    </Command>
   </Discover>
   ```

   此指令會產生下列輸出：

   ```
   interface=en0
   interface=et0
   interface=lo0
   ```

2. chuserParam.xml 型錄會使用下列 **discover** 指令，來取得所有可載入授權模組的所有使用者清單：

   ```
   <Discover>
     <Command>
     /usr/sbin/lsuser -a registry ALL | /usr/bin/sed -e "s/\(.*\) registry=\(.*\)/module=
   \2;user=\1/g"
   </Command>
   </Discover>
   ```

   此指令會產生下列輸出：

   ```
   module=LDAP;user=daemon
   module=LDAP;user=bin
   module=LDAP;user=sys
   module=LDAP;user=adm
   ...
   module=files;user=root
   module=files;user=daemon
   module=files;user=bin
   module=files;user=sys
   module=files;user=adm
   ...
   ```

<Command> 元素
<Command> 元素會定義基本指令，用來執行由母元素所定義的作業。

語法

母元素：**<Get>**、**<Set>**、**<Diff>**、**<Discover>**、**<PrereqDef>**、**<Prereq>**、**<PropertyDef>**、**<Property>**、**<Command>**

用法

已展開 **<Command>** 元素的內容（如展開指令行元素一節中所述），並與其他指令行元素結合以組成完整指令行。 如需詳細資料，請參閱產生指令行一節。

XML 文件中不容許部分在 Shell 表示式中經常發現的字元，例如 <、 > 及 &。這些字元必須以相對應的 XML 實體來取代：

表 *38. XML* 實體	
字元	**XML 實體**
<	&lt;
>	&gt;
&	&amp;

此外，如果表示式包含許多這類字元，則可以使用 CDATA 區段。CDATA 區段的開頭為 **<![CDATA[**，並以 **]]>** 結尾。

必須於 **<CfgMethod>** 層次或 **<ParameterDef>** 層次，針對每一個參數支援的每一個作業定義 **<Command>** 元素。

範例

envParam.xml 型錄會定義代表 /etc/profile 檔案內容且稱為 profile 的參數。 對於此參數，**<Get>** 元素會使用 **cat** 指令來擷取 /etc/profile 檔案的內容：

```
<ParameterDef name="profile">
  <Get type="current">
    <Command>/usr/bin/cat /etc/environment</Command>
  </Get>
</ParameterDef>
```

<Argument> 元素

語法

母元素：**<Get>**、**<Set>**、**<Diff>**、**<PrereqDef>**、**<Prereq>**、**<PropertyDef>**、**<Property>**

用法

已展開 **<Argument>** 元素的內容（如展開指令行元素一節中所述），並與 **<Command>** 及（或）<Stdin> 元素結合以組成完整的指令行。如需詳細資料，請參閱產生指令行一節。

XML 文件中不容許部分在 Shell 表示式中經常發現的字元，例如 <、 > 及 &。這些字元必須以相對應的 XML 實體來取代：

表 *39. XML* 實體	
字元	**XML 實體**
<	&lt;
>	&gt;
&	&amp;

此外，如果表示式包含許多這類字元，則可以使用 CDATA 區段。CDATA 區段的開頭為 **<![CDATA[**，並以 **]]>** 結尾。

範例

vmoParam.xml 型錄會使用 **<Argument>** 元素，針對設定檔中的每一個 **vmo** 參數，將引數新增至 **vmo** 指令：

```
<CfgMethod id="vmo">
  <Set type="permanent">
    <Command>/usr/sbin/vmo -p%a</Command>
    <Argument> -o %n=%v1</Argument>
  </Set>
</CfgMethod>
```

<Stdin> 元素

語法

母元素：**<Get>**、**<Set>**、**<Diff>**、**<PrereqDef>**、**<Prereq>**、**<PropertyDef>**、**<Property>**

用法

已展開 **<Stdin>** 元素的內容（如展開指令行元素一節所述），而且結果資料已寫入針對母元素中定義的作業所產生的指令行標準輸入中。

範例

envParam.xml 型錄會定義代表 /etc/profile 檔案內容且稱為 profile 的參數。 針對此參數，set 作業會將參數值寫入至 **cat** 指令的標準輸入，以改寫 /etc/profile 檔案：

```
<ParameterDef name="profile">
  <Set type="permanent">
    <Command>/usr/bin/cat &gt; /etc/profile</Command>
     <Stdin>%v1</Stdin>
   </Set>
 </Get>
```

相關資訊

產生指令行

<Filter> 元素

語法

母元素：**<Get>**、**<Diff>**、**<PropertyDef>**、**<Property>**

用法

<Filter> 元素的內容為指令，其將針對在母元素中定義的作業所產生的指令行輸出作為輸入來傳遞。

XML 文件中不容許部分在 Shell 表示式中經常發現的字元，例如 <、 > 及 &。這些字元需要由相對應的 XML 實體來取代：

表 *40. XML* 實體

字元	**XML 實體**
<	&lt;
>	&gt;
&	&amp;

此外，如果表示式包含許多這類字元，則可以使用 CDATA 區段。CDATA 區段的開頭為 **<![CDATA[**，並以 **]]>** 結尾。

範例

nfsParam.xml 型錄針對參數 *v4_root_node* 的 get 作業使用 **<Filter>** 元素，以從 **nfds –getnode** 指令的輸出擷取根節點：

```
<ParameterDef id="v4_root_node">
  <Get type="current">
    <Command>
       /usr/sbin/nfsd -getnodes
    </Command>
    <Filter>
       /usr/bin/awk -F: 'NR == 2 { printf("%s", $1) }'
    </Filter>
  </Get>
</ParameterDef>
```

相關資訊

產生指令行

<Mask> 元素

語法

母元素：**<Get>**、**<Diff>**、**<Discover>**（僅在一個 <**SeedDef**> 之下）、**<PropertyDef>**、**<Property>**

支援在 **<Get>** 或 **<Diff>** 元素中使用下列屬性：

表 *41.* 屬性

屬性	**必要**	**類型**	**說明**
名稱	否	整數	指定符合參數名稱的子表示式索引。有效值為 1 及 2。
value	否	整數	指定符合參數值的子表示式索引。有效值為 1 及 2。

支援在 **<SeedDef>** 元素的 **<Discover>** 子元素下使用下列屬性：

表 *42.* 屬性

屬性	**必要**	**類型**	**說明**
catalog	是	整數	指定符合型錄名稱的子表示式索引。有效值為 1、2 及 3。
名稱	是	整數	指定符合參數名稱的子表示式索引。有效值為 1、2 及 3。
target	否	整數	指定符合參數目標的子表示式索引。有效值為 1、2 及 3。

支援在 **<PropertyDef>** 或 **<Property>** 元素下使用下列屬性：

表 *43.* 屬性			
屬性	**必要**	**類型**	**說明**
value	否	整數	指定符合參數名稱的子表示式索引。如果有指定的話，必須設為 "1"。

用法

<Mask> 元素會定義一個正規表示式，該表示式將套用到指令輸出的每一行，以從那些行中擷取資料。擷取的資料取決於使用 **<Mask>** 元素的位置。

如果未指定任何屬性，則會使用符合正規表示式的指令輸出中的最後一行來擷取資料。所擷取的資料是符合正規表示式的行的一部分。在 **<Get>** 或 **<Diff>** 元素下使用時，會使用所擷取的資料作為參數值。在 **<PropertyDef>** 或 **<Property>** 元素下使用時，會使用所擷取的資料作為內容值。

如果只指定 *value* 屬性，則必須將它設為 1，而且正規表示式只能包含一個子表示式。會使用符合正規表示式的指令輸出中的最後一行來擷取資料。所擷取的資料是符合第一個（而且是唯一的）子表示式的行的一部分。在 **<Get>** 或 **<Diff>** 元素下使用時，會使用所擷取的資料作為參數值。在 **<PropertyDef>** 或 **<Property>** 元素下使用時，會使用所擷取的資料作為內容值。

如果指定 *name* 和 *value* 屬性，則那些屬性的其中之一必須設為 1，而其他屬性必須設為 2，因此正規表示式必須包含兩個子表示式。*name* 和 *value* 是擷取自符合正規表示式的指令輸出的每一行。在 **<Get>** 元素中使用時，會使用 name 作為參數名稱，並使用 value 作為參數值。在 **<Diff>** 元素中使用時，會使用 name 作為參數名稱，並使用 value 作為比較結果。使用此函數，可使用單一 **get** 指令來擷取數個參數值，而且可使用單一 **diff** 指令來比較數個參數。

在 **<SeedDef>** 元素的 **<Discover>** 子元素中使用時，必須指定 catalog 和 name 屬性。型錄名稱和參數名稱是擷取自符合正規表示式的指令輸出的每一行。如果在系統中找到符合所擷取的型錄名稱的型錄，而且它包含符合所擷取的參數名稱之參數的定義，則會將參數插入到設定檔中。您可以新增選用的目標引數，來擷取每一個探索到的參數的目標定義。目標定義必須遵循以分號區隔的 "class=instance" 配對清單格式（例如，`class1=instance1;class2=instance2;...` 格式）。

範例

1. `vmoParam.xml` 型錄搭配使用 **<Mask>** 元素與 *name* 及 *value* 屬性，以從單一 **vmo –a** 指令擷取所有參數值：

```
<CfgMethod id="vmo">
  <Get type="current">
    <Command>/usr/sbin/vmo -a</Command>
    <Mask name="1" value="2">[[:space:]]*(.*) = (.*)</Mask>
  </Get>
</CfgMethod>
```

2. 如果撰寫 `vmoParam.xml` 型錄的方式是使用某個別指令來擷取每一個參數的值，則可能已使用 **<Mask>** 元素搭配 *value* 屬性集，而且沒有 *name* 屬性：

```
<CfgMethod id="vmo">
  <Get type="current">
    <Command>/usr/sbin/vmo -o %n</Command>
    <Mask value="1"> = (.*)</Mask>
  </Get>
</CfgMethod>
```

3. 或者，使用只符合該值的正規表示式：

```
<CfgMethod id="vmo">
  <Get type="current">
    <Command>/usr/sbin/vmo –o %n</Command>
    <Mask>[^ ]*$</Mask>
  </Get>
</CfgMethod>
```

上述三個範例中，第一個是最有效率的，因為它只需要一個指令就能擷取所有 **vmo** 指令參數。範例 2 和 3 會針對每一個 **vmo** 指令參數來產生個別指令，因為已在 **<Command>** 元素中使用參數名稱。

4. 下列 **<SeedDef>** 元素會定義一個種子，而該種子可用來探索所有裝置的所有屬性。它會使用 target 來指定在其上操作的裝置：

```
<SeedDef name="devAttr">
  <Discover>
    <Command>
      /usr/sbin/lsdev -F 'name class subclass type' |
      while read DEV CLASS SUBCLASS TYPE
                  do
        /usr/sbin/lsattr -F attribute -l $DEV |
        while read PAR
                    do
          echo device=$DEV devParam.$CLASS.$SUBCLASS.$TYPE $PAR
        done
              done
            </Command>
    <Mask target="1" catalog="2" name="3">(.*) (.*) (.*) <Mark>
  </Discover>
</SeedDef>
```

Discovery 指令會使用下列格式，在不同行上顯示每一個探索到的裝置屬性：

```
device=DeviceName devParam.Class.Subclass.Type AttributeName
```

例如：

```
device=en0 devParam.if.EN.en tcp_recvspace
device=en0 devParam.if.EN.en tcp_sendspace
device=ent0 devParam.adapter.vdevice.IBM,l-lan alt_addr
device=ent0 devParam.adapter.vdevice.IBM,l-lan chksum_offload
```

相關資訊

產生指令行

<SeedDef> 元素
<SeedDef> 元素會定義一個種子，您可以使用 **<Seed>** 元素在設定檔中使用此種子。

語法

母元素：**<Catalog>**

支援下列屬性：

表 *44.* 屬性

屬性	**必要**	**類型**	**說明**
名稱	是	字串	指定種子的名稱。每個型錄中的這個名稱都必須是唯一的。

支援下列子元素：

表 *45.* 子元素

子元素	**必要**	**說明**
<Discover>	是	指定用來探索參數的指令。

用法

種子用來在 Get 作業期間動態地探索參數。

發出 **artexget** 指令時，會根據編目檔的相符 **<SeedDef>** 元素中所定義的規則，將輸入設定檔中的每一個 **<Seed>** 元素展開至一個以上的 **<Parameter>** 元素中。此處理程序稱為參數探索。接著，**artexget** 指令會使用展開的設定檔如常處理。

<SeedDef> 元素只包含一個 **<Discover>** 子元素，此子元素可定義要執行的指令，以及用來擷取參數名稱、型錄名稱（以冒號區隔的清單，不含 **.xml** 副檔名）和指令輸出中的目標（選用的，使用 *class1=instance1;class2=instance2;...* 格式）的遮罩。 對於輸出的每一行，會載入系統上所找到並以冒號區隔的清單中的第一個型錄。如果在此型錄中找到參數定義，則會在輸出設定檔中建立參數，而該輸出設定檔具有從行中所擷取的目標。如果指令輸出中的行不符合遮罩，或者找不到其編目檔，或沒有參數定義（已在編目檔中找到參數定義時），則會忽略這幾行。

範例

1. 下列型錄會定義一個稱為 *vmoTunables* 的 **<SeedDef>** 元素，該元素會探索 AIX Runtime Expert 所支援的所有非限制 *vmo tunables* 種子：

```
<?xml version="1.0" encoding="UTF-8"?>
<Catalog id="vmoSeed">
  <SeedDef name="vmoTunables">
    <Discover>
      <Command>/usr/sbin/vmo -x | /usr/bin/awk -F, '{ print "vmoParam:" $1 }'</Command>
      <Mask catalog="1" name="2">(.*):(.*)/Mask>
    </Discover>
  </SeedDef>
</Catalog>
```

   Discovery 指令會在不同行上顯示每一個可調整值，前面加上定義這些可調整值的型錄名稱：

```
...
vmoParam:enhanced_affinity_vmpool_limit
vmoParam:esid_allocator
vmoParam:force_relalias_lite
vmoParam:kernel_heap_psize
...
```

   下列設定檔會使用 *vmo tunables* 種子來擷取 AIX Runtime Expert 支援的所有非限制 *vmo tunables* 種子：

```
<?xml version="1.0" encoding="UTF-8"?>
<Profile>
  <Catalog id="vmoSeed">
    <Seed name="vmoTunables"/>
  </Catalog>
</Profile>
```

   對設定檔執行 **artexget -r** 指令時，該指令會產生類似於下列範例的設定檔：

```
<?xml version="1.0" encoding="UTF-8"?>
<Profile>
 <Catalog id="vmoParam">
  ...
  <Parameter name="enhanced_affinity_vmpool_limit" value="10"/>
  <Parameter name="esid_allocator" value="0"/>
  <Parameter name="force_relalias_lite" value="0"/>
  <Parameter name="kernel_heap_psize" value="65536" applyType="nextboot" reboot="true"/>
  ...
</Catalog>
</Profile>
```

2. 下列 **<SeedDef>** 元素會定義一個種子，而該種子是用來探索所有裝置的所有屬性。此元素會使用目標種子來指定在其上操作的裝置：

```
<SeedDef name="devAttr">
  <Discover>
    <Command>
      /usr/sbin/lsdev -F 'name class subclass type' |
      while read DEV CLASS SUBCLASS TYPE
                  do
        /usr/sbin/lsattr -F attribute -l $DEV |
        while read PAR
                    do
```

```
          echo device=$DEV devParam.$CLASS.$SUBCLASS.$TYPE:devParam.$CLASS
.$SUBCLASS:devParam.$CLASS $PAR
        done
              done
            </Command>
    <Mask target="1" catalog="2" name="3">(.*) (.*) (.*)</Mask>
  </Discover>
</SeedDef>
```

Discovery 指令會使用下列格式，在不同行上顯示每一個探索到的裝置屬性：

```
device=DeviceName devParam.Class.Subclass.Type:devParam.Class.Subclass:devParam.Class
AttributeName
```

例如：

```
device=en0 devParam.if.EN.en:devParam.if.EN:devParam.if tcp_recvspace
device=en0 devParam.if.EN.en:devParam.if.EN:devParam.if tcp_sendspace
device=ent0 devParam.adapter.vdevice.IBM,l-lan:devParam.adapter.vdevice:devParam.adapter
alt_addr
device=ent0 devParam.adapter.vdevice.IBM,l-lan:devParam.adapter.vdevice:devParam.adapter
chksum_offload
```

<Prereq> 元素
<Prereq> 元素可將必要條件指派給 **<Get>**、**<Set>** 及 **<Discover>** 作業。

語法

母元素：**<Get>**、**<Set>** 及 **<Discover>**

支援下列屬性：

表 *46.* 屬性

屬性	**必要**	**類型**	**說明**
id	否	字串	指定唯一的 ID

支援下列子元素：

表 *47.* 子元素

子元素	**必要**	**說明**
<Command>	否	指令
<Argument>	否	指令行引數
<Stdin>	否	<Stdin> 元素所支援的引數
<ErrMessage>	否	必要條件失敗時所要列印的訊息

註 : 必須針對每一個必要條件定義 **<Command>** 元素：在 **<ParameterDef>** 層次、在 **<CfgMethod>** 層次，或在 **<PrereqDef>** 元素中。

用法

Prereqs 指令可針對使用 **<Get>**、**<Set>** 及 **<Discover>** 作業的參數，決定處理此 **<Get>**、**<Set>** 及 **<Discover>** 作業的條件。將忽略其 **prereq** 指令失敗（非零回覆碼）的參數，而且會列印在必要條件中所定義的錯誤訊息。

<Prereq> 元素可將必要條件指派給母項 **<Get>**、**<Set>** 或 **<Discover>** 作業。必要條件是在本端的 **<Prereq>** 元素下定義，或是繼承自更高層次的 **<Prereq>** 或 **<PrereqDef>** 元素，而此元素需要有相符的 *id* 屬性。

參數會在本端的 **<ParameterDef>** 元素下定義所有必要條件。 此外，如果使用配置方法，則必要條件還會在參數的配置方法中定義內容。結果是：如果必要條件是定義在 **<CfgMethod>** 元素中，則使用配置方法的所有 **<ParameterDef>** 元素都將自動包含該必要條件（即使部分元素可能在本端重新定義必要條件）。

將以下列順序搜尋針對給定的作業定義必要條件的 **<Command>**、**<Argument>**、**<Stdin>** 及 **<ErrMessage>** 元素：

- 在 **<ParameterDef>** 元素的相關作業的**<Prereq>** 子元素下。
- 如果 **<ParameterDef>** 元素具有 *cfgmethod* 屬性，則是在配置方法的相關作業中具有相符 *id* 的 **<Prereq>** 子元素下。
- 在具有相符 ID 的型錄的 **<PrereqDef>** 元素下。

範例

下列範例會定義一個必要條件，此必要條件會檢查是否會將 **netaddr** 和 **netaddr6** 參數套用在擷取它們的相同系統上：

```
<ParameterDef name="netaddr" type="string" targetClass="device" cfgmethod="attr">
  <Set type="permanent">
    <Prereq>
      <Command>[[ `/usr/bin/uname -f` = %p[nodeId] ]]</Command>
      <ErrMessage>Parameter cannot be applied to a different node</ErrMessage>
    </Prereq>
  </Set>
</ParameterDef>

<ParameterDef name="netaddr6" type="string" targetClass="device" cfgmethod="attr">
  <Set type="permanent">
    <Prereq>
      <Command>[[ `/usr/bin/uname -f` = %p[nodeId] ]]</Command>
      <ErrMessage>Parameter cannot be applied to a different node</ErrMessage>
    </Prereq>
  </Set>
</ParameterDef>
```

在此範例中，測試會執行兩次：一次是針對 **netaddr** 參數，另一次是針對 **netaddr6** 參數。這項雙重處理的原因是每一個參數都具有其專屬的必要條件，以及其專屬的 **<Command>** 元素。若要參考只需要執行測試一次的範例，請參閱 artex_catalog_elem_PrereqDef.dita。

<PrereqDef> 元素
<PrereqDef> 元素可在之後於 **<Prereq>** 元素中使用。

語法

母元素：**<Catalog>**

支援下列屬性：

表 *48.* 屬性

屬性	**必要**	**類型**	**說明**
名稱	是	字串	指定內容的名稱。

支援下列子元素：

表 *49.* 子元素

子元素	**必要**	**說明**
<Command>	否	指令
<Argument>	否	指令行引數
<Stdin>	否	<Stdin> 元素所支援的引數
<ErrMessage>	否	必要條件失敗時所要列印的訊息

註 : 必須針對每一個必要條件定義 **<Command>** 元素：在 **<ParameterDef>** 層次、在 **<CfgMethod>** 層次，或在 **<PrereqDef>** 元素中。

用法

Prereq 指令可針對使用 **<Get>**、**<Set>** 或 **<Discover>** 作業的參數，決定執行 **<Get>**、**<Set>** 及 **<Discover>** 作業的條件。將忽略其 **prereq** 指令失敗（非零回覆碼）的參數，而且會列印在必要條件中所定義的錯誤訊息。

<PrereqDef> 元素會定義一個必要條件。此必要條件之後可以使用具有相同 *id* 屬性的 **<Prereq>** 元素，與參數或配置方法的作業相關聯。

範例

下列範例會定義 *nodeId* 必要條件，並將它指派給 **netaddr** 及 **netaddr6** 參數：

```
<PrereqDef id="nodeId">
  <Command>[[ `/usr/bin/uname -f` = %p[nodeId] ]]</Command>
  <ErrMessage>Parameter cannot be applied to a different node</ErrMessage>
</PrereqDef>

<ParameterDef name="netaddr" type="string" targetClass="device" cfgmethod="attr">
  <Set type="permanent">
    <Prereq id="nodeId"/>
  </Set>
  <Property name="nodeId"/>
</ParameterDef>

<ParameterDef name="netaddr6" type="string" targetClass="device" cfgmethod="attr">
  <Set type="permanent">
    <Prereq id="nodeId"/>
  </Set>
  <Property name="nodeId"/>
</ParameterDef>
```

在此範例中，測試只會執行一次，因為這兩個參數會對其必要條件使用相同的 **<Command>** 元素，且對這兩個參數所產生的指令行是一樣的。

<Property> 元素
<Property > 元素會將內容指派給參數或配置方法。

語法

母元素：**<CfgMethod>**、**<ParameterDef>**

支援下列屬性：

表 *50.* 屬性

屬性	必要	類型	說明
名稱	是	字串	指定內容的名稱。

支援下列子元素：

表 *51.* 子元素

子元素	必要	說明
<Command>	否	指令
<Argument>	否	指令行引數
<Stdin>	否	<Stdin> 元素所支援的引數
<Filter>	否	過濾器
<Mask>	否	輸出擷取遮罩

註 : 必須針對每一個內容定義 **<Command>** 元素：在 **<ParameterDef>** 層次、在 **<CfgMethod>** 層次，或在 **<PropertyDef>** 元素中。

用法

內容是與參數相關聯的鍵值組。鍵值組的值是透過 **artexget -r** 和 **artexget -n** 指令來擷取，並儲存在輸出設定檔中。可以使用 %p[property_name] 序列，將儲存在設定檔中的內容值插入到指令行中。

<Property> 元素會將內容指派給參數或配置方法。內容是在本端的 **<Property>** 元素下定義，或是繼承自更高層次的 **<Property>** 或 **<PropertyDef>** 元素，而這些元素需要有相符的 name 屬性。

參數會在本端的 **<ParameterDef>** 元素下定義所有內容。此外，如果使用配置方法，則參數還會在參數的配置方法下定義所有內容。結果是：如果內容是定義在 **<CfgMethod>** 元素下，則使用配置方法的所有 **<ParameterDef>** 元素都將自動包含該內容（即使部分元素可能在本端重新定義內容）。

內容值是擷取自指令行輸出。指令行是透過結合 **<Command>**、**<Argument>**、**<Stdin>** 及 **<Filter>** 元素來建置（如產生指令行小節中所述）。您必須使用下列其中一個內容值：指令行的原始輸出，或符合遮罩（如果已指定 **<Mask>** 元素的話）的部分輸出。

將以下列順序搜尋定義內容的 **<Command>**、**<Argument>**、**<Stdin>**、**<Filter>** 及 **<Mask>** 元素：

- 在 **<ParameterDef>** 層次的 **<Property>** 元素下。
- 如果 **<ParameterDef>** 元素具有 *cfgmethod* 屬性，則是在具有相符的 *name* 屬性的 **<Property>** 元素的配置方法下。
- 在具有相符 name 屬性的型錄的 **<PropertyDef>** 元素下。

範例

下列範例會將 *nodeId* 內容指派給 **netaddr** 和 **netaddr6** 參數：

```
<ParameterDef name="netaddr" type="string" targetClass="device" cfgmethod="attr">
  <Property name="nodeId">
    <Command>/usr/bin/uname -f/<Command>
    <Mask>.*/<Mask>
  </Property>
</ParameterDef>

<ParameterDef name="netaddr6" type="string" targetClass="device" cfgmethod="attr">
  <Property name="nodeId">
    <Command>/usr/bin/uname -f</Command>
    <Mask>.*</Mask>
  </Property>
</ParameterDef>
```

在此範例中，遮罩符合整行，而且只用來排除指令輸出尾端的 *newline* 字元。

在此範例中，**uname** 指令會執行兩次：一次是針對 **netaddr** 參數，另一次是針對 **netaddr6** 參數。 因為每一個參數都具有其專屬的內容以及其專屬 **<Command>** 元素，所以指令會執行兩次。若要參考只需要執行 **uname** 指令一次的範例，請參閱 artex_catalog_elem_PropertyDef.dita。

<PropertyDef> 元素

<PropertyDef> 元素會定義一個可以在 **<Property>** 元素中使用的內容。

語法

母元素：**<Catalog>**

支援下列屬性：

表 *52.* 屬性			
屬性	**必要**	**類型**	**說明**
名稱	是	字串	指定內容的名稱。

支援下列子元素：

表 53. 子元素		
子元素	**必要**	**說明**
<Command>	否	指令
<Argument>	否	指令行引數
<Stdin>	否	<Stdin> 元素所支援的引數
<Filter>	否	過濾器
<Mask>	否	輸出擷取遮罩

註 : 必須針對每一個內容定義 **<Command>** 元素：在 **<ParameterDef>** 層次、在 **<CfgMethod>** 層次，或在 **<PropertyDef>** 元素中。

用法

內容是與參數相關聯的鍵值組。鍵值組的值是透過 **artexget -r** 和 **artexget -n** 指令來擷取，並儲存在輸出設定檔中。可以使用 %p[property_name] 序列，將儲存在設定檔中的內容值插入到指令行中。

<PropertyDef> 元素會定義一個內容。 此內容之後可以使用具有相同 name 屬性的 **<Property>** 元素，與參數或配置方法相關聯。

範例

下列範例會將 *nodeId* 內容指派給 **netaddr** 和 **netaddr6** 參數：

```
<PropertyDef name="nodeId">
  <Command>/usr/bin/uname -f</Command>
  <Mask>.*</Mask>
</PropertyDef>

<ParameterDef name="netaddr" type="string" targetClass="device" cfgmethod="attr">
  <Property name="nodeId"/>
</ParameterDef>

<ParameterDef name="netaddr6" type="string" targetClass="device" cfgmethod="attr">
  <Property name="nodeId"/>
</ParameterDef>
```

在此範例中，**uname** 指令只會執行一次，因為這兩個參數會對其內容使用相同的 **<Command>** 元素，且對這兩個參數所產生的指令行是一樣的。

產生指令行

AIX Runtime Expert 架構根據外部指令來擷取、設定及選擇性地比較參數值。此主題說明如何根據編目檔中所提供的語法資訊來建置指令行。

作業

針對每一個參數，可定義下列作業：

- Get **type="current"**，用來擷取參數的現行值。
- Get **type="nextboot"**，用來擷取參數在重新開機之後將具有的值。
- Set **type="current"**，用來設定參數的現行值。重新開機期間會遺失此參數值。
- Set **type="nextboot"**，用來設定參數在重新開機之後將具有的值。
- Set **type="permanent"**，用來設定參數的現行值，並瞭解在重新開機之後將會持續保存此值。
- **diff** 作業，用來比較參數的兩個值。
- discover 作業，用來尋找支援它們的參數目標。
- Property，用來擷取參數內容。
- Prerequisite，用來決定針對給定的參數執行 get、set 或 discover 作業的條件。

並非所有作業都需要針對所有參數來定義。 必須定義參數所支援的兩個 **get** 作業和所有 **set** 作業。**diff** 為選用作業，如果未定義該作業，則會根據參數類型（例如，string 及 integer）在內部執行參數值之間的比較。只針對具有目標的參數，才必須定義 **discover** 作業。內容和必要條件只在需要時才定義。

指令行元素

針對每一個參數所支援的作業，最多可以使用五個不同的元素來定義如何建置指令行以執行作業：

- **<Command>** 元素，用來定義處理參數的基本指令。
- **<Stdin>** 元素，用來定義將寫入至指令行標準輸入的資料。
- **<Argument>** 元素，用來將參數特定的資料插入至 **<Command>** 或 **<Stdin>** 元素。
- **<Filter>** 元素，用來過濾 **get** 及 **diff** 作業的指令行輸出。
- **<Mask>** 元素，用來從 **get**、**diff** 及 **property** 作業的指令行輸出擷取資料。

需要執行作業時，針對所要求作業定義的 **<Command>**、**<Stdin>**、**<Argument>** 及 **<Filter>** 元素會相互結合，以產生指令行集（如指令行產生演算法主題中所說明）。然後，所產生的指令行會由 Shell 來執行。對於 **get**、**diff** 及 **property** 作業，會使用 **<Mask>** 元素，從指令輸出中擷取所要求的資料（參數值、比較結果或內容值）。

配置方法

指令行元素可於本端的 **<ParameterDef>** 元素內定義，或使用 *cfgmethod* 屬性繼承自 **<CfgMethod>** 元素（參照於 **<ParameterDef>** 元素中）。

允許下列組合：針對特定參數的特定作業所定義的指令行元素集是在 **<ParameterDef>** 元素下於本端所定義的指令行元素聯集，以及針對 **<CfgMethod>** 元素（由 **<ParameterDef>** 元素的 *cfgmethod* 屬性所參照）中的相同作業所定義的指令行元素。如果在本端和配置方法中已定義相同指令行元素，則以本端定義為優先。

例如，在此非最佳化編目檔中：

```
<CfgMethod id=” vmo” >
  <Get type=” nextboot” >
    <Command>/usr/sbin/vmo -r%a</Command>
    <Mask name="1" value="2">[[:space:]]*(.*) = (.*)</Mask>
  </Get>

  <Set type=” permanent” >
    <Command>/usr/sbin/vmo –p –o%a</Command>
    <Argument> -o %n=%p</Argument>
  </Set>

</CfgMethod>

<ParameterDef name=” lgpg_size”  cfgmethod=” vmo” >
  <Get type=” current” >
    <Command>/usr/sbin/vmo -o lgpg_size</Command>
    <Mask name="1" value="2">[[:space:]]*(.*) = (.*)</Mask>
  </Get>

  <Get type=” nextboot” >
    <Argument> -o lgpg_size</Argument>
  </Get>

</ParameterDef>
```

我們可以得知：

- **<Get type="current">** 作業完全在 **<ParameterDef>** 層次上定義。
- **<Get type="nextboot">** 作業有部分元素是在 **<CfgMethod>** 層次（**<Command>** 及 **<Mask>**）上定義，而部分元素是在 **<ParameterDef>** 層次 (**<Argument>**) 上定義。
- **<Get type="current">** 作業完全在 **<CfgMethod>** 層次上定義。

使用配置方法可提供兩個主要的優點：

- 它可以簡化型錄。在許多情況下，參數定義將會從配置方法中繼承其所有指令行元素，而且 **<ParameterDef>** 元素會是空的。

· 它可容許不同的參數在可能的情況下於單一指令行中組合在一起。

指令行產生演算法

指令行使用演算法來產生，該演算法容許數個參數分組在單一指令中。

參數分組不只是從效能和效率的觀點來看有其需要性，對某些參數而言也是必要的。例如，無法單獨設定 **vmo** 參數 *lgpg_regions* 及 *lgpg_size*，它們需要在單一 **vmo** 指令呼叫中一起設定。

指令行產生演算法於功能上相當於下列步驟：

1. 輸入設定檔中的每一個參數皆有其部分展開的 **<Command>** 及 **<Stdin>** 元素。在此階段期間，將會忽略 %a、%v1[name]、%v2[name]、%f1[name] 和 %f2[name] 序列，而且不會展開它們。
2. 驗證下列所有五個條件的參數已分組在一起：
   - · 使用相同 **<Command>** 元素的參數。
   - · 使用相同 **<Stdin>** 元素的參數。
   - · 使用相同 **<Filter>** 元素的參數。
   - · 在步驟 1 期間展開 **<Command>** 元素已產生相同的字串。
   - · 在步驟 1 期間展開 **<Stdin>** 元素已產生相同的字串。

   該群組目前具有其本身部分展開的 **<Command>** 及 **<Stdin>** 元素，以及其本身的 **<Filter>** 元素（與群組中所有參數共用）。
3. 針對參數的每一個群組，群組 **<Command>** 及 **<Stdin>** 元素具有已展開的 %v1[name]、%v2[name]、%f1[name] 及 %f2[name] 序列。只會在群組內搜尋參數名稱。
4. 針對參數的每一個群組，群組 **<Command>** 及 **<Stdin>** 元素具有已展開的 %a 序列：群組中的每一個參數具有其已展開的 **<Argument>** 元素，而且這些已展開 **<Argument>** 元素的連結會取代 **<Command>** 及 **<Stdin>** 元素中的任何 %a 序列。

此處理程序的結果為指令行集，其具有的選擇性資料可在其標準輸入上寫入，以及可以過濾其輸出的指令。

展開指令行元素

<Command>、**<Stdin>** 及 **<Argument>** 元素支援由 AIX Runtime Expert 架構展開的特殊順序，以產生最終指令行。

下列表格是所有支援順序的簡短參照。 如需順序的詳細資料，請參閱下節。

表 *54.* 序列

序列	展開至
%%	文字 % 字元。
%a	所有參數已展開的 **Argument** 字串連結，可於相同指令行中處理。
%n	參數名稱。
%v1	參數值。
%v2	參數的第二個值。只對 **diff** 作業有效。
%f1	將包含參數值的暫存檔名稱。
%f2	將包含參數第二個值的暫存檔名稱。只對 **diff** 作業有效。
%v1[name]	參數名稱的值。
%v2[name]	參數名稱的第二個值。只對 **diff** 作業有效。
%f1[name]	將包含參數名稱值的暫存檔名稱。

表 *54.* 序列 *(*繼續*)*	
序列	**展開至**
%f2[name]	將包含參數名稱第二個值的暫存檔名稱。只對 **diff** 作業有效。
%t[class]	目標類別的目標實例名稱。
%p[name]	內容 *name* 的值。
%c	型錄 ID。

跳出 % 序列

在要插入（透過 %a 序列）至 **<Command>** 元素的 **<Command>** 元素或 **<Argument>** 元素內使用由 AIX Runtime Expert 展開的參數名稱、參數值及目標名稱時，會以單引號括住這些名稱。此作法是為了確保這些字串將會以單一字組的形式傳遞至 Shell，即使它們包括空格或其他特殊字元，也一樣會以單一字組的方式傳遞。此外，會適當地跳出已展開表示式內的任何單引號字元。

型錄撰寫者必須小心不要在引號內的字串使用 %n、%v1、%v2、%v1[name]、%v2[name] 或 %t[class] 序列。 如果必須在字串內使用這些序列，則字串必須在 % 序列之前以引號括住，如下列範例所示：

```
echo "Parameter "%n" is set to "%v1
```

否則，將導致不正確的指令行並具有安全風險。

%% 序列

%% 序列會展開成為文字 % 字元。

例如，字串：

```
/bin/ps -aeF"%%a"
```

會展開成為下列字串：

```
/bin/ps -aeF"%a"
```

%a 序列

可以在 <Command> 字串或 <Stdin> 字串中使用 %a 序列。將以所有參數的所有已展開 <Argument> 字串的連結來替代，這些參數可以在相同指令中處理（如需參數分組的正式說明，請參閱產生指令行主題）。

例如下列型錄（請注意可以使用 %n 序列來簡化）：

```
<CfgMethod id=" vmo" >
  <Get type=" current"
    <Command>/usr/sbin/vmo%a</Command>
  </Get>
</CfgMethod>
<ParameterDef name=" lgpg_size"  cfgmethod=" vmo" >
  <Get type=" current" >
    <Argument> -o lgpg_size</Argument>
  </Get>
</ParameterDef>
<ParameterDef name=" lgpg_regions"  cfgmethod=" vmo" >
  <Get type=" current" >
    <Argument> -o lgpg_regions</Argument>
  </Get>
</ParameterDef>
```

以及下列設定檔：

```
<Parameter name=" lgpg_size"  />
<Parameter name=" lgpg_regions"  />
```

將產生 "get current" 作業的下列指令行：

```
/usr/sbin/vmo –o lgpg_size –o lgpg_regions
```

%n 序列

%n 序列會以參數名稱來替代。

使用 %n 序列，%a 區段的範例可以如下所示進行簡化：

```
<CfgMethod id=” vmo” >
<Get type=” current” >
    <Command>/usr/sbin/vmo%a</Command>
    <Argument> -o %n</Argument>
</Get>
</CfgMethod>
<ParameterDef name=” lgpg_size”  cfgmethod=” vmo”  />
<ParameterDef name=” lgpg_regions”  cfgmethod=” vmo”  />
```

使用下列設定檔：

```
<Parameter name=” lgpg_size”  />
<Parameter name=” lgpg_regions”  />
```

將產生下列指令行以取得現行作業：

```
/usr/sbin/vmo –o ’ lgpg_size’  –o ’ lgpg_regions’
```

%v1 及 %v2 序列

%v1 序列以參數值來替代。

%v2 序列只對 **<Diff>** 作業有效，而且會以參數的第二個值來替代。

例如下列型錄：

```
<CfgMethod id=” vmo” >
  <Set type=” permanent” >
    <Command>/usr/sbin/vmo -p%a</Command>
    <Argument> -o %n=%v1</Argument>
  </Set>
</CfgMethod>
<ParameterDef name=” lgpg_size”  cfgmethod=” vmo”  />
<ParameterDef name=” lgpg_regions”  cfgmethod=” vmo”  />
```

使用下列設定檔：

```
<Parameter name=” lgpg_size”  value=” 16M” />
<Parameter name=” lgpg_regions”  value=” 128”  />
```

將產生 **set permanent** 作業的下列指令行：

```
/usr/sbin/vmo –p –o ’ lgpg_size’ =’ 16M’  –o ’ lgpg_regions’ =’ 128’
```

%f1 及 %f2 序列

%f1 及 %f2 序列會以執行指令之前所建立的暫存檔名稱來替代。檔案內容為 %f1 的參數值及 %f2 的參數的第二個值。%f2 序列只能用於 **<Diff>** 作業。

例如下列型錄：

```
<ParameterDef name=” some_file” >
  <Diff>
    <Command>/usr/bin/diff %f1 %f2</Command>
```

```
  </Diff>
</ParameterDef>
```

當 **artexdiff** 在兩個設定檔（包括具有不同值的相同參數）之間執行時：

```
<Parameter name=" some_file"  value=" foo"  />
<Parameter name=" some_file"  value=" bar"  />
```

然後，會建立兩個分別包含“foo”及“bar”字串的暫存檔（/tmp/file1 及 /tmp/file2，實際的檔名將會不同），接著會執行下列指令：

```
/usr/bin/diff /tmp/file1 /tmp/file2
```

%v1[name] 及 %v2[name] 序列

%v1[name] 序列以參數名稱值來替代。

%v2[name] 序列只對 **<Diff>** 作業有效，而且會以參數名稱的第二個值來替代。

當配置指令同時接受數個參數時，這些序列會很有用，不過需要將某些序列放在於指令行的特殊位置。 此內容以 **chcons** 指令的案例作為範例，其需要將主控台裝置的路徑或檔案路徑放在指令行的最後面。使用 %v1[name] 序列，就可以下列方式撰寫 **chcons** 型錄：

```
<CfgMethod id=" chcons" >
  <Set type=" nextboot" >
    <Command>/usr/sbin/chcons%a %v1[console_device]</Command>
    <Argument> -a %n=%v1</Argument>
  </Set>
</CfgMethod>
<ParameterDef name=" console_device"  cfgmethod=" chcons"  reboot=" true"  />
<ParameterDef name=" console_logname"  cfgmethod=" chcons"  reboot=" true"  />
<ParameterDef name=" console_logsize"  cfgmethod=" chcons"  reboot=" true"  />
```

使用下列設定檔：

```
<Parameter name=" console_device"  value=" /dev/vty0" />
<Parameter name=" console_logname"  value=" /var/adm/ras/conslog"  />
<Parameter name=" console_logverb"  value=" 9"  />
```

此型錄會針對 **set nextboot** 作業產生下列指令行：

```
/usr/sbin/chcons –a ' console_logname' =' /var/adm/ras/conslog'  –a ' console_logverb' =' 9'  /dev/
vty0
```

%f1[name] 及 %f2[name] 序列

%f1[name] 及 %f2[name] 序列會以執行指令之前所建立的暫存檔名稱來替代。檔案內容為 %f1[name] 的參數名稱值及 %f2[name] 的參數名稱的第二個值。%f2[name] 序列只能用於 **<Diff>** 作業。

%t[class] 序列

%t[class] 序列會以針對目標類別處理的目標實例名稱來替代。

%t[class] 序列是供適用於特定物件的參數使用，而不是整個系統。此範例是 **chuser** 指令，其參數適用於特定登錄（檔案、LDAP）的特定使用者（root、訪客）。**chuser** 指令的型錄可以如下所示寫入：

```
<CfgMethod id=" chuser" >
  <Set type=" permanent" >
<Command>/usr/bin/chuser –R %t[module]%a %t[user]</Command>
    <Argument> %n=%v1</Argument>
  </Set>
</CfgMethod>
<ParameterDef name=" shell"  cfgmethod=" chuser"  targetClass=" module,user" >
<ParameterDef name=" histsize"  cfgmethod=" chuser"  targetClass=" module,user"  />
```

使用下列設定檔，其可在 LDAP 及檔案登錄中針對使用者 *adam* 及 *bob* 設定 Shell 及 *histsize* 參數：

```
<Parameter name=" shell"  value=" /usr/bin/ksh" >
  <Target class=" module"  instance=" LDAP"  />
  <Target class=" user"  instance=" adam"  />
</Parameter>
<Parameter name=" histsize"  value=" 5000" >
  <Target class=" module"  instance=" LDAP"  />
  <Target class=" user"  instance=" adam"  />
</Parameter>
<Parameter name=" shell"  value=" /usr/bin/ksh" >
  <Target class=" module"  instance=" files"  />
  <Target class=" user"  instance=" adam"  />
</Parameter>
<Parameter name=" histsize"  value=" 5000" >
  <Target class=" module"  instance=" files"  />
  <Target class=" user"  instance=" adam"  />
</Parameter>
<Parameter name=" shell"  value=" /usr/bin/bash" >
  <Target class=" module"  instance=" LDAP"  />
  <Target class=" user"  instance=" bob"  />
</Parameter>
<Parameter name=" histsize"  value=" 10000" >
  <Target class=" module"  instance=" LDAP"  />
  <Target class=" user"  instance=" bob"  />
</Parameter>
<Parameter name=" shell"  value=" /usr/bin/bash" >
  <Target class=" module"  instance=" files"  />
  <Target class=" user"  instance=" bob"  />
</Parameter>
<Parameter name=" histsize"  value=" 10000" >
  <Target class=" module"  instance=" files"  />
  <Target class=" user"  instance=" bob"  />
</Parameter>
```

這會執行下列指令：

```
/usr/bin/chuser -R ' LDAP'  ' shell' =' /usr/bin/ksh'  ' histsize' =' 5000'  ' adam'
/usr/bin/chuser -R ' files'  ' shell' =' /usr/bin/ksh'  ' histsize' =' 5000'  ' adam'
/usr/bin/chuser -R ' LDAP'  ' shell' =' /usr/bin/bash'  ' histsize' =' 10000'  ' bob'
/usr/bin/chuser -R ' files'  ' shell' =' /usr/bin/bash'  ' histsize' =' 10000'  ' bob'
```

請注意產生四個指令的方式。原因是 %t[module] 及 %t[user] 序列已用於 **<Command>** 字串，這表示每一個指令都是特定模組和使用者的特定項目。由於這個原因，只有適用於相同模組和使用者的參數會分組在一起。

%p[name] 序列

%p[name] 序列會被輸入設定檔中所指定的內容名稱值所替代。例如，下列必要條件會使用 %p[nodeId] 序列來檢查本端系統的節點 ID（由 **uname -f** 指令傳回）是否符合設定檔的 nodeId 內容中所儲存的節點 ID：

```
<PrereqDef id="nodeId">
  <Command>[[ `/usr/bin/uname -f` = %p[nodeId] ]]</Command>
  <ErrMessage>Parameter cannot be applied to a different node</ErrMessage>
</PrereqDef>
```

%c 序列

%c 序列會被參數所屬的編目檔 ID 所替代。這是設定檔中所指定的型錄 ID，它可以與使用型錄繼承時實際定義參數的型錄 ID 不同。

例如，下列必要條件會使用 %c 序列來檢查目標裝置的 *uniquetype* 是否符合編目檔的名稱：

```
<PrereqDef id="devUniqueType">
  <Command>[[ "devParam.`/usr/sbin/lsdev -F uniquetype -l %t[device] | /usr/bin/tr / .`" =
%c ]]</Command>
  <ErrMessage>Parameter cannot be applied to a different device type</ErrMessage>
</PrereqDef>
```

指令和處理程序

指令 為執行運算作業或執行程式的要求。處理程序 為實際在電腦上執行的程式或指令。

您可使用指令，告知作業系統您希望它執行的作業。輸入指令後，指令直譯器（亦稱為 *shell*）會將這些指令解譯，然後處理該作業。

作業系統可以同時執行不同的處理程序。

作業系統可讓您藉由使用輸入/輸出指令與符號，操作資料的輸入和輸出，往返於您的系統。您可藉由指定彙集資料的位置，以資料控制輸入。例如，您可以指定要讀取在鍵盤（標準輸入）上輸入的輸入資料，或是讀取來自檔案的輸入資料。您可藉由指定於何處顯示或儲存資料，以控制資料的輸出。例如，您可指定寫入輸出資料至畫面上（標準輸出），或寫入至檔案中。

指令

某些指令只需簡單鍵入一個字組即可。也可能合併指令，使一指令的輸出成為另一個指令的輸入。

合併指令可讓某個指令的輸出變成另一個指令的輸入，即所謂的管線。

旗標可進一步定義指令的動作。旗標是在指令行上與指令名稱一起使用的修飾元，通常前面會有一連字號。

指令也可以集中起來，並儲存在一檔案中。這些檔案就是所謂的 *shell* 程序或 *shell Script*。而不是單獨執行指令，您所執行的檔案已包含該指令。

如需輸入指令，請在提示下鍵入指令名稱，然後按 Enter 鍵。

```
$ CommandName
```

相關概念

shell 特性

使用 shell 作為系統介面有不少的優點。

相關工作

建立及執行 shell script

shell script 是一種包含一或多個指令的檔案。shell script 提供一個簡單的方法，讓您執行冗長單調的指令、大量或複雜的指令順序，以及例行作業。當您輸入 shell script 檔的名稱時，系統會執行檔案所包含的指令序列。

指令語法及指令名稱

雖然某些指令只需簡單鍵入一個字組即可，但其他指令則需使用旗標與參數。每個指令都有一個語法，指定必要及選用性旗標與參數。

指令的一般格式如下所示：

```
指令名稱 旗標 參數
```

以下是指令的一般規則：

- 指令、旗標和參數之間的空格是很重要的。
- 兩個指令可以在同一行內輸入，只要用分號 (;) 將指令隔開即可。例如：

  ```
  $ CommandOne;CommandTwo
  ```

  shell 將依序執行指令。
- 指令需區分大小寫。shell 將區分大寫字母與小寫字母。對 shell 而言，`print` 與 `PRINT` 或 `Print` 不同。
- 您可以使用反斜線 (\) 字元，在一個以上的指令行上輸入一個非常長的指令。反斜線，對 shell 而言，表示接續前一行。下例是一個跨越兩行的指令：

  ```
  $ ls Mail info temp \
  ( 按 Enter 鍵 )
  > diary
  ( 會出現 > 提示 )
  ```

> 字元為您的次要提示 （$ 為非 root 使用者的預設主要提示），指出現行行接續自上一行。請注意，csh (C shell) 沒有次要提示，且斷行必須是在字詞界限的地方， 其主要提示為 %。

每一指令的第一個字組為指令名稱。有些指令只有指令名稱。

指令旗標
指令名稱後面可以接許多旗標。旗標可修改指令的作業，有時可稱之為選項。

旗標是由空格或標籤隔開，且通常是以連字號 (-) 開始。例外的有：**ps**、**tar** 和 **ar**，它們不需要在某些旗標前加上連字號。例如，在下列指令中：

```
ls -a -F
```

ls 為指令名稱，**-a -F** 則為旗標。

當指令使用旗標時，則直接接續在指令名稱之後。指令中的單一字元旗標，可使用連字號合併在一起。例如，先前的指令也可以寫成：

```
ls -aF
```

某些情況參數的開頭實際上是連字號 (-)。在這個情況下，請在參數前面使用定界字元：兩個連字號 (--)。-- 會告知指令，後面接的不是旗標，而是參數。

例如，如果您要建立一個名稱為 -tmp 的目錄，因此您輸入下列指令：

```
mkdir -tmp
```

系統會顯示類似下列的錯誤訊息：

```
mkdir: Not a recognized flag: t
Usage: mkdir [-p] [-m mode] Directory ...
```

輸入指令的正確方法如下：

```
mkdir -- -tmp
```

您的新目錄，-tmp，已完成建立。

指令參數
在指令名稱之後，可能是許多的旗標，其後跟隨著參數。參數有時候稱為引數或運算元。參數可指定執行指令所需的資訊。

如果您不指定參數，指令可能會採用預設值。例如，在下列指令中：

```
ls -a temp
```

ls 是指令名稱、**-a** 是旗標，而 *temp* 則是參數。此指令會顯示 temp 目錄中的所有 (**-a**) 檔案。

在下列範例中：

```
ls -a
```

因為沒有提供參數，所以預設值即為現行目錄。

在下列範例中：

```
ls temp mail
```

未指定旗標，且 *temp* 及 *mail* 為參數。在此情形下，temp 與 mail 為兩個不同的目錄名稱。**ls** 指令將顯示每個目錄中的所有檔案，隱藏式檔案除外。

只要參數或選項引數為一數值，或包含數值，該數字將解譯為十進位整數，除非另行指定。範圍從 0 到 INT_MAX 的數字（如 /usr/include/sys/limits.h 檔案中所定義），在語法上會辨識為數值。

如果您要使用的指令接受負數作為參數或選項引數，您就可以使用在範圍 INT_MIN 到 INT_MAX 之間的數字（二者皆定義於 /usr/include/sys/limits.h 檔案中）。這並不一定表示此範圍內的所有數字在語法上都是正確的。某些指令具有一內建的規格，允許較小範圍的數字，例如部分的列印指令。如果錯誤產生，錯誤訊息將可讓您知道該數值已超出支援的範圍，而不是表示指令不正確的。

用法陳述式

用法陳述式是一種代表指令語法的方式，它們是由像方括弧 ([ ])、大括弧 ({ }) 及垂直線 (|) 這類符號所組成。

以下為 **unget** 指令的用法陳述式範例：

unget [**-r***SID*] [**-s**] [**-n**] *File ...*

下列慣例使用於指令用法陳述式中：

- 必須在指令行上逐字輸入的項目會以**粗體**來表示。這些項目包括指令名稱、旗標及文字字元。
- 代表必須以名稱來置換變數的項目會以斜體 表示。這些項目包括旗標後面的參數及指令所讀取的參數，例如 *Files* 及 *Directories*。
- 含括在方括弧中的參數為選用性的。
- 包圍在大括弧中的參數是必要的。
- 未包圍在方括弧 ([) 或大括弧中的參數是必要的。
- 垂直線表示您僅選取一個參數。例如，[a | b] 表示您可以選擇 a、b 或不選。同樣地，{ a | b } 表示您必須選擇 a 或 b。
- 省略符號 (. . .) 表示參數可以在指令行上重複。
- 連字號 (-) 代表標準輸入。

***Shutdown* 指令**

如果您具有 root 使用者權限，則可使用 **shutdown** 指令來停止系統。如果您無權使用 **shutdown** 指令，就只要登出作業系統，然後讓它繼續執行即可。

小心：請不要還沒關機就關閉系統。關閉系統會結束系統上正在執行的全部處理程序。如果系統上有其他使用者或者在背景執行的工作，則資料可能會遺失。停止系統之前請執行正確關機程序。

在提示時鍵入：

```
shutdown
```

當 **shutdown** 指令完成並且作業系統停止時，您會收到下列訊息：

```
....Shutdown completed....
```

請參閱 **shutdown** 指令，以取得完整語法。

尋找另一個指令或程式（*whereis* 指令）

whereis 指令可尋找指定檔案的來源端、二元運算子與手冊區段。指令會試圖自標準位置清單中尋找所需程式。

請參閱下列範例：

- 若要尋找現行目錄中沒有文件的檔案，請鍵入：

  ```
  whereis -m -u *
  ```

- 若要尋找名稱中有 Mail 的所有檔案，請鍵入：

  ```
  whereis Mail
  ```

  系統會顯示如下的資訊：

  ```
  Mail: /usr/bin/Mail /usr/lib/Mail.rc
  ```

請參閱 *Commands Reference, Volume 6* 中的 **whereis** 指令，以取得完整語法。

顯示指令的相關資訊（*man* 指令）

man 指令會顯示指令、子常式及檔案的相關資訊。

man 指令的一般格式如下：

```
man CommandName
```

若要取得 **pg** 指令的相關資訊，請鍵入：

```
man pg
```

系統會顯示如下的資訊：

```
    pg 指令

    目的

    顯示的格式檔案。
    語法

  pg [ - Number ] [ -c ] [ -e ] [ -f ] [ -n ] [ -p String ]
  [ -s ] [ +LineNumber | +/Pattern/ ] [ File ... ]


    說明

    pg 指令會從 File 參數讀取檔名，以及
    將該檔案寫入標準輸出，一次一個畫面。如果您
      指定 - ( 連字號 ) 作為 File 參數，或是執行 pg 指令
    而不加選項，pg 指令便會讀取標準輸入。每一個
      畫面後面會接著一個提示。如果您按 Enter 鍵，
      即顯示另一頁。搭配 pg 指令使用的次指令
      可讓您複查或搜尋檔案內容。
```

請參閱 *Commands Reference, Volume 3* 中的 **man** 指令，以取得完整語法。

顯示指令的功能（*whatis* 指令）

whatis 指令會在您使用 **catman -w** 指令所建立的資料庫中，查看由 **Command** 參數指定的給定指令、系統呼叫、程式庫函數或特殊檔名。

如需 **catman -w** 指令的相關資訊，請參閱 **catman -w**。**whatis** 指令顯示手冊區段的標頭行。然後，您可以發出 **man** 指令，以取得相關資訊。如需 **man** 指令的相關資訊，請參閱 **man**。

whatis 指令相當於使用 **man -f** 指令。

若要瞭解 **ls** 指令的功能，請鍵入：

```
whatis ls
```

系統會顯示如下的資訊：

```
ls(1)  - 顯示目錄的內容。
```

請參閱 *Commands Reference, Volume 6* 中的 **whatis** 指令，以取得完整語法。

清單之前輸入的指令（*history* 指令）

使用 **history** 指令來列出您之前已輸入過的指令。

history 指令是 Korn shell 內建指令，它會列出最後 16 個輸入的指令。Korn shell 將您所輸入的指令，儲存在指令歷程檔案，該檔案通常稱為 $HOME/.sh_history。當您需要重複之前的指令時，使用此指令可節省您的時間。

根據預設值，若是非 root 使用者，Korn shell 會儲存最後 128 個指令的文字， 若是 root 使用者，則儲存 512 個指令。雖然非常大的歷程檔案可能會導致 Korn shell 啟動減緩， 歷程檔案大小（由 *HISTSIZE* 環境變數所指定）並沒有限制。

註： Bourne shell 不支援指令歷程。

若要列出先前執行的指令，請在提示時鍵入：

```
歷程
```

由本身輸入的 **history** 指令，將列出先前 16 個輸入的指令。系統會顯示如下的資訊：

```
928   ls
929   mail
930   printenv MAILMSG
931   whereis Mail
932   whatis ls
933   cd /usr/include/sys
934   ls
935   man pg
936   cd
937   ls | pg
938   lscons
939   tty
940   ls *.txt
941   printenv MAILMSG
942   pwd
943   history
```

這份報表首先顯示指令後面的 $HOME/.sh_history 檔案內的指令位置。

若要列出前五個指令，請在提示時鍵入：

```
history -5
```

會顯示類似下列之清單：

```
939   tty
940   ls *.txt
941   printenv MAILMSG
942   pwd
943   history
944   history -5
```

history 指令後接一個數字，將會從該數字開始，列出之前輸入的所有指令。

若要列出 938 之後的指令，請在提示時鍵入：

```
history 938
```

會顯示類似下列之清單：

```
938   lscons
939   tty
940   ls *.txt
941   printenv MAILMSG
942   pwd
943   history
944   history -5
945   history 938
```

相關概念

作業系統 shell
您的作業系統介面稱為 *shell*。

指令歷程替代
使用 **fc** 內建指令來列出或編輯部分的歷程檔。如果要選取編輯或列出檔案的一部分，請指定編號或指令的第一個或前幾個字元。

使用 *r* 別名來重複指令

使用 **r** Korn shell 別名，以重複先前的指令。

輸入 `r` 再按 Enter 鍵，您就可以指定號碼、第一個字元或指令的其中幾個字元。

如果您要列出目前可在系統上使用的顯示器，請在提示下輸入 `lsdisp`。系統會在畫面上傳回資訊。如果您希望再次傳回相同的資訊，請在提示時鍵入：

```
r
```

系統將重新執行最近輸入的指令。在本例中，會執行 **lsdisp** 指令。

若要重複 **ls *.txt** 指令，請在提示時鍵入：

```
r ls
```

r Korn shell 別名將尋找最新帶有字元或指定字元開頭的指令。

使用 *r* 別名來替換字串

您可以使用 **r** Korn shell 別名，在執行指令前修改指令。

在此情況下，可在執行指令之前，使用格式為 *Old=new* 的替代參數來修正指令。

下列範例顯示如何使用 **r** 別名：

- 如果指令行 940 是 **ls** `*.txt`，而您想要執行 **ls** `*.exe`，請在提示時鍵入：

  ```
  r txt=exe 940
  ```

  則將執行指令 940，將 `exe` 替換為 `txt`。
- 如果指令行 940 是開頭為 *l* 這個小寫字母的最新指令，您也可以輸入：

  ```
  r txt=exe l
  ```

註： 只有最先出現的 *Old* 字串才會換成 *New* 字串。輸入不含特定指令號碼或字元的 **r** Korn shell 別名，會在輸入先前的指令時執行替代。

編輯指令歷程

使用 **fc** Korn shell 內建指令，以列出或編輯部分指令歷程檔案。

如果要選取編輯或列出檔案的一部分，請指定編號或指令的第一個或前幾個字元。您可指定單一指令或指令範圍。

如果您未指定編輯器程式作為 **fc** Korn shell 內建指令的引數，將會使用由 *FCEDIT* 變數指定的編輯器。如果沒有定義 *FCEDIT* 變數，將會使用 `/usr/bin/ed` 編輯器。編輯後的指令將在您結束編輯器時列印與執行。使用 **printenv** 指令，以顯示 *FCEDIT* 變數的值。

下列範例說明如何編輯指令歷程：

- 如果您要執行指令：

  ```
  cd /usr/tmp
  ```

  而這個指令與指令行 933 非常類似，請在提示時鍵入：

  ```
  fc 933
  ```

  在此處，您的預設編輯器將出現在指令行 933。將 `include/sys` 變更為 `tmp`，然後當您結束您的編輯器時，便會執行編輯後的指令。
- 您也可指定您在 **fc** 指令中所希望使用的編輯器。比方說，如果想要使用 `/usr/bin/vi` 編輯器來編輯指令，請在提示時鍵入：

  ```
  fc -e vi 933
  ```

  在此處， `vi` 編輯器將出現在指令行 933。

· 您也可以指定編輯指令範圍。例如，如果您想要編輯指令 930 到 940，請在提示時鍵入：

```
fc 930 940
```

在此處，您的預設編輯器將出現指令行 930 至 940。當您結束編輯器時，所有的出現在您編輯器內的指令，都將依序執行。

建立指令別名（*alias shell* 指令）

alias 可讓您建立指令、檔名或任何 shell 文字的快速鍵名稱。藉由使用別名，在執行經常性工作時，您可節省許多時間。您可以建立指令別名。

使用 **alias** Korn Shell 內建指令，將一個字定義為某個指令的別名。您可使別名重新定義內建指令，但無法重新定義保留字。

別名的第一個字元可以是除了 meta 字元以外的任何可列印字元。任何剩餘字元必須同樣是有效的檔名。

用來建立別名的格式如下：

```
alias Name=
String
```

其中 *Name* 參數指定別名的名稱，而 *String* 參數則指定字串字元。如果 *String* 包含空格，請將它含括在引號內。

以下是建立別名的範例：

· 若要建立指令 **rm -i** 的別名（在刪除檔案前提示您），請在提示時鍵入：

```
alias rm="/usr/bin/rm -i"
```

在此範例中，每當您輸入指令 **rm** 時，實際執行的指令是 /usr/bin/rm -i。

· 若要針對指令 **ls -alF | pg**（顯示現行目錄下所有檔案， 包括隱藏檔的詳細資訊；用 * 來標示可執行檔並用 / 標示目錄；以及每次捲動一個畫面）建立別名 **dir**，請在提示時鍵入：

```
alias dir="/usr/bin/ls -alF | pg"
```

在此範例中，每當您輸入指令 **dir** 時，實際執行的指令是 /usr/bin/ls -alF | pg。

· 若要顯示您所有的別名，請在提示時鍵入：

```
alias
```

系統會顯示如下的資訊：

```
rm="/usr/bin/rm -i"
dir="/usr/bin/ls -alF | pg"
```

相關概念

Korn shell 或 POSIX shell 中的指令別名
Korn shell 或 POSIX shell 可讓您建立別名，以自訂指令。

文字格式化的國際字元支援

您可以利用文字格式化指令，來使用由國際擴充字集（用於歐洲語言）所編製的文字。

國際擴充字集，提供許多歐洲語言所使用的字元與符號，以及由英文字元、數字與標點符號所組成的 ASCII 子集。

歐洲擴充字集中的所有字元都有 ASCII 格式。這些格式均可用以代表輸入的擴充字元，或這些字元可由支援歐洲擴充字元的鍵盤裝置直接輸入。

下列文字格式指令，支援所有使用單位元組字元的國際化語言。這些指令位於 /usr/bin 之中。這些以星號 (*) 識別的指令支援多位元組語言的文字處理程序。

```
addbib*         hyphen          pic*            pstext
checkmm         ibm3812         ps4014          refer*
checknr*        ibm3816         ps630           roffbib*
col*            ibm5587G*       psbanne         soelim*
colcrt          ibm5585H-T*     psdit           sortbib*
deroff*         indxbib*        psplot          tbl*
enscript        lookbib*        psrev           troff*
eqn*            makedev*        psroff          vgrind
grap*           neqn*           psrv            xpreview*
hplj            nroff*
```

清單中未含的文字格式指令與巨集套裝軟體，都無法處理國際字元。

相關概念

文字格式化的多位元組字元支援

某些文字格式化指令，可以用來處理多位元組語言的文字。

含擴充單位元組字元的文字格式化

如果您的輸入裝置支援歐洲語言擴充字元集的字元，您可直接輸入。

否則，請使用下列美國國家標準交換碼 (ASCII) ESC 序列格式，代表這些字元：

格式 \[*N*]，其中 *N* 為該字元的 2 或 4 位數十六進位碼。

附註：系統已不再支援 NCesc 格式 \<xx>。

包含擴充字元的文字，根據使用中的語言格式化慣例輸出。未定義介面至特定輸出裝置的字元，將不會產生輸出或錯誤指示。

雖然要求、巨集套裝軟體與指令名稱均是英文，但大部分均能接受包含歐洲擴充字元集字元的輸入（如檔名與參數）。

就 **nroff** 與 **troff** 指令與其前置處理器而言， 輸入的指令必須是 ASCII，否則將會產生無法復原的語法錯誤。國際化字元，不論是單位元組或多位元組，只要括住在引號內，及在其他格式化文字內，均可輸入。例如，從 **pic** 指令使用巨集：

```
define foobar % SomeText %
```

在 define 指引之後，指定的名稱 foobar 必須是 ASCII。然而，置換文字 SomeText 可以包含非美國國家標準交換碼的字元。

文字格式化的多位元組字元支援

某些文字格式化指令，可以用來處理多位元組語言的文字。

在文字格式化的國際字元支援下的清單中的這些指令是以星號 (*) 來識別。清單中未含的文字格式化指令，都無法處理國際字元。

如果您的輸入裝置支援，可直接輸入多位元組字元。否則，您可以用 ASCII 格式 \[*N*] 來輸入任何多位元組字元，其中 *N* 為該字元的 2、4、6、7 或 8 位數十六進位編碼。

雖然要求、巨集套裝軟體與指令名稱均是英文，但大部分均能接受包含任何多位元組字元類型的輸入（如檔名與參數）。

如果您已熟悉以單位元組文字使用文字格式指令，下列清單將彙總多位元組語言環境中，值得留意或唯一的特性。

- 文字並不以連字號連接。
- 多位元組數字輸出需要特殊格式型類。日文格式可供使用。
- 文字依水平線，由左至右方式輸出。
- 字元間隔是固定的，所以字元會在直欄中自動對齊。
- 未定義介面至特定輸出裝置的字元，將不會產生輸出或錯誤指示。

相關概念
文字格式化的國際字元支援
您可以利用文字格式化指令，來使用由國際擴充字集（用於歐洲語言）所編製的文字。

顯示行事曆
使用 **cal** 指令，可以將行事曆寫入至標準輸出。

Month 參數指定行事曆的月份。這可以是 1 到 12（分別表示 1 月到 12 月）的數字。如果未指定 **Month**，則 **cal** 指令會預設為目前的月份。

Year 參數則指定行事曆的年份。因為 **cal** 指令可以顯示 1 到 9999 任何一年的行事曆，所以請鍵入完整的年份，而不要只鍵入最後兩位數。如果未指定 **Year**，則 **cal** 指令會預設為現在這一年。

下列範例顯示如何使用 **cal** 指令：

1. 若要在工作站上顯示 2002 年 2 月的行事曆，請鍵入：

   ```
   cal 2 2002
   ```

2. 按 Enter 鍵。
3. 若要列印 2002 年的行事曆，請鍵入：

   ```
   cal 2002 | qprt
   ```

4. 按 Enter 鍵。

請參閱 *Commands Reference, Volume 1* 中的 **cal** 指令，以取得完整語法。

顯示提示訊息
讀取 calendar 檔案，即可顯示提示訊息。使用 **calendar** 指令，即會在起始目錄中建立此檔案。此指令會將檔案中包含今天或明天日期的所有行都寫入至標準輸出。

您可以讀取 calendar 檔案，使用 **calendar** 指令，即可在起始目錄中建立此檔案。此指令會將檔案中包含今天或明天日期的所有行都寫入至標準輸出。

calendar 指令可辨識日期格式，例如 Dec. 7 或 12/7。也可辨識後面緊接著斜線 (/) 的星號特殊字元 (*)。例如，它會將 */7 解譯為表示每個月的第七天。

星期五時，**calendar** 指令會寫入所有包含星期五、星期六、星期日及星期一日期的行號。不過，此指令無法辨識假日。假日時，此指令會如常運作，而且只提供隔天的排程。

使用一般 calendar 檔案

一般 calendar 檔案可能如下：

```
*/25 - Prepare monthly report
Aug. 12 - Fly to Denver
aug 23 - board meeting
Martha out of town - 8/23, 8/24, 8/25
8/24 - Mail car payment
sat aug/25 - beach trip
August 27 - Meet with Simmons
August 28 - Meet with Wilson
```

若要執行 **calendar** 指令，請鍵入：

```
calendar
```

如果今天是 8 月 24 日星期五，則 **calendar** 指令會顯示下列資訊：

```
*/25 - Prepare monthly report
Martha out of town - 8/23, 8/24, 8/25
8/24 - Mail car payment
sat aug/25 - beach trip
August 27 - Meet with Simmons
```

使用包含 include 陳述式的 calendar 檔案

包含 include 陳述式的 calendar 檔案可能如下：

```
#include </tmp/out>
1/21 -Annual review
1/21 -Weekly project meeting
1/22 *Meet with Harrison in Dallas*
Doctor's appointment - 1/23
1/23 -Vinh's wedding
```

若要執行 **calendar** 指令，請鍵入：

```
calendar
```

如果今天是 1 月 21 日星期三，則 **calendar** 指令會顯示下列資訊：

```
Jan.21 Goodbye party for David
Jan.22 Stockholder meeting in New York
1/21 -Annual review
1/21 -Weekly project meeting
1/22 *Meet with Harrison in Dallas*
```

calendar 指令的結果指出 /tmp/out 檔案包含下列各行：

```
Jan.21 Goodbye party for David
Jan.22 Stockholder meeting in New York
```

請參閱 *Commands Reference, Volume 1* 中的 **calendar** 指令，以取得完整語法。

對數字進行因數分解

您可以使用 **factor** 指令，對數字進行因數分解。

當呼叫 **Number** 參數而未對其指定任何值時，**factor** 指令會等待您輸入一個小於 1E14 (100,000,000,000,000) 的正數。然後，它會將該數字的質因數寫入至標準輸出。它會按次序顯示每個因數，如果相同因數使用一次以上，則顯示適當次數。若要結束，請輸入 0（零）或任何非數值字元。

使用引數進行呼叫時，**factor** 指令會判斷 **Number** 參數的質因數、將結果寫入標準輸出，然後結束。

以下是如何計算因數的範例：

1. 若要計算 123 的質因數，請鍵入：

   ```
   factor 123
   ```

2. 按 Enter 鍵。即會顯示下列項目：

   ```
   123 3 41
   ```

請參閱 *Commands Reference, Volume 2* 中的 **factor** 指令，以取得完整語法。

依關鍵字尋找指令

您可以使用 **apropos** 指令，顯示在其標題中包含任何給定關鍵字 的線上指令說明區段。

apropos 指令不區分每個字的大小寫。亦顯示其他字組的部分字組。例如，尋找字組 *compile* 時，**apropos** 指令也會尋找字組 *compiler* 的所有實例。

註： 包含關鍵字的資料庫是 /usr/share/man/whatis，必須先使用 **catman -w** 指令產生。

apropos 指令相當於搭配使用 **man** 指令與 **-k** 選項。

例如，若要尋找其標題包含字組 *password* 的手冊小節，請執行下列指令：

```
apropos password
```

請參閱 *Commands Reference, Volume 1* 中的 **apropos** 指令，以取得完整語法。

處理程序

實際在電腦上執行的程式或指令，稱為處理程序。

處理程序存在於親子階層中。 由程式或指令所啟動的處理程序稱為上代程序；子處理程序是上代程序的產品。一個上代程序可以有數個子處理程序，但一個子處理程序只能有一個上代程序。

系統會於每一個處理程序啟動時，指派一個處理程序 ID（PID 號碼）。如果您啟動相同的程式數次，則該程式每一次都具有不同的 PID 號碼。

當系統上啟動一個處理程序時，該程序會使用部分可用的系統資源。執行一個以上的處理程序時，會在作業系統中建立一個排程程式，以基於已建立的優先順序來提供電腦的時間配額給每一個程序。可以使用 **nice** 或 **renice** 指令來改變這些優先順序。

註： 若要將處理程序的優先順序變高，您必須具有 root 使用者權限。所有的使用者都可以使用 **nice** 指令，降低所要啟動之處理程序的優先順序，或是使用 **renice** 指令，降低已啟動之處理程序的優先順序。

下列清單說明處理程序的類型：

前端與背景處理程序

需要使用者來啟動或與其互動的處理程序，就叫做前景處理程序。不需使用者介入執行的處理程序則稱為背景處理程序。依預設值而言，程式和指令是以前景處理程序的形式來執行。如果要在後端中執行處理程序，請在您用來啟動處理程序的指令名稱尾端加上一個 &。

常駐程式處理程序

常駐程式是無人式執行的處理程序。它持續存在後端中，並且隨時可用。常駐程式通常會在系統啟動時開始執行，並於系統停止時結束。一般來說，常駐程式處理程序執行系統服務，並且隨時可供一個以上的工作或使用者來使用。常駐程式處理程序是由 root 使用者或 root shell 來啟動，並且只能由 root 使用者來停止。例如，**qdaemon** 處理程序提供對系統資源（例如印表機）的存取權限。另一個常見的常駐程式是 **sendmail** 常駐程式。

休眠程序

休眠程序是不再執行的停用處理程序，但仍在處理程序表格中仍可辨識（換句話說，其有 PID 號碼）。沒有任何系統空間配置給它。休眠程序已被刪除或已結束，並且繼續存在於處理程序表格中，直到上代程序消失或系統關機並重新啟動為止。使用 **ps** 指令列出時，修眠程序會顯示成 <defunct>。

處理程序啟動

您可於系統提示上輸入程式名稱或指令名稱，來從顯示站啟動前景處理程序。

啟動前景處理程序之後，處理程序會在您的顯示站上與您互動，直到完成為止。除非處理程序完成或您將它中止，否則顯示站上不會發生任何其他互動（例如，輸入另一個指令）。

單一使用者一次可以執行數個處理程序，每個使用者最多可執行 40 個處理程序（預設值）。

在前景中啟動處理程序

若要在前景啟動處理程序，請鍵入指令名稱，以及適當的參數與旗標：

```
$ CommandName
```

在背景啟動處理程序

若要在背景執行處理程序，請鍵入指令名稱，並加上適當的參數和旗標， 後面再接著一個 & 符號 (&)：

```
$ CommandName&
```

當處理程序在背景執行時，您可以在顯示站上輸入其他指令，以執行其他作業。

一般而言，對於需花費長時間來執行的指令，後端處理程序非常有用。然而，因為背景處理程序會增加處理器執行的總工作量， 背景處理程序會降低系統的其餘部分。

大部分處理程序將它們的輸出導入標準輸出，即使它們是在後端中執行。除非重新導向，否則標準輸出會移至顯示裝置。由於來自後端處理程序的輸出會干擾您在系統上的其他工作，因此最好將後端處理程序的輸出重新導向一個檔案或印表機。然後，您即可隨時查看輸出。

註： 在某些情況下，在背景中和前景中執行的處理程序，可能會以不同的順序來產生它的輸出。程式設計師可能會想要使用 **fflush** 子常式，以確保無論處理程序是在前景或後背景中執行，都會以適當的次序來產生輸出。

當背景處理程序正在執行時，您可以使用 **ps** 指令來檢查它的狀態。

檢查處理程序狀態的指令（*ps* 指令）

任何時候，系統在執行時，處理程序也會執行。您可以使用 **ps** 指令來尋找正在執行中的處理程序，並顯示這些處理程序的相關資訊。

ps 指令具有數個旗標，可讓您指定列出哪些處理程序，以及顯示每一個處理程序的哪些資訊。

若要顯示所有在系統上執行的處理程序，請在提示時鍵入：

```
ps -ef
```

系統會顯示如下的資訊：

```
    USER   PID  PPID   C    STIME    TTY  TIME CMD
        root     1     0   0   Jun 28      -  3:23 /etc/init
        root  1588  6963   0   Jun 28      -  0:02 /usr/etc/biod 6
        root  2280     1   0   Jun 28      -  1:39 /etc/syncd 60
        mary  2413 16998   2 07:57:30      -  0:05 aixterm
        mary 11632 16998   0 07:57:31  lft/1  0:01 xbiff
        mary 16260  2413   1 07:57:35  pts/1  0:00 /bin/ksh
        mary 16469     1   0 07:57:12  lft/1  0:00 ksh /usr/lpp/X11/bin/xinit
        mary 19402 16260  20 09:37:21  pts/1  0:00 ps -ef
```

上列輸出資料的直欄定義如下：

項目	說明
USER	使用者登入名稱
PID	處理程序 ID
PPID	母項處理程序 ID
C	處理程序的 CPU 使用率
STIME	處理程序的開始時間
TTY	處理程序的控制工作站
TIME	處理程序的執行時期總計
CMD	指令

在前一個範例中，**ps -ef** 指令的處理程序 ID 是 19402。它的母項處理程序 ID 是 16260，也就是 /bin/ksh 指令。

如果報表很長，則頂端部分會捲出畫面之外。若要一次顯示一頁（畫面），請使用 **ps** 指令加上 **pg** 指令。在提示時鍵入：

```
ps -ef | pg
```

若要顯示所有在系統上執行之處理程序的狀態資訊，請在提示時鍵入：

```
ps gv
```

此種指令格式列出每一個作用中處理程序的許多統計資料。此指令的輸出值看起來像下面這個樣子：

```
      PID    TTY STAT  TIME PGIN  SIZE   RSS   LIM  TSIZ   TRS %CPU %MEM COMMAND
        0      - A     0:44    7     8     8    xx     0     0  0.0  0.0 swapper
        1      - A     1:29  518   244   140    xx    21    24  0.1  1.0 /etc/init
      771      - A     1:22    0    16    16    xx     0     0  0.0  0.0 kproc
     1028      - A     0:00   10    16     8    xx     0     0  0.0  0.0 kproc
     1503      - A     0:33  127    16     8    xx     0     0  0.0  0.0 kproc
     1679      - A     1:03  282   192    12 32768   130     0  0.7  0.0 pcidossvr
     2089      - A     0:22  918    72    28    xx     1     4  0.0  0.0 /etc/sync
```

```
    2784      - A     0:00    9    16     8    xx     0     0  0.0  0.0 kproc
    2816      - A     5:59 6436  2664   616     8   852   156  0.4  4.0 /usr/lpp/
    3115      - A     0:27  955   264   128    xx    39    36  0.0  1.0 /usr/lib/
    3451      - A     0:00    0    16     8    xx     0     0  0.0  0.0 kproc
    3812      - A     0:00   21   128    12 32768    34     0  0.0  0.0 usr/lib/lpd/
    3970      - A     0:00    0    16     8    xx     0     0  0.0  0.0 kproc
    4267      - A     0:01  169   132    72 32768    16    16  0.0  0.0 /etc/sysl
    4514  lft/0 A     0:00   60   200    72    xx    39    60  0.0  0.0 /etc/gett
    4776  pts/3 A     0:02  250   108   280     8   303   268  0.0  2.0 -ksh
    5050      - A     0:09 1200   424   132 32768   243    56  0.0  1.0 /usr/sbin
    5322      - A     0:27 1299   156   192    xx    24    24  0.0  1.0 /etc/cron
    5590      - A     0:00    2   100    12 32768    11     0  0.0  0.0 /etc/writ
    5749      - A     0:00    0   208    12    xx    13     0  0.0  0.0 /usr/lpp/
    6111      - T     0:00   66   108    12 32768    47     0  0.0  0.0 /usr/lpp/
```

請參閱 *Commands Reference, Volume 4* 中的 **ps** 指令，以取得完整語法。

設定處理程序的起始優先順序（*nice* 指令）

您可以將處理程序的起始優先順序設定為比基本排程優先順序低的值。

若要將處理程序的起始優先順序設定為比基本排程優先順序低的值，請使用 **nice** 指令來啟動處理程序。

註： 若要以高於基本排程優先順序的優先順序來執行處理程序，您必須具有 root 使用者權限。

若要設定處理程序的初始優先順序，請鍵入：

```
nice -n Number CommandString
```

其中 *Number* 的範圍從 0 至 39，而 39 的低優先順序最低。*nice* 值（十進位值）是系統排定的處理程序優先順序。數字愈高，優先順序就愈低。如果您使用零，則處理程序將以其基本排程優先順序來執行。*CommandString* 是您要執行的指令、旗標以及參數。

請參閱 *Commands Reference, Volume 4* 中的 **nice** 指令，以取得完整語法。

您亦可使用 **smit nice** 指令來執行此作業。

變更執行中處理程序的優先順序（*renice* 指令）

您可以使用指令行中的 **renice** 指令來變更執行中處理程序的排程優先順序， 改變的值可低於或高於基本排程優先順序。此指令改變處理程序的 nice 值。

註： 若要以較高的優先順序來執行處理程序，或變更不是由您啟動之處理程序的優先順序，您必須具有 root 使用者權限。

若要變更執行中處理程式的優先順序，請鍵入：

```
renice Priority -p ProcessID
```

其中的 *Priority* 的數字是從 -20 到 20。數字愈高，優先順序就愈低。如果您使用零，則處理程序將以其基本排程優先順序來執行。*ProcessID* 是您要變更其優先順序的 PID。

您亦可使用 **smit renice** 指令來執行此作業。

前景處理程序取消

如果您啟動了前景處理程序，然後決定不要完成它，您可按 INTERRUPT 來取消。這通常是 Ctrl-C 或 Ctrl-Backspace。

註： INTERRUPT (Ctrl-C) 不會取消背景處理程序。若要取消背景處理程序，您必須使用 **kill** 指令。

大部分的簡式指令都執行地非常快，在您取消它們之前就已經完成了。因此，本節範例所使用的指令會花費較多的時間來執行：**find / -type f**。此指令會顯示系統上所有檔案的路徑名稱。您不需要為了完成此步驟而去學習 **find** 指令，因為該指令在此的目的只是為了示範如何使用處理程序而已。

在下列範例中，**find** 指令會啟動一個處理程序。在處理程序執行幾秒鐘之後，您可按下 INTERRUPT 鍵來取消：

```
$ find / -type f
/usr/sbin/acct/lastlogin
/usr/sbin/acct/prctmp
/usr/sbin/acct/prdaily
```

```
/usr/sbin/acct/runacct
/usr/sbin/acct/sdisk
/usr/sbin/acct/shutacct INTERRUPT (Ctrl-C)
$ _
```

系統將提示返回畫面。現在您可輸入其他指令。

相關工作

列出您的終端機所用的控制鍵指定（stty 指令）

若要顯示您的終端機設定，請使用 **stty** 指令。請特別注意您的終端機是用哪些按鍵來作為控制鍵。

停止前景處理程序的鍵盤指令

可以令處理程序停止，但無法從處理程序表格中移除其處理程序 ID (PID)。您可使用鍵盤上的 Ctrl-Z 來停止前景處理程序。

註 : Ctrl-Z 可在 Korn shell (**ksh**) 及 C shell (**csh**) 中執行， 但不能在 Bourne shell (**bsh**) 中執行。

重新啟動已停止的處理程序

此程序說明如何重新啟動已透過 Ctrl-Z 所停止的處理程序。

註 : Ctrl-Z 可在 Korn shell (**ksh**) 及 C shell (**csh**) 中執行， 但不能在 Bourne shell (**bsh**) 中執行。若要重新啟動已停止的處理程序，您必須是啟動該處理程序的使用者，或具有 root 使用者權限。

1. 若要顯示系統上正在執行或停止但尚未刪除之所有處理程序，請鍵入：

   ```
   ps -ef
   ```

   您可將此指令加入 **grep** 指令， 清單限制在您最有可能想要重新啟動的那些處理程序。例如，如果您想要重新啟動 **vi** 階段作業，您可輸入：

   ```
   ps -ef | grep vi
   ```

   這個指令只會顯示 **ps** 指令輸出中，有包含 `vi` 字組的那些行。輸出看起來如下所示：

   ```
   UID    PID   PPID   C      STIME       TTY  TIME  COMMAND
   root   1234  13682  0      00:59:53    -    0:01  vi test
   root  14277  13682  1      01:00:34    -    0:00  grep vi
   ```

2. 在 **ps** 指令輸出中，尋找您要重新啟動的處理程序，並且注意其 PID 號碼。 在本範例中，PID 是 1234。
3. 若要傳送 CONTINUE 信號給已停止的處理程序，請鍵入：

   ```
   kill -19 1234
   ```

   將 1234 替換成您的處理程序 PID。 -19 指出 CONTINUE 信號。此指令會在背景重新啟動處理程序。如果處理程序可以在背景中執行，則表示您已完成該程序。如果處理程序必須在前景中執行（如 **vi** 階段作業），則必須進行下一步。
4. 若要將處理程序帶至前景，請鍵入：

   ```
   fg 1234
   ```

   再一次將 1234 替換成您的處理程序之 PID。您的處理程序現在應該在前景中執行。（您現在位於 **vi** 編輯階段作業中）。

排定稍後作業的處理程序

您可設定一個處理程序成為批次處理，於已排定時間在後端中執行。

at 及 **smit** 指令可讓您輸入稍後要執行的指令名稱，並允許您指令應該執行的時間。

註 : `/var/adm/cron/at.allow` 及 `/var/adm/cron/at.deny` 檔案控制著您是否可以使用 **at** 指令。具有 root 使用者權限的人，可以建立、編輯或刪除這些檔案。這些檔案中的項目是使用者登入名稱，且一行一個名稱。下列是 `at.allow` 檔案的一個範例：

```
root
nick
```

```
dee
sarah
```

如果 at.allow 檔案存在，則只有登入名稱列在此檔案中的使用者，才能使用 **at** 指令。系統管理者可將使用者的登入名稱列在 at.deny 檔案中，以明確地阻止使用者使用 **at** 指令。如果只有 at.deny 檔案存在，則名稱未出現在檔案中的任一使用者，皆可使用 **at** 指令。

如果下列任一項為真，您就不能使用 **at** 指令：

· at.allow 檔案及 at.deny 檔案皆不存在（僅允許 root 使用者）。

· at.allow 檔案存在，但未列出使用者的登入名稱。

· at.deny 檔案存在，並且列出使用者的登入名稱。

如果 at.allow 檔案不存在，且 at.deny 檔案不存在或是空的，則僅具有 root 使用者權限的人員可以使用 **at** 指令提交工作。

at 指令語法可讓您指定日期字串、時間和日期字串、或增量字串，以指出您要何時執行處理程序。它亦可讓您指定所要使用的 shell 或佇列。下列各範例顯示該指令的一些典型用法。

例如，如果您的登入名稱是 joyce，而且您想在午夜執行名稱為 WorkReport 的 Script，請執行下列步驟：

1. 輸入您要程式開始執行的時間：

   ```
   at midnight
   ```

2. 鍵入您要執行的程式名稱，每一個名稱之後按 Enter 鍵。在鍵入暱稱之後，按檔案結尾字元 (Ctrl-D)，以指出清單尾端。

   ```
   WorkReport^D
   ```

   按下 Ctrl-D 之後，系統會顯示類似下列的資訊：

   ```
   job joyce.741502800.a at Fri Jul  6 00:00:00 CDT 2002.
   ```

   會給予程式 WorkReport 一個工作號碼 joyce.741502800.a，並且將於 7 月 6 日的午夜執行。

3. 若要列出已送出將於稍後執行之程式，請鍵入：

   ```
   at -l
   ```

   系統會顯示如下的資訊：

   ```
   joyce.741502800.a       Fri Jul  6 00:00:00 CDT 2002
   ```

請參閱 **at** 指令，以取得完整語法。

相關工作

列出所有排定的處理程序（at 或 atq 指令）
將 **-l** 旗標搭配 **at** 指令或搭配 **atq** 指令，可以列出所有排定的處理程序。

從排程中移除處理程序
您可以使用 **at** 指令加上 **-r** 旗標，來移除已排定的處理程序。

列出所有排定的處理程序（*at* 或 *atq* 指令）

將 **-l** 旗標搭配 **at** 指令或搭配 **atq** 指令，可以列出所有排定的處理程序。

這兩個指令都提供相同的輸出；但是，**atq** 指令可根據發出 **at** 指令的時間來排序處理程序，並且僅顯示佇列中的處理程序數目。

您可使用下列方法來列出全部已排定的處理程序：

· 自指令行執行 **at** 指令

· 使用 **atq** 指令

at 指令

若要列出已排程的處理程序，請鍵入：

```
at -l
```

此指令列出您的佇列中全部已排定的處理程序。如果您是 root 使用者，則此指令列出全部使用者的已排定處理程序。請參閱 **at** 指令，以取得有關語法的完整明細資訊。

atq 指令

請參閱下列有關如何使用 **atq** 指令的範例：

· 若要列出佇列中所有已排程的處理程序，請鍵入：

```
atq
```

· 如果您是 root 使用者，則您可列出特定使用者的佇列中已排定之處理程序，請鍵入：

```
atq UserName
```

· 若要列出佇列中已排程的處理程序數目，請鍵入：

```
atq -n
```

相關工作

排定稍後作業的處理程序
您可設定一個處理程序成為批次處理，於已排定時間在後端中執行。

從排程中移除處理程序
您可以使用 **at** 指令加上 **-r** 旗標，來移除已排定的處理程序。

從排程中移除處理程序

您可以使用 **at** 指令加上 **-r** 旗標，來移除已排定的處理程序。

請參閱下列有關如何使用 **at** 或 **atq** 指令的範例：

1. 若要移除已排定的處理程序，您必須知道它的處理程序號碼。
   您可以使用 **at -l** 指令或 **atq** 指令，來取得處理程序號碼。
2. 當您已知道所要移除的處理程序號碼時，請鍵入：

```
at -r ProcessNumber
```

您亦可使用 `smit rmat` 指令來執行此作業。

相關工作

列出所有排定的處理程序（at 或 atq 指令）
將 **-l** 旗標搭配 **at** 指令或搭配 **atq** 指令，可以列出所有排定的處理程序。

排定稍後作業的處理程序
您可設定一個處理程序成為批次處理，於已排定時間在後端中執行。

移除背景處理程序（*kill* 指令）

如果 INTERRUPT 沒有中止您的前景處理程序，或您在啟動背景處理程序之後， 決定不要完成該處理程序，您可使用 **kill** 指令來取消處理程序。

您必須先知道 PID 號碼，才能夠使用 **kill** 指令來取消處理程序。**kill** 指令的一般格式如下：

```
kill ProcessID
```

註：

· 若要移除處理程序，您必須具有 root 使用者權限，或是啟動處理程序的使用者。自 **kill** 指令發出給處理程序的預設信號是 -15 (SIGTERM)。

· 若要移除休眠程序，您必須移除其上代程序。

1. 使用 **ps** 指令來決定您要移除之處理程序的處理程序 ID。您可將此指令經由管線傳遞給 **grep** 指令，以便僅列出您要的處理程序。例如，如果您想要 vi 階段作業的處理程序 ID，您可以輸入：

   ```
   ps -l | grep vi
   ```

2. 在下列範例中，您發出 **find** 指令，在背景中執行。然後您決定要取消該處理程序。請發出 **ps** 指令來列出 PID 號碼。

   ```
   $ find / -type f > dir.paths &
   [1]    21593
   $ ps
         PID   TTY  TIME  COMMAND
       1627  pts3  0:00  ps
       5461  pts3  0:00  ksh
     17565  pts3  0:00  -ksh
     21593  pts3  0:00  find / -type f
   $ kill 21593
   $ ps
         PID   TTY  TIME  COMMAND
       1627  pts3  0:00  ps
       5461  pts3  0:00  ksh
     17565  pts3  0:00  -ksh
   [1]  +  Terminated 21593     find  /  -type f > dir.paths  &
   ```

   kill 21593 指令會結束背景 find 處理程序，並且第二個 **ps** 指令不會傳回 PID 21593 的任何狀態資訊。系統不會顯示終止訊息，直到您輸入下一個指令為止，除非該指令是 **cd**。

   kill 指令可讓您取消背景處理程序。如果您發覺已誤放一個處理程序在背景中，或處理程序執行的時間太長，則您可能想要執行此動作。

請參閱 *Commands Reference, Volume 3* 中的 **kill** 指令，以取得完整語法。

也可以輸入下列指令，在 **smit** 中使用 **kill** 指令：

```
smit kill
```

指令及處理程序的指令摘要

下列是指令及處理程序的指令摘要。

表 *55.* 指令的指令摘要

項目	說明
alias	將別名清單列印至標準輸出的 shell 指令
history	顯示歷程事件清單的 shell 指令
man	顯示關於指令、子常式與線上檔案的資訊
whatis	說明指令所執行的功能
whereis	尋找所安裝之程式的來源、二進位運算子或手冊

表 *56.* 處理程序的指令摘要

項目	說明
at	於稍後執行指令、列出所有已排定的處理程序，或從時程表中移除處理程序
atq	顯示等待執行的工作佇列
kill	傳送信號至執行中的處理程序
nice	以較低或較高的優先順序執行指令
ps	顯示處理程序的現行狀態
renice	改變執行處理程序的優先順序

管理系統懸置

系統懸置管理是讓使用者在增進應用程式可用性的同時，仍可持續執行關鍵性任務的應用程式。系統懸置偵測會警示系統管理者可能的問題所在，並允許管理者以 root 的身分登入，或重新啟動系統來解決問題。

shconf 指令

啟用**系統當機偵測**時，即會呼叫 **shconf** 指令。**shconf** 指令會判別哪些事件需要調查，以及發生這類事件時應該採取的動作。您可以指定下列其中一個動作，如檢查的優先順序層次、在較低或同等優先順序上沒有執行任何處理程序或執行緒的逾時設定、執行警告動作的終端機裝置，以及 **getty** 指令動作：

- 在 errlog 檔案中記載錯誤
- 在系統主控台（英數主控台）或指定的 TTY 上顯示警告訊息
- 重新啟動系統
- 提供特殊的 **getty**，允許使用者以 root 身分登入並啟動指令
- 啟動指令

若採用**啟動指令**與**提供特殊的 getty** 選項，系統當機偵測會以最高優先順序啟動特殊的 **getty** 指令或指定的指令。特殊 **getty** 指令會列印警告訊息，表示此為回復中的 **getty**，其執行優先順序為 0。下表擷取了優先順序懸置偵測的各種動作及相關預設參數。每一種偵測類型只能啟用一個動作。

選項	是否啟用	優先順序	逾時（秒）
在 errlog 檔案中記載錯誤	停用	60	120
顯示警告訊息	停用	60	120
提供回復的 getty	啟用	60	120
啟動指令	停用	60	120
重新啟動系統	停用	39	300

註：啟用**在主控台上啟動回復 getty** 時，**shconf** 指令會將 **-u** 旗標新增至與主控台登入相關之 **inittab** 的 **getty** 指令。

若為遺失 IO 的偵測，您可以設定逾時值，並啟用下列動作：

選項	是否啟用
顯示警告訊息	停用
重新啟動系統	停用

shdaemon 常駐程式

shdaemon 常駐程式是 **init** 所啟動的處理程序，其執行優先順序為 0。它負責處理系統懸置偵測，方法是擷取配置資訊、起始工作結構、並啟動使用者所設定的偵測時間。

相關概念

優先順序當機偵測
AIX 可以根據使用者定義的動作， 偵測系統當機狀況，並嘗試從此類狀況中回復。

遺失 I/O 當機偵測
AIX 可以根據使用者定義的動作， 偵測系統當機狀況，並嘗試從此類狀況中回復。

配置系統懸置偵測

您可以從 SMIT 管理工具來管理系統懸置偵測的配置。

SMIT 功能表選項可讓您啟用或停用偵測機制、 顯示此功能的現行狀態，以及變更或顯示現行配置。系統懸置偵測功能表的捷徑是：

smit shd
管理系統懸置偵測

smit shstatus
系統懸置偵測狀態

smit shpriocfg
變更/顯示優先順序問題偵測的性質

smit shreset
還原預設優先順序問題配置

smit shliocfg
變更/顯示遺失 I/O 偵測的性質

smit shlioreset
還原預設的遺失 I/O 偵測配置

您也可以使用 **shconf** 指令來管理系統當機偵測。

優先順序當機偵測

AIX 可以根據使用者定義的動作， 偵測系統當機狀況，並嘗試從此類狀況中回復。

所有處理程序（亦稱為執行緒）均以一定的優先順序執行。在範圍 0-126 內優先順序在數字上是顛倒的。零的優先順序最高，126 的優先順序最低。所有執行緒的預設優先順序均是 60。任何使用者均可使用 **nice** 指令降低處理程序的優先順序。擁有 root 權限的任何人員亦可提高處理程序的優先順序。

核心排程程式總會挑選最高優先順序的可執行緒放置到 CPU 上。因此，高優先順序執行緒的數量達到某種程度時，可能會完全佔用機器，這樣，低優先順序的執行緒就永遠無法執行。如果執行中的執行緒的優先順序高於預設值 60，則可能會封鎖所有正常的 shell 及登入，導致系統似乎處於當機狀態。

「系統當機偵測」特性提供一種偵測機制，並會對系統管理者提供回復方法。執行時，此特性將被視為最優先處理的常駐程式 (**shdaemon**)。此常駐程式會查詢核心，取得特定執行期間內優先順序最低的執行緒。如果優先順序高於配置的臨界值，則常駐程式可採取數個動作中的一個動作。這些動作每個都可以獨立啟用，並可配置觸發時的優先順序及時間間隔。這些動作及其預設值為：

```
     動作                  預設值       預設        預設          預設
                                                  已啟用        優先順序值   逾時值       裝置值

1)   記載錯誤               否          60           2
2)   主控台訊息             否          60           2          /dev/console
3)   高優先順序             是          60           2          /dev/tty0
          登入 shell
4)   以高優先順序           否          60           2
          執行指令
5)   損毀及重新開機         否          39           5
```

相關概念

管理系統懸置
系統懸置管理是讓使用者在增進應用程式可用性的同時，仍可持續執行關鍵性任務的應用程式。系統懸置偵測會警示系統管理者可能的問題所在，並允許管理者以 root 的身分登入，或重新啟動系統來解決問題。

遺失 I/O 當機偵測

AIX 可以根據使用者定義的動作， 偵測系統當機狀況，並嘗試從此類狀況中回復。

發生 I/O 錯誤時，I/O 路徑會受到封鎖，且該路徑上的後續 I/O 資料均會受到影響。 於這些情況下，作業系統必須警示使用者並執行使用者定義的動作。 作為「遺失 I/O」偵測及通知的一部分，**shdaemon** 在「邏輯磁區管理程式」的幫助下，會監視 I/O 緩衝區一段時間，檢查是否有任何 I/O 擱置過久。 若 I/O 的等待時間超過 **shconf** 檔案定義的等待時間臨界值，則視為偵測到遺失 I/O，並執行進一步動作。遺失 I/O 的相關資訊將記載在錯誤日誌中。 同樣，根據 **shconf** 檔案中的設定，系統亦可能重新啟動以從遺失 I/O 狀況中回復。

針對遺失 I/O 偵測，可設定逾時值及啟用下列動作：

動作	預設啟用	預設裝置
主控台訊息	否	/dev/console
損毀及重新開機	否	-

如需系統當機偵測的相關資訊，請參閱第 124 頁的『管理系統懸置』。

相關概念

管理系統懸置

系統懸置管理是讓使用者在增進應用程式可用性的同時，仍可持續執行關鍵性任務的應用程式。系統懸置偵測會警示系統管理者可能的問題所在，並允許管理者以 root 的身分登入，或重新啟動系統來解決問題。

處理程序管理

處理程序是作業系統用來控制系統資源使用的實體。執行緒 可控制處理器時間用量，但是大部分系統管理工具仍需要您參照執行緒正在其中執行的處理程序，而不是執行緒本身。

工具可用於：

- 觀察處理程序的建立、取消、識別及資源用量
  - **ps** 指令是用來報告處理程序 ID、 使用者、CPU 時間用量及其他屬性。
  - **who -u** 指令報告已登入使用者的 shell 處理程序 ID。
  - **svmon** 指令用來報告處理程序的實際記憶體用量。（請參閱 *Performance Toolbox Version 3: Guide and Reference*，以取得 **svmon** 指令的相關資訊。）
  - **acct** 指令機制於處理程序終止時寫入記錄，彙總處理程序的資源使用。
- 控制處理程序爭奪 CPU 的優先順序。
  - **nice** 指令導致使用指定的處理程序優先順序來執行指令。
  - **renice** 指令變更給定處理程序的優先順序。
- 終止不受控制的處理程序。
  - **kill** 指令會向一或多個處理程序傳送終止信號。

處理程序監視

身為系統管理者的您可以管理處理程序。

ps 指令是觀察系統中處理程序的主要工具。**ps** 指令的大部分旗標分成兩種：

- 指定要包含於輸出之處理程序類型的旗標
- 指定顯示處理程序屬性的旗標

就系統管理目的而言，最有用的 **ps** 變式如下：

項目	說明
ps -ef	列出所有非核心處理程序，並附有使用者 ID、處理程序 ID、最近的 CPU 使用情況、CPU 使用總計，以及啟動處理程序的指令（包括其參數）。
ps -fu *UserID*	列出由 *UserID* 所擁有的所有處理程序，並附有處理程序 ID、最近 CPU 使用情況、CPU 使用情況總計，以及啟動處理程序的指令（包括其參數）。

若要識別目前使用 CPU 時間最繁忙的使用者，您可以輸入：

```
ps -ef | egrep -v "STIME|$LOGNAME" | sort +3 -r | head -n 15
```

此指令會以遞減順序列出 15 個非您所有、使用 CPU 最密集的處理程序。

如需特殊使用方法，下面兩個表格彙總了旗標的效用，可以用來簡化選擇 **ps** 旗標的作業。

處理程序指定旗標													
	-A	**-a**	**-d**	**-e**	**-G -g**	**-k**	**-p**	**-t**	**-U -u**	**a**	**g**	**t**	**x**
所有處理程序	Y	-	-	-	-	-	-	-	-	-	Y	-	-

處理程序指定旗標 *(繼續)*													
	-A	**-a**	**-d**	**-e**	**-G -g**	**-k**	**-p**	**-t**	**-U -u**	**a**	**g**	**t**	**x**
非處理程序群組領導者且與終端機無關	-	Y	-	-	-	-	-	-	-	-	-	-	-
非處理程序群組領導者	-	-	Y	-	-	-	-	-	-	-	-	-	-
非核心處理程序	-	-	-	Y	-	-	-	-	-	-	-	-	-
指定處理程序群組成員	-	-	-	-	Y	-	-	-	-	-	-	-	-
核心處理程序	-	-	-	-	-	Y	-	-	-	-	-	-	-
在處理程序號碼清單中的指定	-	-	-	-	-	-	Y	-	-	-	-	-	-
與清單中 tty 相關的	-	-	-	-	-	-	-	Y (*n* ttys)	-	-	-	Y (1 tty)	-
指定的使用者處理程序	-	-	-	-	-	-	-	-	Y	-	-	-	-
終端機的處理程序	-	-	-	-	-	-	-	-	-	Y	-	-	-
與 tty 無關	-	-	-	-	-	-	-	-	-	-	-	-	Y

直欄選取旗標										
Default1	**-f**	**-l**	**-U -u**	**Default2**	**e**	**l**	**s**	**u**	**v**	
PID	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
TTY	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
TIME	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
CMD	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
USER	-	Y	-	-	-	-	-	-	Y	-
UID	-	-	Y	Y	-	-	Y	-	-	-
PPID	-	Y	Y	-	-	-	Y	-	-	-
C	-	Y	Y	-	-	-	Y	-	-	-
STIME	-	Y	-	-	-	-	-	-	Y	-
F	-	-	Y	-	-	-	-	-	-	-
S/STAT	-	-	Y	-	Y	Y	Y	Y	Y	Y
PIR	-	-	Y	-	-	-	Y	-	-	-
NI/NICE	-	-	Y	-	-	-	Y	-	-	-

直欄選取旗標 (繼續)										
Default1	**-f**	**-l**	**-U -u**	**Default2**	**e**	**l**	**s**	**u**	**v**	
ADDR	-	-	Y	-	-	-	Y	-	-	-
SIZE	-	-	-	-	-	-	-	-	Y	-
SZ	-	Y	-	-	-	Y	-	Y	-	-
WCHAN	-	-	Y	-	-	-	Y	-	-	-
RSS	-	-	-	-	-	-	Y	-	Y	Y
SSIZ	-	-	-	-	-	-	-	Y	-	-
%CPU	-	-	-	-	-	-	-	-	Y	Y
%MEM	-	-	-	-	-	-	-	-	Y	Y
PGIN	-	-	-	-	-	-	-	-	-	Y
LIM	-	-	-	-	-	-	-	-	-	Y
TSIZ	-	-	-	-	-	-	-	-	-	Y
TRS	-	-	-	-	-	-	-	-	-	Y
環境（接在指令之後）	-	-	-	-	-	Y	-	-	-	-

如果 **ps** 未指定任何旗標，或指定以減號起首的處理程序旗標，則會顯示出現於 Default1 中的直欄。如果指令指定為非以減號起首的處理程序旗標，則會出現 Default2 中的直欄。**-u** 或 **-U** 旗標是處理程序指定旗標，也是直欄選取旗標。

下面是直欄內容的簡短說明：

項目	**說明**
PID	處理程序 ID
TTY	與處理程序相關聯的終端機或虛擬終端機
TIME	累加的 CPU 使用時間，以分鐘與秒鐘為單位
CMD	處理程序正在執行的指令
USER	處理程序所屬之使用者的登入名稱
UID	處理程序所屬之使用者的數字使用者 ID
PPID	此處理程序的上代程序 ID
C	最近的 CPU 使用時間
STIME	如果小於 24 小時，則為處理程序啟動的時間。否則，則為處理程序啟動的日期
F	8 個字元的十六進位值，說明與處理程序相關的旗標（請參閱 **ps** 指令的詳細說明）
S/STAT	處理程序的狀態（請參閱 **ps** 指令的詳細說明）
PRI	處理程序的現行優先順序值
NI/NICE	處理程序的 NICE 值
ADDR	處理程序堆疊的區段號碼
SIZE	（**-v** 旗標）處理程序之資料區段的虛擬大小（以 KB 為單位）
SZ	（**-l** 及 **l** 旗標）處理程序之核心映像檔的大小（以 KB 為單位）
WCHAN	處理程序正在等待的事件

項目	說明
RSS	記憶體中工作區段與程式區段頁數總和乘以 4
SSIZ	核心堆疊的大小
%CPU	自處理程序啟動後使用 CPU 的時間百分比
%MEM	名義上，此指處理程序使用的實際記憶體百分比，該測量與任何其他記憶體統計資料無關
PGIN	尋頁錯失所造成的進入分頁數。因為所有 I/O 都被分類為尋頁錯失，基本上，這是一種 I/O 磁區的測量方法
LIM	一定是 **xx**
TSIZ	可執行檔的文字區段大小
TRS	程式區段頁數乘以 4
環境	處理程序的所有環境變數值

處理程序優先順序改變
基本上，如果您已識別出使用太多 CPU 時間的處理程序，您可以使用 **renice** 指令來增加它的 NICE 值，以降低它的有效優先順序。

例如：

```
renice +5 ProcID
```

ProcID 的 NICE 值可以將前景處理程序的正常值 20 增加至 25。您必須具有 root 權限，才能將處理程序 *ProcID* 的 NICE 值重設為 20。請鍵入：

```
renice -5 ProcID
```

處理程序終止
通常，您可使用 **kill** 指令來結束處理程序。

kill 指令會傳送信號至指定的處理程序。依據信號類型及處理程序中執行的程式本質， 處理程序可能會結束或繼續執行。您傳送的信號如下：

項目	說明
SIGTERM	（信號 15）要求程式終止。如果程式具有 SIGTERM 的信號處理程式，實際上它並不會終止應用程式，因此 **kill** 可能沒有任何作用。這是由 **kill** 所傳送的預設信號。
SIGKILL	（信號 9）立即結束處理程序的指引。此信號不能被截取或忽略。

一般而言，最好使用 SIGTERM 而非 SIGKILL。如果程式具有 SIGTERM 的處理程式，就可以依序清除及終止。請鍵入：

```
kill -term ProcessID
```

（將省略 **-term**。）如果處理程序不回應 SIGTERM，請鍵入：

```
kill -kill ProcessID
```

您可能注意到處理程序表中偶爾會出現已廢止的處理程序（亦稱為 *zombies* 休眠程序）。這些處理程序雖然已不再執行，也未配置系統空間，但仍保留其 PID 號碼。您可以在處理程序表中辨識出休眠程序，因為它在 CMD 直欄中會顯示 <defunct>。例如：

```
  UID   PID  PPID   C    STIME    TTY  TIME CMD
                              .
                              .
                              .
   lee 22392 20682   0   Jul 10      -  0:05 xclock
   lee 22536 21188   0   Jul 10  pts/0  0:00 /bin/ksh
```

```
 lee 22918 24334   0   Jul 10  pts/1  0:00 /bin/ksh
lee 23526 22536  22                 0:00 <defunct>
 lee 24334 20682   0   Jul 10      ?  0:00 aixterm
   lee 24700     1   0   Jul 16      ?  0:00 aixterm
root 25394 26792   2   Jul 16  pts/2  0:00 ksh
 lee 26070 24700   0   Jul 16  pts/3  0:00 /bin/ksh
 lee 26792 20082   0   Jul 10  pts/2  0:00 /bin/ksh
root 27024 25394   2 17:10:44  pts/2  0:00 ps -ef
```

休眠程序會繼續存在於處理程序表中，直到結束上代處理程序或系統關閉又重新啟動。在上列範例中，上代處理程序 (PPID) 即為 **ksh** 指令。當 Korn shell 結束時，會從處理程序表中移除已廢止的處理程序。

有時您的處理程序表中會聚集一些這樣的已廢止處理程序，因為應用程式已建立了數個子處理程序，且尚未結束。如果這造成問題，最簡單的解決方案就是修改應用程式，使其 **sigaction** 子常式忽略 **SIGCHLD** 信號。

相關資訊

sigaction 指令

連結或切斷處理程序

您可以將處理程序與處理器連結，或切斷之前連結的處理程序。

您必須具有 root 使用者權限，才能連結或切斷非您所有的處理程序。

在多重處理器系統中，您可以使用下列工具來連結處理程序至處理器，或切斷之前連結的處理程序：

- SMIT
- 指令行

註： 將處理程序連結至處理器可能會導致連結處理程序的效能提升（減少硬體快取遺失），但過度使用此機能會造成個別處理器超載，而其他處理器卻未得到充分利用。 如此所造成的瓶頸會減少整體產量與效能。在正常作業期間，最好讓作業系統指派自動執行處理器的處理程序，以分散所有處理器的系統負荷。只連結那些您確定可從單一處理器執行而獲利的處理程序。

連結或切斷處理程序作業		
作業	***SMIT* 捷徑**	**指令或檔案**
連結處理程序	**smit bindproc**	**bindprocessor -q**
切斷處理程序	**smit ubindproc**	**bindprocessor -u**

修正停滯或不想要的處理程序

停滯或不想要的處理程序會造成終端機發生問題。部分問題會在畫面上產生訊息，提供可能原因的相關資訊。

若要執行下列程序，您必須具有第二台終端機、數據機或網路登入。如果您沒有其中任何一項，請重新啟動終端機以修正終端機問題。

選取適當的程序以修正您的終端機問題：

釋放處理程序所接管的終端機

您可以停止已停滯或不想要的處理程序。

請執行下列以識別並停止已停滯或不想要的處理程序：

1. 鍵入下列 **ps** 指令，以判斷在畫面上執行的作用中處理程序：

   ```
   ps -ef | pg
   ```

   ps 指令會顯示處理程序狀態。**-e** 旗標會寫入所有處理程序（核心處理程序除外）的相關資訊，且 **f** 旗標會產生處理程序的完整報表，包括建立處理程序時所使用的指令名稱與參數。**pg** 指令會限制以每次都以單頁顯示輸出，所以資訊不會快速地捲離畫面。

   可疑的處理程序包括使用過量系統資源（如 CPU 或磁碟空間）的系統或使用者處理程序。系統處理程序（如 **sendmail**、**routed** 及 **lpd**）經常會失控。請使用 **ps -u** 指令檢查 CPU 使用情況。

2. 使用 **who** 指令，判斷正在此機器上執行處理程序的使用者：

```
who
```

who 指令顯示目前正在系統上的所有使用者相關資訊，如登入名稱、工作站名稱、登入的日期及時間。

3. 判斷您是否需要停止、暫停或變更使用者處理程序的優先順序。

註：您必須具有 root 權限才能停止非您所有的處理程序。如果您終止或變更使用者處理程序的優先順序，請聯絡處理程序擁有者並解釋您的執行動作。

· 使用 **kill** 指令以停止處理程序。例如：

```
kill 1883
```

kill 指令會傳送信號給執行中的處理程序。若要停止處理程序，請指定處理程序 ID (PID)，在本例中是 1883。使用 **ps** 指令以判斷指令的 PID 號碼。

· 暫停處理程序，並使用 & 符號在背景中執行。例如：

```
/u/bin1/prog1 &
```

& 表示您想要在背景中執行此處理程序。在背景處理程序中，shell 傳回 shell 提示之前並不會等待指令完成。一但處理程序需要一些時間才能完成，請在指令行結尾鍵入 **&**，以在背景中執行該指令。使用一般的 **ps** 指令就可以顯示在背景中執行的工作。

· 使用下列 **renice** 指令，變更已接管之處理程序的優先順序：

```
renice 20 1883
```

renice 指令會變更一或多個執行中處理程序的排程優先順序。數字愈高，優先順序愈低，20 代表最低的優先順序。

在之前的範例中，**renice** 會重新將處理程序號碼 1883 的優先順序排為最低。在有未使用的處理器時間可用時，它才會執行。

回應畫面訊息
使用這個程序，可以回應畫面訊息並自其中回復：

1. 請確定 DISPLAY 環境變數的設定正確無誤。使用下列方法之一來檢查 DISPLAY 環境：

· 使用 **setsenv** 指令來顯示環境變數。

```
setsenv
```

當您登入時，**setsenv** 指令會顯示受保護狀態的環境。

請判斷是否已設定 DISPLAY 變數。在下面範例中並沒有 DISPLAY 變數， 這表示 DISPLAY 變數未設定為特定的值。

```
SYSENVIRON:
NAME=casey
TTY=/dev/pts/5
LOGNAME=casey
LOGIN=casey
```

或

· 變更 DISPLAY 變數的值。例如，將它設定為機器 bastet 與終端機 0，請鍵入：

```
DISPLAY=bastet:0
export DISPLAY
```

如果沒有明確地設定，DISPLAY 環境變數會預設為 unix:0（主控台）。變數值的格式為 *name*:*number*，其中 *name* 為特定機器的主機名稱， *number* 則為指明系統的 X 伺服器號碼。

2. 使用下列 **stty** 指令， 將終端機重設為預設值：

```
stty sane
```

stty sane 指令會還原終端機驅動程式的「正常狀態」。該指令會從 /etc/termcap 檔案（或 /usr/share/lib/terminfo，如果適用的話）輸出適當的終端機重設程式碼。

3. 如果 Return 鍵不能正確運作，請鍵入下列指令來重設：

```
^J stty sane ^J
```

^J 代表 Ctrl-J 鍵順序。

使用環境變數 *RT_MPC* 及 *RT_GRQ* 來執行多個佇列

使用多重佇列會增加執行緒的處理器親緣性， 但可能會發生特殊狀況以致於您想抑制此效果。

只有一個執行佇列時，由另一個執行中執行緒（啟動者執行緒）所啟動的執行緒（啟動中執行緒），通常可以 立即使用啟動者執行緒所執行的 CPU。使用多重執行佇列時，啟動中執行緒可能位於另一個 CPU 的執行佇列上，以致於在下一次排程原則前無法通知啟動中執行緒。這可能造成最多 10 毫秒的延遲。

這類似於此作業系統舊版中，使用 bindprocessor 選項時可能發生的範例情節。如果所有 CPU 都一直在工作中，且有許多互相依賴的執行緒正在啟動，您可以使用兩個選項。

- 第一個選項（使用一個執行佇列）是設定環境變數 RT_GRQ=ON，來強制切斷所選執行緒的連結，將其分派離開整體執行佇列。
- 或者，使用者也可以針對所選的處理程序，選擇即時核心選項（依序鍵入指令 bosdebug -R on 和 bosboot）與 RT_MPC=ON 環境變數。這對於維護系統效能日誌而言非常重要，您可用以緊密監視所嘗試執行的任何調整之影響。

系統統計作業

系統統計作業公用程式可讓您收集與報告各種系統資源的個別與群組使用。

此統計資訊可用來對利用系統資源的使用者記帳，以及監視所選擇的系統作業層面。為輔助記帳的作業，統計作業系統會提供管理群組成員所定義的資源使用情形總計，且如果包括 **chargefee** 指令，則將帳單費用也計算在內。

統計作業系統還會提供資料來評估現行資源的指派是否適當、設定資源限制及限額、預測未來需要，以及訂購印表機及其他裝置的補給品。

下列資訊應有助於您瞭解如何實作系統中的統計公用程式。

統計資料報告

在收集完各種不同類型的統計資料之後，會處理記錄並將記錄轉換成報告。

當數字變大時，統計指令會自動將記錄轉換成科學記號表示法。數字在科學記號表示法中以下面的格式代表：

*Base***e+***Exp*

*Base***e-***Exp*

此數字等於 *Base* 數字與 10 的 **+***Exp* 或 **-***Exp* 次方的乘積。例如，科學記號表示法 1.345e+9 等於 1.345×10^{9} 或 1,345,000,000。科學記號表示法 1.345e-9 等於 1.345×10^{-9} 或 0.000000001345。

相關概念

處理程序統計作業資料

「統計作業」系統會在每個處理程序執行時，收集其資源使用情形的相關資料。

每日統計報告

若要產生每日報告，請使用 **runacct** 指令。

此指令會將資料彙總到一個 ASCII 檔，檔名為 /var/adm/acct/sum(x)/rprt*MMDD*。*MMDD* 指定執行報告的月份及日期。報告包含下列各項：

· 每日報告
· 每日使用情況報告
· 每日指令摘要
· 每月指令摘要總計
· 前次登入

每日報告
每日統計報告包含有關連線時間、處理程序、磁碟使用情形、印表機使用情形，以及該收取的費用。

acctmerg 指令會將有關連線時間、處理程序、 磁碟使用情形、印表機使用情形，以及該收取的費用之原始統計資料合併到每日報告中。由 **runacct** 指令在其每日作業中呼叫的 **acctmerg** 指令會產生如下內容：

/var/adm/acct/nite(x)/dacct
輸入檔之一已滿時產生的中間報告。

/var/adm/acct/sum(x)/tacct
tacct 格式的累計總計報告。**monacct** 指令使用此檔案來產生 ASCII 每月摘要。

acctmerg 指令可在 ASCII 與二進位格式之間轉換記錄，並針對每個使用者將不同來源的記錄合併到單一記錄中。如需 **acctmerg** 指令的相關資訊，請參閱 **acctmerg**。

「每日」報告的第一行是以報告中收集資料的開始與完成時間起首， 接下來是系統層次事件的清單，包括所有現有的關機、重新開機與執行層次變更。也會列出總持續時間，以指出統計作業期間的分鐘總數（如果報告是每 24 小時執行，則通常是 1440 分鐘）。報告含有下列資訊：

項目	說明
LINE	使用中的主控台、tty 或 pty
MINUTES	使用線路的分鐘總數
PERCENT	在統計作業期間，使用線路的時間百分比
# SESS	啟動的新登入階段作業數
# ON	與 **# SESS** 相同
# OFF	登出數加上線路岔斷數

每日使用情況統計報告
「每日使用情況」報告是一份彙總報告，說明在統計期間內，每一個使用者 ID 的系統使用情況。

部分欄位會分成黃金時間及非黃金時間，依照統計作業管理者在 /usr/lib/acct/holidays 目錄中的定義。報告含有下列資訊：

項目	說明
UID	使用者 ID
LOGIN NAME	使用者名稱
CPU (PRIME/NPRIME)	所有使用者處理程序的 CPU 時間總計（以分鐘為單位）
KCORE (PRIME/NPRIME)	執行中處理程序所使用的記憶體總計（以 KB/分鐘為單位）
CONNECT (PRIME/NPRIME)	連接時間總計（使用者登入的時間長度）（以分鐘為單位）
DISK BLOCKS	在啟用統計作業的所有檔案系統上，使用者所使用的總磁碟空間量平均值
FEES	使用 **chargefee** 指令輸入的費用總計
# OF PROCS	屬於此使用者的處理程序總數
# OF SESS	此使用者的不同登入階段作業數

項目	說明
# DISK SAMPLES	統計作業期間，執行的磁碟範例次數。如果未擁有任何 DISK BLOCK，則此值將是 0。

每日指令摘要統計報告
「每日指令摘要」報告顯示在統計作業期間所執行的每一個指令，一行顯示一個唯一的指令名稱。

表格是按 TOTAL KCOREMIN（說明如下）排序， 第一行包含所有指令的總資訊。每一個指令所列出的資料會累加統計作業期間所有執行指令。本表中的直欄包含下列資訊：

項目	說明
COMMAND NAME	執行的指令
NUMBER CMDS	指令執行次數
TOTAL KCOREMIN	執行指令所使用的記憶體總計（以 KB/分鐘為單位）
TOTAL CPU-MIN	指令使用的 CPU 時間總計（以分鐘為單位）
TOTAL REAL-MIN	指令經歷的實際時間總計（以分鐘為單位）
MEAN SIZE-K	每一 CPU 分鐘內，指令所使用的記憶體大小平均值
MEAN CPU-MIN	執行每一個指令的 CPU 分鐘數平均值
HOG FACTOR	當指令作用中時，佔用 CPU 的時間測量。它是 **TOTAL CPU-MIN** 對 **TOTAL REAL-MIN** 的比例
CHARS TRNSFD	具有系統讀取及寫入權的指令所轉送的字元數
BLOCKS READ	指令所執行的實體區塊讀取與寫入數

每月指令摘要總計統計報告
「每月指令摘要總計」是由 **monacct** 指令建立的，提供自前一個每月報告之後所執行的所有指令之相關資訊。

欄位與資訊所代表的意義與「每日指令摘要」中的一樣。

前次登入
「前次登入」報告顯示每一個使用者 ID 的兩個欄位。第一個欄位是 **YY-MM-DD**，指出指定使用者的最近一次登入。第二個欄位是使用者帳戶的名稱。

日期欄位 `00-00-00` 指出該使用者 ID 從未登入。

統計報告摘要
您可以產生一份彙總原始統計資料的報告。

若要彙總原始統計資料，請使用 **sa** 指令。此指令會讀取通常是收集在 `/var/adm/pacct` 檔案中的原始統計資料，也會讀取 `/var/adm/savacct` 檔案中的現行使用情況摘要資料（如果摘要資料存在）。它會合併此資訊與新的使用情況摘要報告，並清除原始資料檔案，以釋出空間供進一步的資料收集使用。

先決要件

sa 指令需要原始統計資料的輸入檔，如 `pacct` 檔案（處理程序統計作業檔案）。若要收集原始統計資料，您必須設定統計作業系統並執行。

程序

sa 指令的目的是彙總處理程序統計作業資訊，並顯示或儲存該資訊。此指令的最簡單用法是顯示讀取 `pacct` 檔案的有效期間內，所執行的每一個處理程序的統計資料清單。若要產生這樣的清單，請鍵入：

```
/usr/sbin/sa
```

若要彙總統計作業資訊並將它合併到摘要檔案，請鍵入：

```
/usr/sbin/sa -s
```

sa 指令提供許多其他旗標，指定如何處理及顯示統計作業資訊。請參閱 **sa** 指令說明，以取得進一步資訊。

相關工作

設定統計作業系統
可以設定統計作業系統。

每月報告

您可以產生「每月」統計報告。

由 **cron** 常駐程式呼叫的 **monacct** 指令會產生下列內容：

項目	說明
/var/adm/acct/fiscal	這是由 **monacct** 指令根據 /var/adm/acct/sum/tacct 報告產生的週期性摘要報告。可以配置 **monacct** 指令每月執行或在會計期間結束時執行。

連線時間報告

統計記錄包括登入、登出、系統關機及上次登入記錄。

runacct 指令會呼叫兩個指令（**acctcon1** 及 **acctcon2**）來處理在 /var/adm/wtmp 檔案中收集的登入、登出及系統關機記錄。**acctcon1** 指令會將這些記錄轉換成階段作業記錄，並將它們寫入 /var/adm/acct/nite(x)/lineuse 檔案。然後，acctcon2 指令會將階段作業記錄轉換成統計記錄總計 /var/adm/logacct， 讓 **acctmerg** 指令將其新增到每日報告中。如需這些指令的相關資訊，請參閱 **runacct**、**acctcon1** 及 **acctcon2**。

如果您是從指令行執行 **acctcon1** 指令，則必須加上 **-l** 旗標，來產生線路使用報告 /var/adm/acct/nite(x)/lineuse。若要產生統計期間的整體階段作業報告 /var/adm/acct/nite(x)/reboots，請使用 **acctcon1** 指令加上 **-o** 旗標。

lastlogin 指令會產生提供每個使用者之前次登入日期的報告。如需 **lastlogin** 指令的相關資訊，請參閱 **lastlogin**。

相關概念

連線時間統計資料
連線時間資料是由 **init** 指令及 **login** 指令收集的。

磁碟使用情形統計作業資料
許多統計資訊都是在使用資源時進行收集的。**dodisk** 指令（由 **cron** 常駐程式指定執行），會定期將每個使用者的磁碟使用情形記錄寫入 /var/adm/acct/nite(x)/dacct 檔案。

磁碟使用情形統計作業報告

在 /var/adm/acct/nite(x)/dacct 檔案中收集的磁碟使用情形記錄，都會藉由 **acctmerg** 指令合併到每日統計報告中。

如需 **acctmerg** 指令的相關資訊，請參閱 **acctmerg**。

印表機使用情況統計報告

/var/adm/qacct 檔案中的 ASCII 記錄可以轉換成總統計記錄，以由 **acctmerg** 指令新增到每日報告中。

如需 **acctmerg** 指令的相關資訊，請參閱 **acctmerg**。

相關概念

印表機使用情形統計資料
印表機使用情形資料的收集需要 **enq** 指令與佇列常駐程式協力完成。

費用統計報告
如果您是使用 **chargefee** 指令向使用者收取諸如檔案還原、諮詢或素材等服務費用，則會在 /var/adm/fee 檔案中寫入 ASCII 統計記錄總計。此檔案由 **acctmerg** 指令新增至每日報告中。

如需 **chargefee** 及 **acctmerg** 指令的相關資訊，請參閱 **chargefee** 及 **acctmerg**。

相關概念
費用統計資料
您可以在 /var/adm/fee 檔案中產生 ASCII 總統計記錄。

財務統計報告
「會計統計報告」通常是每月使用 **monacct** 指令進行收集。

此報告會儲存在 /var/adm/acct/fiscal(x)/fiscrptMM，其中的 *MM* 代表執行 **monacct** 指令的月份。此報告所含的資訊類似整個月所彙總的每日報告。

統計作業系統活動報告
可以產生一份顯示「統計作業」系統活動的報告。

若要產生系統活動的報告，請使用 **prtacct** 指令。此指令會讀取總統計檔（tacct 檔案格式）中的資訊，並產生格式化輸出。總統計檔包含每日的連接時間、處理時間、磁碟使用情況與印表機使用情況報告。

先決要件

prtacct 指令需要 tacct 檔案格式的輸入檔。這意指您已設定並執行統計作業系統，或您過去曾執行統計作業系統。

程序

輸入下列以產生系統活動的相關報告：

```
prtacct -f Specification -v Heading File
```

Specification 是以逗點分隔的欄位號碼或範圍清單，由 **acctmerg** 指令使用。可選用的 **-v** 旗標會產生詳細的輸出，其中會以較高的精準度表示法顯示浮點數字。*Heading* 是您要在報告上顯示的標題，這是可選用的。*File* 是輸入所使用的總統計檔完整路徑名稱。您可以指定多個檔案。

相關工作
設定統計作業系統
可以設定統計作業系統。

支援 *8* 個字元以上的使用者名稱
為了和所有 Script 保有舊版相容性，根據預設值，在統計作業內不會啟用長使用者名稱支援。而會截斷所有使用者 ID 成為保留前 8 個字元。

為了啟用長使用者名稱支援，大部分的指令都已提供額外的 **-X** 旗標， 讓這些指令可以接受及輸出 8 個字元以上的使用者 ID（ASCII 和二進位格式）。此外，當啟用長使用者名稱支援時，指令和 Script 會處理 /var/adm/acct/sumx 、/var/adm/acct/nitex 以及 /var/adm/acct/fiscalx 中的檔案，而不會使用 /var/adm/acct/sum、 /var/adm/acct/nite 以及 /var/adm/acct/fiscal。

統計指令
統計指令以幾種不同的方式工作。

部分指令：

- 針對特定類型的統計作業收集資料或產生報告：連線時間、處理程序、磁碟使用情形、 印表機使用情形或指令使用情形。
- 呼叫其他指令。例如，**runacct** 指令（通常是由 **cron** 常駐程式自動執行）會呼叫許多指令，來收集及處理統計資料並準備報告。若要取得自動統計，您必須先配置 **cron** 常駐程式來執行 **runacct** 指令。 請參閱 **crontab** 指令，取得如何配置 **cron** 常駐程式以固定排程之間隔提交指令的進一步資訊。如需這些指令的相關資訊，請參閱 **runacct**、**cron** daemon 及 **crontab**。
- 執行維護功能並確保作用中資料檔的完整性。

· 使管理群組的成員藉由在鍵盤輸入指令來執行非經常性的作業，如顯示特定記錄。
· 可讓使用者顯示特定的資訊。僅有一個使用者指令（即 **acctcom** 指令）會顯示處理程序統計作業摘要。

自動執行的指令

有數個指令會自動收集統計資料。

通常由 **cron** 常駐程式執行的幾個指令會自動收集統計資料。這些指令為：

runacct
處理主要的每日統計作業程序。**runacct** 指令通常會在非主要時間由 **cron** 常駐程式起始，該指令會呼叫其他幾個統計指令來處理作用中資料檔，並產生指令及資源使用情形摘要（依使用者名稱排序）。它還會呼叫 **acctmerg** 指令來產生每日摘要報告檔，並呼叫 **ckpacct** 指令來維護作用中資料檔的完整性。

ckpacct
處理 pacct 檔案大小。在處理這些記錄時失敗後，如果必須重新啟動 **runacct** 程序，則擁有幾個較小的 pacct 檔案是比較有利的。**ckpacct** 指令會檢查 /var/adm/pacct 作用中資料檔的大小，且如果該檔案大於 500 個區塊，則該指令會呼叫 **turnacct switch** 指令，來暫時停用處理程序統計作業。該資料會轉送到新的 pacct 檔案 /var/adm/pacct *x* (*x* 是一個整數，每次建立一個新的 pacct 檔案時， 就會增加該整數）。如果可用磁碟區塊數在 500 以下，則 **ckpacct** 指令會呼叫 **turnacct off** 指令來停用處理程序統計作業。

dodisk
會呼叫 **acctdisk** 指令以及 **diskusg** 指令或 **acctdusg** 指令，將磁碟使用情形記錄寫入 /var/adm/acct/nite/dacct 檔案。此資料稍後會合併到每日報告中。

dodisk
會呼叫 **acctdisk** 指令以及 **diskusg** 指令或 **acctdusg** 指令，將磁碟使用情形記錄寫入 /var/adm/acct/nite/dacct 檔案。此資料稍後會合併到每日報告中。

monacct
從每日報告產生週期性摘要。

sa1
收集二進位資料，並將其儲存在 /var/adm/sa/sa *dd* 檔案中，其中 *dd* 代表日期。

sa2
將每日報告寫入 /var/adm/sa/sa*dd* 檔案，其中 *dd* 代表日期。該指令會將已存在一週以上的報告從 /var/adm/sa/sa*dd* 檔案移除。

其他指令由 **cron** 常駐程式之外的程序自動執行：

startup
若新增至 /etc/rc 檔案時，**startup** 指令會針對統計作業系統起始啟動程序。

shutacct
藉由呼叫 **acctwtmp** 指令在 /var/adm/wtmp 檔案中寫入一行，來記錄停用統計作業的時間。然後，它會呼叫 **turnacct off** 指令，來停用處理程序統計作業。

鍵盤指令

管理群組的成員可以從鍵盤輸入下列指令。

ac
列印連線時間記錄。提供此指令是為了與 Berkeley Software Distribution (BSD) 系統相容。

acctcom
顯示處理程序統計作業摘要。此指令也可供使用者使用。

acctcon1
顯示連線時間摘要。必須使用 **-l** 旗標或 **-o** 旗標。

accton
啟用及停用處理程序統計作業。

chargefee
針對執行的工作單元，向使用者收取預定的費用。費用會由 **acctmerg** 指令新增至每日報告中。

fwtmp
將檔案在二進位與 ASCII 格式之間轉換。

last
顯示先前登入的相關資訊。提供此指令是為了與 BSD 系統相容。

lastcomm
顯示所執行之最後指令的相關資訊。提供此指令是為了與 BSD 系統相容。

lastlogin
顯示每個使用者上次登入的時間。

pac
準備印表機/繪圖機統計記錄。提供此指令是為了與 BSD 系統相容。

prctmp
顯示階段作業記錄。

prtacct
顯示總統計檔。

sa
彙總原始統計資訊來幫助管理大量的統計資訊。提供此指令是為了與 BSD 系統相容。

sadc
報告各種不同的本端系統動作，如緩衝區使用情形、磁碟及磁帶 I/O 活動、TTY 裝置活動計數器，以及檔案存取計數器。

sar
將作業系統中所選累計活動計數器內容寫入標準輸出。**sar** 指令僅報告本端活動。

time
列印執行指令所需的實際時間、使用者時間及系統時間。

timex
以秒為單位報告經歷時間、使用者時間及執行時期。

相關概念
系統資料收集及報告
您可以設定系統，使其自動收集資料並產生報告。

統計檔
兩個主要的統計目錄為 /usr/sbin/acct 目錄（儲存執行統計作業系統所需的所有 C 語言程式及 shell 程序之處），以及 /var/adm 目錄（包含資料、報及摘要檔案）。

統計資料檔屬於管理群組的成員，且所有作用中資料檔（如 wtmp 及 pacct）都位在管理起始目錄 /var/adm 中。

統計資料檔
下列檔案位於 /var/adm 目錄中。

項目	說明
/var/adm/diskdiag	執行磁碟統計程式期間的診斷輸出
/var/adm/dtmp	**acctdusg** 指令的輸出
/var/adm/fee	**chargefee** 指令的輸出，為 ASCII tacct 記錄
/var/adm/pacct	作用中處理程序統計檔
/var/adm/wtmp	作用中處理程序統計檔
/var/adm/Spacct.*mmdd*	執行 **runacct** 指令期間，*mmdd* 的處理程序統計作業檔案。

統計報告及摘要檔案
啟用「統計作業」系統之前，需要一些子目錄。

報告及摘要檔案位於 /var/adm/acct 子目錄中。在啟用「統計作業」系統之前，您必須先建立下列子目錄。

/var/adm/acct/nite(x)
包含 **runacct** 指令每日重覆使用的檔案

/var/adm/acct/sum(x)
包含 **runacct** 指令每日更新的累計摘要檔案

/var/adm/acct/fiscal(x)
包含 **monacct** 指令建立的每月摘要檔案。

相關工作

設定統計作業系統
可以設定統計作業系統。

啟動 *runacct* 指令進行統計
可以啟動 **runacct** 指令。

先決要件

1. 您必須已安裝好統計作業系統。
2. 您必須具有 root 使用者或 adm 群組權限。

附註：

1. 如果您呼叫 **runacct** 指令而沒有使用任何參數，則指令會假設這是指令今天第一次執行。因此，當重新啟動 **runacct** 程式時，必須併入 *mmdd* 參數，以取得正確的月份與日期。如果您未指定狀態，**runacct** 程式會讀取 /var/adm/acct/nite(x)/statefile 檔案，來判斷處理程序的進入點。若要置換 /var/adm/acct/nite(x)/statefile 檔案，請在指令行指定所需的狀態。
2. 當執行下列作業時，您可能必須使用完整路徑名稱 /usr/sbin/acct/runacct，而不只是指令名稱 **runacct**。

程序

若要啟動 **runacct** 指令，請鍵入下列：

```
nohup runacct 2> \
/var/adm/acct/nite/accterr &
```

此項目會造成指令在執行背景處理程序時， 忽略所有 **INTR** 及 **QUIT** 信號。它會將所有標準錯誤輸出重新導向至 /var/adm/acct/nite/accterr 檔案。

重新啟動 *runacct* 指令進行統計
如果 **runacct** 指令未順利完成，您可以重新加以啟動。

此程序的必要條件為：

· 您必須已安裝好統計作業系統。

· 您必須具有 root 使用者或 adm 群組權限。

註： 最常見的 **runacct** 指令失敗理由是：

· 系統當機。

· /usr 檔案系統空間不足。

· /var/adm/wtmp 檔案記錄的日期戳記不一致。

如果 **runacct** 指令未順利完成，請執行下列動作：

1. 檢查 /var/adm/acct/nite(x)/active *mmdd* 檔案，以取得錯誤訊息。
2. 如果 acct/nite 中同時有作用中的檔案和鎖定檔案存在，則檢查 accterr 檔案， 當 **cron** 常駐程式呼叫 **runacct** 指令時，會在這個檔案中重新指出錯誤訊息。
3. 執行排除錯誤所需的任何動作。

4. 重新啟動 **runacct** 指令。
5. 若要重新啟動特定日期的 **runacct** 指令，請鍵入下列：

```
nohup runacct 0601 2>> \
/var/adm/acct/nite/accterr &
```

這會針對 6 月 1 日 (0601) 重新啟動 **runacct** 程式。**runacct** 程式會讀取 /var/adm/acct/nite/statefile 檔案， 找出開始的狀態。所有標準錯誤輸出均會附加到 /var/adm/acct/nite/accterr 檔案中。

6. 若要在指定的狀態重新啟動 **runacct** 程式， 例如，MERGE 狀態，請鍵入下列：

```
nohup runacct 0601 MERGE 2>> \
/var/adm/acct/nite/accterr &
```

runacct 指令檔

runacct 指令會產生報告及摘要檔案。

下列報告及摘要檔案（由 **runacct** 指令產生）是特別關心的檔案：

項目	說明
/var/adm/acct/nite(x)/lineuse	包含系統上每個終端機線路的使用情形統計資料。此報告對於偵測錯誤線路尤其有用。如果登出數與登入數之間的比例超過 3 比 1，則很可能有線路失敗。
/var/adm/acct/nite(x)/daytacct	包含前一天的總統計檔。
/var/adm/acct/sum(x)/tacct	包含每日的 nite/daytacct 檔案彙集，可用於記帳。**monacct** 指令會在每月或每個會計期間重新啟動該檔案。
/var/adm/acct/sum(x)/cms	包含每日指令摘要的彙集。**monacct** 指令會讀取此檔案的二進位版本並清除之。ASCII 版本為 nite/cms。
/var/adm/acct/sum(x)/daycms	包含每日指令摘要。ASCII 版本儲存在 nite/daycms 中。
/var/adm/acct/sum(x)/loginlog	包含上次使用每個使用者 ID 的記錄。
/var/adm/acct/sum(x)/rprt *mmdd*	此檔案包含 **runacct** 指令儲存之每日報告的副本。

/var/adm/acct/nite(x) 目錄中的檔案

下列檔案位於 /var/adm/acct/nite(x) 目錄中。

項目	說明
active	由 **runacct** 指令用來記錄進度及列印警告和錯誤訊息。檔案 active.*mmdd* 是 **runacct** 程式在偵測到錯誤後，所建立之 active 檔案的副本。
cms	**prdaily** 指令使用的 ASCII 總計指令摘要。
ctacct.*mmdd*	連接總統計記錄。
ctmp	連接階段作業記錄。
daycms	**prdaily** 指令使用的 ASCII 每日指令摘要。
daytacct	一天的總統計記錄。
dacct	**dodisk** 指令建立的磁碟總統計記錄。
accterr	執行 **runacct** 指令期間產生的診斷輸出。

項目	說明
lastdate	上次執行 **runacct** 的日期，格式為 *date+%m%d*。
lock1	用來控制 **runacct** 指令的序列使用。
lineuse	**prdaily** 指令使用的 tty 線路使用情形報告。
log	**acctcon1** 指令的診斷輸出。
log*mmdd*	與 **runacct** 指令偵測到錯誤後的 **log** 相同。
reboots	包含 wtmp 的開始及結束日期，以及系統重新啟動的報表。
statefile	用來在執行 **runacct** 指令期間記錄現行狀態。
tmpwtmp	**wtmpfix** 指令更正的 wtmp 檔案。
wtmperror	包含 **wtmpfix** 錯誤訊息。
wtmperr*mmdd*	與 runacct 指令偵測到錯誤後的 **wtmperror** 相同。
wtmp.*mmdd*	包含前一天的 wtmp 檔案。在 **runacct** 指令清除期間移除。

/var/adm/acct/sum(x) 目錄中的檔案

下列檔案位於 /var/adm/acct/sum(x) 目錄中。

項目	說明
cms	現行會計期間的總計指令摘要檔案（二進位格式）。
cmsprev	沒有最新更新的指令摘要檔案。
daycms	前一天的指令摘要檔案（二進位格式）。
lastlogin	**lastlogin** 指令建立的檔案。
pacct.*mmdd*	*mmdd* 的所有 pacct 檔案的連結版本。在系統啟動後，**remove** 指令會移除此檔案。如需 **remove** 指令的相關資訊，請參閱 **remove**。
rprt*mmdd*	**prdaily** 指令的儲存輸出。
tacct	現行會計期間的累計總統計檔。
tacctprev	與沒有最新更新的 tacct 相同。
tacct*mmdd*	*mmdd* 的總統計檔。

/var/adm/acct/fiscal(x) 目錄中的檔案

下列檔案位於 /var/adm/acct/fiscal(x) 目錄中。

項目	說明
cms*?*	*?* 指定之會計期間的總計指令摘要檔案（二進位格式）
fiscrpt*?*	與 **prdaily** 指令之報告相似的報告，為二進位格式，會計期間由 *?* 指定
tacct*?*	*?* 指定之會計期間的總統計檔（二進位格式）。

統計檔格式

下表彙總統計檔輸出及格式。

項目	說明
wtmp	產生作用中處理程序統計檔。wtmp 檔案的格式定義於 utmp.h 檔案。如需 utmp.h 檔案的相關資訊，請參閱 utmp.h。
ctmp	產生連接階段作業記錄。格式說明於 ctmp.h 檔案中。

項目	說明
pacct*	產生作用中處理程序統計記錄。輸出的格式定義於 /usr/include/sys/acct.h 檔案。
Spacct*	執行 **runacct** 指令期間會產生 *mmdd* 的處理程序統計作業檔案。這些檔案的格式定義於 sys/acct.h 檔案。
daytacct	產生一天的總統計記錄。檔案的格式定義於 tacct 檔案格式。
sum/tacct	產生累計每日指令摘要的二進位檔。此檔案的格式定義於 /usr/include/sys/acct.h 標頭檔。
ptacct	產生 pacct 檔案的連接版本。這些檔案的格式定義於 tacct 檔案。
ctacct	產生連接總統計記錄。此檔案的輸出定義於 tacct 檔案。
cms	產生 **prdaily** 指令使用的總統計指令摘要（二進位格式）。ASCII 版本為 nite/cms。
daycms	**prdaily** 指令使用的每日指令摘要（二進位格式）。ASCII 版本為 nite/daycms。

管理系統統計作業

有多個您可以針對系統統計作業完成的作業。這些作業包括設定統計作業系統、顯示 CPU 使用率，以及顯示統計作業處理程序。

設定統計作業系統

可以設定統計作業系統。

您必須具有 root 權限才能完成此程序。

以下資訊是設定統計作業系統時，必須採取的步驟概觀。請參照這些步驟中指示的指令與檔案，以取得其餘特定資訊。

1. 使用 **nulladm** 指令以確定每一個檔案確實具有正確的存取權：檔案擁有者及群組具有讀取 (r) 及寫入 (w) 許可權，其他人具有讀取 (r) 許可權，請鍵入：

   ```
   /usr/sbin/acct/nulladm wtmp pacct
   ```

   這會提供對 pacct 及 wtmp 檔案的存取權。
2. 更新 /etc/acct/holidays 檔案，以併入您指定為黃金時間的時數，並反映年度的假期排程。

   註： 註解行可以出現在檔案中的任何位置，只要行的第一個字元是星號 (*) 即可。

   a. 若要定義黃金時間，請使用 24 小時表示法，填寫第一資料行（不是註解的第一行）上的欄位。此行是由三個 4 位數欄位組成，次序如下：

      1) 今年
      2) 黃金時間開始 (*hhmm*)
      3) 黃金時間結束 (*hhmm*)

      前導的空白會被忽略。您可以將午夜輸入為 0000 或 2400。

      例如，若要指定 2000 年，主要時間開始於早上八點並結束於下午五點，請輸入：

      ```
      2000  0800  1700
      ```

   b. 在下一資料行定義年度的公司假期。每一行含有四個欄位，次序如下：

      1) 一年的第幾天
      2) 月份
      3) 一個月的第幾天
      4) 假期說明

      「一年的第幾天」欄位含有放假的日數，且必須是 1 到 365（若為閏年，則為 366）間的數字。例如，2 月 1 日是 32 天。其他三個欄位僅供參考，且被視為註解。

下面是兩行範例：

```
  1  Jan  1  New Year's Day
332  Nov 28  Thanksgiving Day
```

3. 將下行新增至 /etc/rc 檔案，或刪除該行前端的註解符號 "#"（如果存在），以啟用處理程序統計作業：

```
/usr/bin/su - adm -c /usr/sbin/acct/startup
```

啟動程序會記錄統計作業啟用的時間，並清除前一天的統計檔。

4. 在 /etc/filesystems 檔案中，將下行新增至檔案系統的段落，以指定您要併入磁碟統計作業中的每一個檔案系統：

```
account = true
```

5. 在 /etc/qconfig 檔案中，將下行新增至佇列段落，以指定用於印表機資料的資料檔：

```
acctfile = /var/adm/qacct
```

6. 身為 adm 使用者的您，請建立 /var/adm/acct/nite、/var/adm/acct/fiscal 及 /var/adm/acct/sum 目錄，以收集每日及會計期間記錄：

```
su - adm
cd /var/adm/acct
mkdir nite fiscal sum
exit
```

若是長使用者名稱，請改用下列的指令：

```
su - adm
cd /var/adm/acct
mkdir nitex fiscalx sumx
exit
```

7. 編輯 /var/spool/cron/crontabs/adm 檔案，在其中包括 **dodisk**、**ckpacct** 及 **runacct** 指令，以將每日統計作業程序設定為自動執行。
例如：

```
0 2 * * 4 /usr/sbin/acct/dodisk
5 * * * * /usr/sbin/acct/ckpacct
0 4 * * 1-6 /usr/sbin/acct/runacct
              2>/var/adm/acct/nite/accterr
```

若是長使用者名稱，請加入下列幾行：

```
0 2 * * 4 /usr/sbin/acct/dodisk -X
5 * * * * /usr/sbin/acct/ckpacct
0 4 * * 1-6 /usr/sbin/acct/runacct -X
              2>/var/adm/acct/nitex/accterr
```

第一行會在每星期四 (4) 的早上二點 (0 2) 啟動磁碟統計作業。 第二行會在每天 (*) 的每小時 5 分 (5 *) 時，開始檢查作用中資料檔的完整性。第三行會在每星期一到星期六 (1-6) 的早上四點 (0 4)，執行大部分的統計作業程序，並處理作用中的資料檔。如果這些時間不符合系統作業的時間，請調整您的項目。

註：您必須具有 root 使用者權限，才能編輯 /var/spool/cron/crontabs/adm 檔案。

8. 在 /var/spool/cron/crontabs/adm 檔案中併入 **monacct** 指令，以將每月統計作業摘要設定為自動執行。
例如，請鍵入：

```
15 5 1 * * /usr/sbin/acct/monacct
```

若是長使用者名稱，請加入下列幾行：

```
15 5 1 * * /usr/sbin/acct/monacct -X
```

請務必盡早排程此程序以完成報告。此範例會在每個月第一天的早上 5:15 啟動程序。

9. 若要提交已編輯的 cron 檔案，請鍵入：

```
crontab /var/spool/cron/crontabs/adm
```

相關概念

系統資料收集及報告
您可以設定系統，使其自動收集資料並產生報告。

統計作業系統活動報告
可以產生一份顯示「統計作業」系統活動的報告。

統計報告摘要
您可以產生一份彙總原始統計資料的報告。

相關工作

顯示作用中統計作業處理程序的處理時間
您可以顯示作用中處理程序的處理時間。

顯示已完成之統計作業處理程序的處理時間
您可以顯示已完成之處理程序的處理時間。

顯示每一個統計作業處理程序的 CPU 使用情況
您可以使用 **acctprc1** 指令，按使用者顯示有關 CPU 使用情況的格式化報告。

顯示每一個使用者的 CPU 統計使用情況
您可以使用 **acctprc1** 與 **prtacct** 指令的組合，按使用者顯示有關 CPU 使用情況的格式化報告。

顯示印表機或繪圖機使用情況統計記錄
您可以使用 **pac** 指令來顯示印表機或繪圖機使用情況統計記錄。

相關參考

統計報告及摘要檔案
啟用「統計作業」系統之前，需要一些子目錄。

顯示統計作業系統活動

您可以使用 **sar** 指令，顯示有關系統活動的格式化資訊。

若要顯示系統活動統計資料，則必須執行 **sadc** 指令。

註： 執行 **sadc** 指令的一般方法，是在 root crontab 檔案中放入 **sa1** 指令的項目。**sa1** 指令是設計來與 **cron** 常駐程式一起使用的 **sadc** 指令之 shell 程序變式。

若要顯示基本系統活動資訊，請鍵入：

```
sar 2 6
```

其中第一個數字是取樣間隔之間的秒數，第二個數字是要顯示的間隔數。此指令的輸出與下面類似：

```
        arthurd 2 3 000166021000    05/28/92
```

```
14:03:40    %usr    %sys    %wio   %idle
14:03:42       4       9       0      88
14:03:43       1      10       0      89
14:03:44       1      11       0      88
14:03:45       1      11       0      88
14:03:46       3       9       0      88
14:03:47       2      10       0      88
```

```
Average        2      10       0      88
```

sar 指令也提供許多旗標，可以顯示系統統計資料的擴充陣列。若要察看所有可用的統計資料，請使用 **-A** 旗標。如需可用統計資料與顯示資料之旗標的清單，請參閱 **sar** 指令。

註 : 若要將每日系統活動報告寫入 /var/adm/sa/sa*dd*，請在 root crontab 檔案中加入 **sa2** 指令的項目。**sa2** 指令是 **sar** 指令的 shell 程序變式， 其設計目的是要與 **cron** 常駐程式一起使用。

執行指令時顯示統計作業系統活動

您可以在特定指令執行時，顯示有關系統活動的格式化資訊。

timex 指令的 **-o** 與 **-p** 旗標要求必須啟動系統統計作業。

您可以使用 **time** 與 **timex** 指令以在執行特定指令時， 顯示有關系統活動的格式化資訊。

若要顯示特定指令的經歷時間、使用者時間與系統執行時期，請鍵入：

```
time CommandName
```

或

```
timex CommandName
```

若要顯示執行特定指令期間的總系統活動 (**sar** 指令所報告的所有資料項目），請鍵入：

```
timex -s CommandName
```

timex 指令有兩個其他旗標。**-o** 旗標會報告由該指令及所有子項所讀取或寫入的區塊總數。**-p** 旗標會列出指令及其所有子項的所有處理程序統計記錄。

顯示作用中統計作業處理程序的處理時間

您可以顯示作用中處理程序的處理時間。

acctcom 指令會以統計記錄總計格式 (acct 檔案格式）讀取輸入。這意指您已啟用處理程序統計作業，或曾經執行過處理程序統計作業。

ps 指令提供許多旗標以裁製顯示的資訊。

若要產生所有作用中處理程序（核心處理程序除外）的完整清單，請鍵入：

```
ps -ef
```

您也可以顯示與終端機相關聯之所有處理程序的清單。若要執行此動作，請鍵入：

```
ps -al
```

這兩種用法均會顯示每一個處理程序的大量直欄，包括處理程序的現行 CPU 時間，以分鐘及秒鐘為單位。

相關工作

設定統計作業系統
可以設定統計作業系統。

顯示已完成之統計作業處理程序的處理時間

您可以顯示已完成之處理程序的處理時間。

acctcom 指令會以統計記錄總計格式 (acct 檔案格式）讀取輸入。這意指您已啟用處理程序統計作業，或曾經執行過處理程序統計作業。

您可以使用 **startup** 指令啟用處理程序統計作業功能，它通常是在系統起始設定時，利用 /etc/rc 檔案中的呼叫來啟動。當處理程序統計作業的功能正在執行時，每一個完成的處理程序的記錄會寫入 /var/adm/pacct（統計記錄總計檔案）， 包括處理程序的開始與停止時間。您可以使用 **acctcom** 指令，來顯示 pacct 檔案中的處理時間資訊。此指令有許多旗標，可以彈性地指定要顯示的處理程序。

例如，若要察看執行最小 CPU 秒數或更長時間的所有處理程序，請使用 **-O** 旗標，請鍵入：

```
acctcom -O 2
```

這會顯示每一個至少執行 2 秒鐘的處理程序記錄。如果您沒有指定輸入檔，則 **acctcom** 指令會讀取 /var/adm/pacct 目錄中的輸入。

相關工作
設定統計作業系統
可以設定統計作業系統。

顯示每一個統計作業處理程序的 *CPU* 使用情況

您可以使用 **acctprc1** 指令，按使用者顯示有關 CPU 使用情況的格式化報告。

acctprc1 指令要求輸入必須採取統計記錄總計格式（acct 檔案格式）。這意指您已啟用處理程序統計作業，或曾經執行過處理程序統計作業。

若要按處理程序產生 CPU 使用情況的格式化報告，請鍵入：

```
acctprc1 </var/adm/pacct
```

相關工作
設定統計作業系統
可以設定統計作業系統。

顯示每一個使用者的 *CPU* 統計使用情況

您可以使用 **acctprc1** 與 **prtacct** 指令的組合，按使用者顯示有關 CPU 使用情況的格式化報告。

acctprc1、acctprc2 或 accton 指令要求輸入必須採取統計記錄總計格式（acct 檔案格式）。這意指您已啟用處理程序統計作業，或曾經執行過處理程序統計作業。

若要顯示每一位使用者的 CPU 使用情況，請執行下列步驟：

1. 若要按處理程序產生 CPU 使用情況的輸出檔，請鍵入：

   ```
   acctprc1 </var/adm/pacct >out.file
   ```

   /var/adm/pacct 檔案是處理程序統計作業記錄的預設輸出。您可能想要指定保存的 pacct 檔案來代替。
2. 若要從前一步驟的輸出中，產生二進位的總統計記錄檔，請鍵入：

   ```
   acctprc2 <out.file >/var/adm/acct/nite/daytacct
   ```

   註：acctmerg 指令會使 daytacct 檔案與其他總統計記錄合併，以產生每日摘要記錄 /var/adm/acct/sum(x)/tacct。
3. 若要使用顯示每一個使用者的 CPU 統計使用情況指令，來顯示按使用者彙總之 CPU 使用率的格式化報告，請鍵入：

   ```
   prtacct </var/adm/acct/nite/daytacct
   ```

相關工作
設定統計作業系統
可以設定統計作業系統。

顯示統計作業的連線時間用量

您可以使用 **ac** 指令，顯示所有使用者、個別使用者及個別登入的連線時間。

ac 指令會從 /var/adm/wtmp 檔案中擷取登入資訊，所以此檔案必須存在。如果檔案尚未建立，則會傳回下列錯誤訊息：

```
No /var/adm/wtmp
```

如果檔案已滿，則會建立別的 wtmp 檔案；您可以使用 **-w** 旗標來指定這些檔案，以顯示其中的連線時間資訊。如需 **ac** 指令的相關資訊，請參閱 **ac**。

若要顯示所有使用者的總連接時間，請鍵入：

```
/usr/sbin/acct/ac
```

此指令會顯示一個十進位數， 代表在現行 wtmp 檔案的有效期間內所有登入使用者的連接時間總計（以分鐘為單位）。

若要顯示一或多個特定使用者的總連接時間，請鍵入：

```
/usr/sbin/acct/ac User1 User2 ...
```

此指令會顯示一個單一十進位數，代表在現行 wtmp 檔案的有效期間內，您所指定之一或多個登入使用者的總連接時間合計（以分鐘為單位）。

若要顯示個別使用者的連接時間加上總連接時間，請鍵入：

```
/usr/sbin/acct/ac -p User1 User2 ...
```

此指令會顯示成一個十進位數，代表在現行 wtmp 檔案有效期間內指定使用者的總連線時間（分鐘）。它也會顯示一個十進位數，代表所有指定的使用者的連接時間總和。如果指令中沒有指定任何使用者，則清單會併入在 wtmp 檔案的有效期間內登入的所有使用者。

顯示統計作業的磁碟空間使用率
您可以使用 **acctmerg** 指令，顯示磁碟空間使用量資訊。

若要顯示磁碟空間的使用量資訊，**acctmerg** 指令需要 dacct 檔案（磁碟統計）的輸入。磁碟使用情形統計記錄的收集是由 **dodisk** 指令執行。

若要顯示磁碟空間使用量資訊，請鍵入：

```
acctmerg -a1 -2,13 -h </var/adm/acct/nite(x)/dacct
```

此指令會顯示磁碟統計記錄，其中包括每一個使用者使用 1 KB 區塊數。

註：acctmerg 指令一律從標準輸入來讀取，且最多可以讀取九個額外的檔案。如果您沒有傳送輸入至指令，則必須重新導向某一檔案的輸入；您可以指定其餘的檔案而不用重新導向。

顯示印表機或繪圖機使用情況統計記錄
您可以使用 **pac** 指令來顯示印表機或繪圖機使用情況統計記錄。

- 若要收集印表機使用情況的資訊，您必須設定並執行統計作業系統。請參閱第 142 頁的『設定統計作業系統』以取得指引。
- 您想要取得其統計記錄的印表機或繪圖機，必須在 /etc/qconfig 檔案的印表機段落中具有 acctfile= 子句。acctfile= 子句中所指定的檔案必須將讀取與寫入許可權授與 root 使用者或 printq 群組。
- 如果在 **pac** 指令指定 **-s** 旗標，則指令會附加 **_sum** 到 /etc/qconfig 檔案中 acctfile= 子句所指定的路徑名稱，以重寫摘要檔案名稱。此檔案必須存在，並將讀取與寫入許可權授與 root 使用者或 printq 群組。

若要為特定印表機的所有使用者顯示印表機使用情況資訊，請鍵入：

```
/usr/sbin/pac -PPrinter
```

如果您沒有指定印表機，則預設印表機是由 PRINTER 環境變數指名。如果未定義 PRINTER 變數，則預設值是 lp0。

若要為特定印表機之特定使用者顯示印表機使用情況資訊，請鍵入：

```
/usr/sbin/pac -PPrinter User1 User2 ...
```

pac 指令提供其他旗標，以控制顯示的資訊。

相關工作
設定統計作業系統
可以設定統計作業系統。

更新假日檔案
在過了最後一個列出的假日，或年份已變更時，假日檔案即會過期。您可以更新假日檔案。

當 /usr/lib/acct/holidays 檔案過期時，**acctcon1** 指令（從 **runacct** 指令啟動）會傳送郵件給 **root** 及 **adm** 帳戶。

若要更新過期的假日檔案，可以編輯 /var/adm/acct/holidays 檔案，以區別黃金時間與非黃金時間。

主要時間是假設系統大部分都在作用中的時期，如工作日。星期六與星期日一定是統計作業系統的非主要時間，如同您所列出的假日。

假日檔案含有三種類型的項目： 註解、年份與主要時間時期，以及假日清單，如下面範例所示：

```
* Prime/Non-Prime Time Table for Accounting System
*
*   Curr       Prime             Non-Prime
*   Year       Start             Start
        1992       0830              1700
*
*   Day of     Calendar          Company
*   Year       Date              Holiday
*
*   1          Jan 1             New Year's Day
*   20         Jan 20            Martin Luther King Day
*   46         Feb 15            President's Day
*   143        May 28            Memorial Day
*   186        Jul 3             4th of July
*   248        Sep 7             Labor Day
*   329        Nov 24            Thanksgiving
*   330        Nov 25            Friday after
*   359        Dec 24            Christmas Eve
*   360        Dec 25            Christmas Day
*   361        Dec 26            Day after Christmas
```

第一個非註解行必須指定目前年份（以四位數表示），以及主要時間的開始與結束（各以四位數表示）。主要及非主要時間的概念只會影響統計作業程式處理統計記錄的方式。

如果假日的清單太長，則 **acctcon1** 指令會產生錯誤，且您將必須縮短清單。假日數在 20 以下是安全的。如果您想要新增更多的假日， 只要每個月編輯假日檔即可。

收集統計資料
一旦設定系統統計作業之後，您會想要開始收集及處理不同類型的統計資料。

系統資料收集及報告
您可以設定系統，使其自動收集資料並產生報告。

若要自動收集資料，adm 群組的成員必須已設定為統計作業系統。統計作業系統設定可讓 **cron** 常駐程式執行會產生下列項目之相關資料的指令：

- 每個使用者登入系統所花費的時間量
- 處理裝置、記憶體及 I/O 資源的使用情形
- 每個使用者檔案所佔據的磁碟空間量
- 印表機及繪圖機的使用情形
- 提供特定指令的次數。

系統會在每個階段作業及處理程序完成後寫入它們的記錄。這些記錄會轉換成使用者整理的總統計 (tacct) 記錄，並合併到每日報告中。會定期合併每日報告，以針對定義的會計期間產生總計。在接下來的各節中會討論收集及報告資料的方法，以及各種統計指令及檔案。

雖然大部分統計資料是自動進行收集及處理的，但管理群組的成員可以從鍵盤輸入某些特定指令來取得特定的資訊。

相關工作
設定統計作業系統
可以設定統計作業系統。

相關參考
鍵盤指令
管理群組的成員可以從鍵盤輸入下列指令。

連線時間統計資料
連線時間資料是由 **init** 指令及 **login** 指令收集的。

當您登入時，**login** 程式就會在 /etc/utmp 檔案中寫入一筆記錄。此記錄會包括您的使用者名稱、登入的日期及時間，以及登入埠。指令（如 **who** ）會使用此檔案來找出登入各個顯示站的使用者。如果有 /var/adm/wtmp 連線時間統計檔存在，則 **login** 指令會將此登入記錄的副本新增到該檔案中。如需 **init** 及 **login** 指令的相關資訊，請參閱 **init** 及 **login**。

當您的登入程式結束時（通常是在您登出時），**init** 指令會在 /var/adm/wtmp 檔案中寫入另一筆記錄，來記錄階段作業已結束。登出記錄與登入記錄的不同之處在於它們具有空白的使用者名稱。登入及登出記錄的格式在 utmp.h 檔案中都有說明。如需 utmp.h 檔案的相關資訊，請參閱 utmp.h。

acctwtmp 指令還會在 /var/adm/wtmp 檔案中，寫入有關系統關機及啟動的特殊項目。

相關概念
連線時間報告
統計記錄包括登入、登出、系統關機及上次登入記錄。

處理程序統計作業資料
「統計作業」系統會在每個處理程序執行時，收集其資源使用情形的相關資料。

此資料包括：

- 執行處理程序所依據的使用者和群組號碼
- 指令名稱的前八個字元
- 代表處理程序所屬之「工作量管理程式」類別的 64 位元數字鍵
- 處理程序所用的經歷時間及處理器時間
- 記憶體使用
- 所轉送的字元數
- 代表處理程序讀取或寫入的磁碟區塊數

accton 指令會將這些資料記錄在指定的檔案（通常是 /var/adm/pacct 檔案）中。如需 **accton** 指令的相關資訊，請參閱 **accton**。

相關指令為 **startup** 指令、**shutacct** 指令、**dodisk** 指令、**ckpacct** 指令及 **turnacct** 指令。如需這些指令的相關資訊，請參閱 **startup**、**shutacct**、**dodisk**、**ckpacct** 及 **turnacct**。

相關概念
統計資料報告
在收集完各種不同類型的統計資料之後，會處理記錄並將記錄轉換成報告。

處理程序統計作業報告
兩個指令會處理在 /var/adm/pacct 或其他指定檔案中收集的帳目相關資料。

acctprc1 指令會將使用者 ID 轉換成使用者名稱，並寫入包含可收費項目（主要及非主要 CPU 時間、平均記憶體大小及 I/O 資料）的 ASCII 記錄。**acctprc2** 指令會將這些記錄轉換成總統計記錄，而 **acctmerg** 指令會將這些總統計記錄新增到每日報告。如需 **acctmerg** 指令的相關資訊，請參閱 **acctmerg**。

處理程序統計資料還會提供您可用來監視系統資源使用情形的資訊。**acctcms** 指令會依指令名稱彙總資源使用情形。這會提供每個指令執行次數、所用處理器時間及記憶體量，以及資源的使用程度（又稱 *hog* 因數）的相關資訊。**acctcms** 指令會產生系統使用率的長期統計資料，提供總計系統使用情形及指令使用頻率的相關資訊。如需 **acctcms** 指令的相關資訊，請參閱 **acctcms**。

acctcom 指令處理的資料與 **acctcms** 指令處理的資料相同，但提供每個處理程序的詳細資訊。您可以顯示所有處理程序統計記錄，或選取特別關心的記錄。選取準則包括處理程序強制的負荷、處理程序結束的時間期間、指令的名稱、呼叫處理程序的使用者或群組、處理程序所屬之 WLM 類別的名稱，以及處理程序執行所在的埠。與其他統計指令不同，**acctcom** 可由所有使用者執行。如需 **acctcom** 指令的相關資訊，請參閱 **acctcom**。

磁碟使用情形統計作業資料
許多統計資訊都是在使用資源時進行收集的。**dodisk** 指令（由 **cron** 常駐程式指定執行），會定期將每個使用者的磁碟使用情形記錄寫入 /var/adm/acct/nite(x)/dacct 檔案。

為完成此作業，**dodisk** 指令會呼叫其他指令。根據統計作業搜尋的完全程度而定，可使用 **diskusg** 指令或 **acctdusg** 指令來收集資料。**acctdisk** 指令用來寫入總統計記錄。總統計記錄又依次由 **acctmerg** 指令使用來準備每日統計報告。

dodisk 指令會針對在使用者登入目錄中找到之檔案的鏈結向使用者收費，並將每個檔案的費用在鏈結之間平均分攤。這會將使用檔案的花費分散給所有使用該檔案的使用者，並在使用者放棄對檔案的存取時將費用從使用者身上移除。如需 **dodisk** 指令及 **cron** 指令的相關資訊，請參閱 **dodisk** 及 **cron**。

相關概念
連線時間報告
統計記錄包括登入、登出、系統關機及上次登入記錄。

印表機使用情形統計資料
印表機使用情形資料的收集需要 **enq** 指令與佇列常駐程式協力完成。

enq 指令會將要列印之檔案的名稱、使用者名稱及工作號碼移入佇列。在列印檔案之後，**qdaemon** 指令會將 ASCII 記錄寫入檔案（通常是 /var/adm/qacct 檔案），包含使用者名稱、使用者號碼及列印的頁數。您可以將這些記錄排序，並將它們轉換成總統計記錄。如需這些指令的相關資訊，請參閱 **enq** 及 **qdaemon**。

相關概念
印表機使用情況統計報告
/var/adm/qacct 檔案中的 ASCII 記錄可以轉換成總統計記錄，以由 **acctmerg** 指令新增到每日報告中。

費用統計資料
您可以在 /var/adm/fee 檔案中產生 ASCII 總統計記錄。

您可以輸入 **chargefee** 指令來在 /var/adm/fee 檔案中產生 ASCII 總統計記錄。此檔案將由 **acctmerg** 指令新增至每日報告中。

如需 **chargefee** 及 **acctmerg** 指令的相關資訊，請參閱 **chargefee** 及 **acctmerg**。

相關概念
費用統計報告
如果您是使用 **chargefee** 指令向使用者收取諸如檔案還原、諮詢或素材等服務費用，則會在 /var/adm/fee 檔案中寫入 ASCII 統計記錄總計。此檔案由 **acctmerg** 指令新增至每日報告中。

疑難排解系統統計作業
使用這些疑難排解方法，可以解決使用系統統計作業時可能發生的部分基本問題。如果疑難排解資訊未解決您的問題，請聯絡客戶服務代表。

修正 *tacct* 錯誤
如果您是使用統計作業系統來向使用者收取系統資源使用費，則 /var/adm/acct/sum/tacct 檔案的完整性是相當重要的。偶爾，會出現奇怪的 **tacct** 記錄，其中含有負數、重複的使用者號碼或使用者號碼 65,535。您可以修正這些問題。

您必須具有 root 使用者或 adm 群組權限。

若要修補 tacct 檔案，請執行下列步驟：

1. 請鍵入下列指令以移至 /var/adm/acct/sum 目錄：
   ```
   cd /var/adm/acct/sum
   ```
2. 使用 **prtacct** 指令以檢查總統計檔 tacctprev，請鍵入：
   ```
   prtacct tacctprev
   ```

prtacct 指令會格式化並顯示 tacctprev 檔案，使您可以檢查連線時間、處理時間、磁碟使用情況及印表機使用情況。

3. 如果 tacctprev 檔案正確，請將最新的 tacct .*mmdd* 檔從二進位檔變更為 ASCII 檔。在下面範例中，**acctmerg** 指令會將 tacct.*mmdd* 檔轉換成 ASCII 檔 tacct.new：

```
acctmerg -v < tacct.mmdd > tacct.new
```

註： 在 **acctmerg** 指令中使用 **-a** 旗標，也會產生 ASCII 輸出。**-v** 旗標會產生更精確的浮點數字表示。

acctmerg 指令是用來將中間帳戶記錄報告合併到累加總計報告 (**tacct**)。此累加總計是 **monacct** 指令產生 ASCII 每月摘要報告的來源。因為 **monacct** 指令程序會移除所有 tacct.*mmdd* 檔案，所以您可以合併這些檔案來重建 tacct 檔案。

4. 編輯 tacct.new 檔案以移除錯誤記錄，並將重複的使用者號碼記錄寫入另一個檔案，請鍵入：

```
acctmerg -i < tacct.new > tacct.mmdd
```

5. 請鍵入下列指令以重建 tacct 檔案：

```
acctmerg tacctprev < tacct.mmdd > tacct
```

修正 *wtmp* 錯誤

/var/adm/wtmp 或 "who temp" 檔案可能會在統計作業系統的日常作業中造成問題。您可以修正 wtmp 錯誤。

您必須具有 root 使用者或 adm 群組權限，才能執行這個程序。

當日期變更且系統處於多使用者模式中時，日期變更記錄會寫入 /var/adm/wtmp 檔案。發生日期變更時，**wtmpfix** 指令會調整 wtmp 記錄中的時間戳記。日期變更與系統重新啟動的部分組合可能會遺漏傳送 **wtmpfix** 指令，並造成 **acctcon1** 指令失敗，且 **runacct** 指令會傳送郵件至 **root** 與 **adm** 帳戶並列出不正確的日期。

若要修正 wtmp 錯誤，請執行下列程序：

1. 請鍵入下列指令以移至 /var/adm/acct/nite 目錄：

```
cd /var/adm/acct/nite
```

2. 將二進位 wtmp 檔案轉換成您可以編輯的 ASCII 檔，請鍵入：

```
fwtmp < wtmp.mmdd > wtmp.new
```

fwtmp 指令會將 wtmp 從二進位轉換成 ASCII。

3. 編輯 ASCII wtmp.new 檔案，從檔案的開頭刪除受損的記錄或全部記錄，一直到所需的日期變更記錄，請鍵入：

```
vi wtmp.new
```

4. 鍵入下列指令，將 ASCII wtmp.new 檔案轉換回二進位格式：

```
fwtmp -ic < wtmp.new > wtmp.mmdd
```

5. 如果 wtmp 檔案無法修復，請使用 **nulladm** 指令建立一個空的 wtmp 檔案。

   這可避免連線時的費用。

```
nulladm wtmp
```

nulladm 指令會建立檔案，並指定檔案擁有者及群組的讀取與寫入許可權，以及其他使用者的讀取許可權。它可確保檔案擁有者與群組是 **adm**。

相關工作

修正統計作業錯誤
您可以更正日期與時間戳記的不一致。

修正不正確的統計檔許可權

檔案的所有權與許可權都必須正確，才能使用「統計作業」系統。

您必須具有 root 使用者或 adm 群組權限，才能執行這個程序。

adm 管理帳戶擁有統計作業指令與 Script，但 /var/adm/acct/accton 除外，它的擁有者是 root。

若要修正不正確的「統計檔」許可權，請執行下列程序：

1. 若要使用 **ls** 指令檢查檔案許可權，請鍵入：

   ```
   ls -l /var/adm/acct

   -rws--x--- 1 adm adm 14628 Mar 19 08:11 /var/adm/acct/fiscal
   -rws--x--- 1 adm adm 14628 Mar 19 08:11 /var/adm/acct/nite
   -rws--x--- 1 adm adm 14628 Mar 19 08:11 /var/adm/acct/sum
   ```

2. 如果需要，請使用 **chown** 指令來調整檔案許可權。

   許可權是 755（擁有者具有所有許可權，並且所有其他使用者具有讀取及執行許可權）。同時，目錄本身應具有寫入保護以防止他人寫入。

   例如：

   a. 鍵入下列指令以移至 /var/adm/acct 目錄：

      ```
      cd /var/adm/acct
      ```

   b. 若要將 sum、nite 及 fiscal 目錄的所有權變更為 **adm** 群組權限，請鍵入：

      ```
      chown adm sum/* nite/* fiscal/*
      ```

      為了防止有使用者嘗試不付費而弄亂檔案，請拒絕其他使用者在這些檔案上的寫入許可權。將 **accton** 指令群組擁有者變更為 **adm**，並將許可權變更為 710，亦即，其他使用者沒有任何許可權。**adm** 擁有的處理程序可以執行 **accton** 指令，但一般使用者則不能。

3. /var/adm/wtmp 檔案也必須由 **adm** 擁有。如果 /var/adm/wtmp 由 root 擁有，則您會在啟動期間看到下列訊息：

   ```
   /var/adm/acct/startup: /var/adm/wtmp: Permission denied
   ```

   若要更正 /var/adm/wtmp 的所有權，請鍵入下列指令， 將所有權變更為 **adm** 群組：

   ```
   chown adm /var/adm/wtmp
   ```

修正統計作業錯誤

您可以更正日期與時間戳記的不一致。

您必須具有 root 使用者或 adm 群組權限，才能執行這個程序。

處理 /var/adm/wtmp 檔案可能會產生一些警告訊息，並寄發給 root。wtmp 檔案含有 /etc/init 及 /bin/login 所收集的資訊，且主要是由統計作業 Script 用來計算連接時間（使用者登入的時間長度）。不幸的是，日期變更會混淆處理 wtmp 檔案的程式。結果，**runacct** 指令會傳送郵件給 root 與 adm， 抱怨自前次執行統計作業後，因日期變更後所產生的所有錯誤。

1. 請確定您是否收到任何錯誤。

   acctcon1 指令會輸出錯誤訊息，並利用 **runacct** 指令將訊息郵寄給 adm 與 root。

   例如，如果 **acctcon1** 指令在日期變更後而產生錯誤且無法收集到連接時間，則 adm 可能會收到類似下面的郵件訊息：

   ```
   Mon Jan 6 11:58:40 CST 1992
   acctcon1: bad times: old: Tue Jan 7 00:57:14 1992
   new: Mon Jan 6 11:57:59 1992
   acctcon1: bad times: old: Tue Jan 7 00:57:14 1992
   new: Mon Jan 6 11:57:59 1992
   acctcon1: bad times: old: Tue Jan 7 00:57:14 1992
   new: Mon Jan 6 11:57:59 1992
   ```

2. 請鍵入下列指令以調整 wtmp 檔案：

```
/usr/sbin/acct/wtmpfix wtmp
```

wtmpfix 指令會檢查 wtmp 檔案是否有日期與時間戳記不一致，並更正會造成 **acctcon1** 失敗的問題。然而，**wtmpfix** 可能會漏掉部分日期變更。

3. 正好在關機之前或在啟動後立即執行統計作業。

在這些時候使用 **runacct** 指令，可將錯誤時間的項目數量降至最低。**runacct** 指令會持續傳送郵件給 root 與 adm 帳戶， 直到您編輯 **runacct** Script、找出 WTMPFIX 區段，並註銷檔案日誌郵寄給 root 及 adm 帳戶所在的那一行為止。

相關工作

修正 wtmp 錯誤

/var/adm/wtmp 或 "who temp" 檔案可能會在統計作業系統的日常作業中造成問題。您可以修正 wtmp 錯誤。

執行 *runacct* 指令時發生統計作業錯誤

執行 **runacct** 指令時，可能會發生錯誤。

註 : 您必須具有 root 使用者或 adm 群組權限，才能執行 **runacct** 指令。

runacct 指令處理通常是非常大型的檔案。程序包含傳送某些檔案，且在執行時會使用相當大量的系統資源。因為 **runacct** 指令很耗費資源，所以通常會在一大早執行， 這樣它就可以接管機器，但不會干擾任何人。

runacct 指令是一個分成不同階段的 Script。這些階段可讓您在指令停止的地方重新啟動，而不必重新執行整個 Script。

當 **runacct** 發生問題時，它會傳送錯誤訊息至不同的目的地，視錯誤發生的位置而定。通常，它會將日期與訊息傳送至主控台，以引導您查看 active*MMDD* 檔案（如 active0621 表示 6 月 21 日），該檔案儲存在 /usr/adm/acct/nite 目錄中。當 **runacct** 指令中斷時，它會將整個 active 檔案移至 active*MMDD*，並附加說明問題的訊息。

複查下列錯誤訊息表，以取得執行 **runacct** 指令時所發生的錯誤。

註 :

- *MMDD* 縮寫代表月份與日期，如 0102 代表 1 月 2 日。例如，在 1 月 2 日的 CONNECT1 處理程序期間發生無法復原的錯誤，即會建立內含錯誤訊息的檔案 active0102。
- "SE message" 縮寫代表標準錯誤訊息，如：

```
********* ACCT ERRORS : see active0102 *********
```

runnacct 指令的初步狀態與錯誤訊息				
狀態	**指令**	**無法復原嗎？**	**錯誤訊息**	**目的地**
pre	**runacct**	是	* 2 CRONS 或 ACCT PROBLEMS * 錯誤：發現鎖定，執行中斷	主控台、郵件、active
pre	**runacct**	是	runacct：/usr 中的空間不足（*nnn* blks）；終止程序	主控台、郵件、active

runnacct 指令的初步狀態與錯誤訊息 (繼續)				
狀態	**指令**	**無法復原嗎？**	**錯誤訊息**	**目的地**
pre	**runacct**	是	SE 訊息； 錯誤：已經執行 'date' 的 acctg：檢查 lastdate	主控台、郵件、active*MMDD*
pre	**runacct**	否	* 系統統計作業已啟動 *	主控台
pre	**runacct**	否	在 'date' 時，以 STATE 重新啟動 acctg	作用中的主控台、主控台
pre	**runacct**	否	在 'date' 時，以狀態 (引數 $2) 重新啟動 acctg，之前的狀態是 STATE	active
pre	**runacct**	是	SE 訊息；錯誤：呼叫 runacct 所使用的引數無效	主控台、郵件、active*MMDD*

runacct 指令的狀態與錯誤訊息				
狀態	**指令**	**無法復原嗎？**	**錯誤訊息**	**目的地**
SETUP	**runacct**	否	ls -l fee pacct* /var/adm/wtmp	active
SETUP	**runacct**	是	SE 訊息；錯誤：turnacct 切換參數傳回 rc=error	主控台、郵件、active*MMDD*
SETUP	**runacct**	是	SE 訊息；錯誤：SpacctMMDD 已存在，可能已執行檔案設定	active*MMDD*
SETUP	**runacct**	是	SE 訊息；錯誤：wtmpMMDD 已存在：以手動方式執行設定	主控台、郵件、active*MMDD*
WTMPFIX	**wtmpfix**	否	SE 訊息；錯誤：wtmpfix 錯誤，請參閱 xtmperrorMMDD	active*MMDD*、wtmperror*MMDD*
WTMPFIX	**wtmpfix**	否	wtmp 處理完成	active
CONNECT1	**acctcon1**	否	SE 訊息；（acctcon1 日誌中發生錯誤）	主控台、郵件、active*MMDD*
CONNECT2	**acctcon2**	否	連接 acctg 完成	active

runacct 指令的狀態與錯誤訊息 *(繼續)*				
狀態	**指令**	**無法復原嗎？**	**錯誤訊息**	**目的地**
PROCESS	**runacct**	否	警告：已執行 pacct *N* 的統計作業	active
PROCESS	**acctprc1 acctprc2**	否	SpacctNMMDD 的處理程序 acctg 完成	active
PROCESS	**runacct**	否	date 的所有處理程序 actg 完成	active
MERGE	**acctmerg**	否	tacct 合併以建立 dayacct 完成	active
FEES	**acctmerg**	否	合併費用或無費用	active
DISK	**acctmerg**	否	合併磁碟記錄或無磁碟記錄	active
MERGEACCT	**acctmerg**	否	警告：重建 sum/tacct	active
MERGEACCT	**acctmerg**	否	更新 sum/tacct	active
CMS	**runacct**	否	警告：重建 sum/cms	active
CMS	**acctcms**	否	指令摘要完成	active
CLEANUP	**runacct**	否	系統統計作業已於 'date' 完成	active
CLEANUP	**runacct**	否	*系統統計作業完成*	主控台
<wrong>	**runacct**	是	SE 訊息；錯誤：狀態無效，檢查 STATE	主控台、郵件、active*MMDD*

註：前一表格中的標籤 <wrong> 不代表狀態，而是寫入狀態檔 /usr/adm/acct/nite/statefile 中正確狀態以外的狀態。

訊息目的地摘要	
目的地	**說明**
主控台	/dev/console 裝置
郵件	傳送訊息郵件至 **root** 及 **adm** 帳戶
active	/usr/adm/acct/nite/active 檔案
activeMMDD	/usr/adm/acct/nite/active*MMDD* 檔案
wtmperrMMDD	/usr/adm/acct/nite/wtmperror*MMDD* 檔案
STATE	/usr/adm/acct/nite/statefile 檔案中的現行狀態
fd2log	任何其他錯誤訊息

系統資源控制器

「系統資源控制器 (SRC)」可提供指令及子常式集，以便系統管理員及程式設計師能夠更容易地建立及控制子系統。

子系統 是通常可以獨立操作或與控制系統一同操作的任何程式、處理程序或是程式或處理程序集。子系統設計為提供指定功能的一個裝置。

設計 SRC 的用意是要讓操作員介入的需要縮至最小。它提供使用常用指令行及 C 介面來控制子系統處理程序的機制。此機制包括下列內容：

· 啟動、停止及狀態查詢的一致使用者介面

· 記載子系統的異常終止

· 相關處理程序異常系統終止時呼叫的通知程式

· 追蹤子系統、子系統群組或子伺服器

· 支援控制遠端系統上的作業

· 重新整理子系統（如配置資料變更之後）。

如果您要使用常用的方法來啟動、停止及收集有關處理程序的狀態資訊，則 SRC 是很有用的。

相關概念

AIX for BSD 系統管理程式簡介

下列是協助 Berkeley Software Distribution (BSD) 系統管理者開始管理 AIX 的要訣。

子系統元件

下列是子系統的特性及元件。

子系統可能具有下列特性中的一或多個：

· 系統依名稱識別

· 需要比子常式或非特許程式更複雜的執行環境

· 包括應用程式及程式庫以及子系統程式碼

· 控制可依名稱啟動及停止的資源

· 若相關處理程序未順利執行清除或回復資源，則需要通知

· 需要比簡單常駐程式處理程序更多的作業控制

· 需要由遠端操作員控制

· 施行子伺服器以管理特定的資源

· 不將其本身置於背景中。

少數子系統範例為 ypserv、ntsd、qdaemon、inetd、syslogd 及 sendmail。

註 : 請參閱每個特定的子系統，以取得其 SRC 功能的進一步資訊。

使用 **lssrc -a** 指令來 列出系統上作用中及非作用中的子系統。

下列定義子系統群組及子伺服器：

子系統群組

子系統群組是所有指定子系統的群組。將子系統分組在一起即可允許同時控制數個子系統。少數子系統群組範例為 TCP/IP、「SNA 服務」、「網路資訊系統 (NIS)」及「網路檔案系統 (NFS)」。

子伺服器

子伺服器是屬於子系統的程式或處理程序。子系統可以有多個子伺服器，並負責啟動、停止子伺服器以及提供其狀態。僅可為具有 IPC 訊息佇列及 Socket 通訊類型的子系統定義子伺服器。使用信號通訊的子系統不支援子伺服器。

啟動母項子系統時，亦會啟動其子伺服器。若嘗試啟動子伺服器，而其母項子系統處於非作用中，則 **startsrc** 指令亦會啟動子系統。

SRC 階層

「系統資源控制器」階層是從作業系統開始，其後是子系統群組（如 **tcpip**），其中包含子系統（如 **inetd** 常駐程式），子系統中又可擁有數個子伺服器（如 **ftp** 常駐程式及 **finger** 指令）。

SRC 管理指令

您可以從指令行管理 SRC。

SRC 管理指令包括：

項目	說明
srcmstr daemon	啟動「系統資源控制器」
startsrc 指令	啟動子系統、子系統群組或子伺服器
stopsrc 指令	停止子系統、子系統群組或子伺服器
refresh 指令	重新整理子系統
traceson 指令	啟用子系統、子系統群組或子伺服器的追蹤
tracesoff 指令	停用子系統、子系統群組或子伺服器的追蹤
lssrc 指令	取得子系統的狀態。

啟動系統資源控制器

在系統起始設定期間，會啟動「系統資源控制器 (SRC)」，並在 /etc/inittab 檔案中寫入 /usr/sbin/srcmstr 常駐程式的記錄。

下列是啟動 SRC 的必要條件：

- 讀取及寫入 /etc/inittab 檔案需要 root 使用者權限。
- **mkitab** 指令需要 root 使用者權限。
- **srcmstr** 常駐程式記錄必須存在於 /etc/inittab 檔案中。

預設的 /etc/inittab 檔案已包含這類記錄，所以此程序可能不必要。您也可以從指令行、設定檔或 shell Script 啟動 SRC， 但在起始設定期間內，有數個理由要啟動它：

- 從 /etc/inittab 檔案啟動 SRC，如果它因任何理由而停止，可以讓 **init** 指令重新啟動 SRC。
- SRC 的設計是要簡化及降低控制子系統時需要操作員介入的次數。自 /etc/inittab 檔案以外的任何來源啟動 SRC，會造成該目標的反效果。
- 預設的 /etc/inittab 檔案含有一個記錄，可以使用 **startsrc** 指令來啟動列印排程子系統 (**qdaemon**)。一般安裝在 /etc/inittab 檔案中，也會有一些使用 **startsrc** 指令來啟動的其他子系統。因為 **srcmstr** 指令要求 SRC 必須在執行中，所以從 /etc/inittab 檔案中移除 **srcmstr** 常駐程式時，會造成這些 **startsrc** 指令失敗。

註： 只有在 /etc/inittab 檔案中還沒有包含 **srcmstr** 常駐程式的記錄時，才需要使用此程序。

1. 使用 **mkitab** 指令，在 /etc/inittab 檔案中建立 **srcmstr** 常駐程式的記錄。
   例如，若要製作與預設 /etc/inittab 檔案中相同的記錄，請鍵入：

   ```
   mkitab -i fbcheck srcmstr:2:respawn:/usr/sbin/srcmstr
   ```

   -i fbcheck 旗標可確定該記錄會插入所有子系統記錄之前。
2. 若要告知 **init** 指令重新處理 /etc/inittab 檔案，請鍵入：

   ```
   telinit q
   ```

   當 **init** 重新造訪 /etc/inittab 檔案時，它會處理新輸入的 **srcmstr** 常駐程式記錄，並啟動 SRC。

相關概念

子系統控制

traceson 指令可以用來開啟追蹤「系統資源控制器 (SRC)」資源（例如子系統、子系統群組或子伺服器），而 **traceoff** 指令則可以用來關閉這種追蹤。

相關工作

重新整理子系統或子系統群組

使用 **refresh** 指令以告知「系統資源控制器 (SRC)」資源（如子系統或子系統群組）， 重新整理資源本身。

啟動或停止子系統、子系統群組或子伺服器

使用 **startsrc** 指令以啟動「系統資源控制器 (SRC)」資源，如子系統、子系統群組或子伺服器。使用 **stopsrc** 指令以停止 SRC 資源，如子系統、子系統群組或子伺服器。

下列是啟動或停止子系統、子系統群組或子伺服器的必要條件：

- 若要啟動或停止 SRC 資源，SRC 必須是在執行中。在系統起始設定時，通常會啟動 SRC。決定在起始設定期間要啟動哪些處理程序的預設 /etc/inittab 檔案中，會包含一個 **srcmstr** 常駐程式記錄 (SRC)。若要查看 SRC 是在否執行中，請鍵入 ps -A，並尋找名稱為 srcmstr 的處理程序。
- 啟動 SRC 資源的使用者或處理程序必須具有 root 使用者權限。起始設定系統的處理程序（**init** 指令）具有 root 使用者權限。
- 停止 SRC 資源的使用者或處理程序必須具有 root 使用者權限。

startsrc 指令可用於：

- 從 /etc/inittab 檔案，使資源在系統起始設定期間啟動
- 從指令行
- 利用 SMIT

當您啟動子系統群組時，也會啟動其所有子系統。當您啟動子系統時，也會啟動其所有次伺服器。當您啟動次伺服器時，如果其母項子系統尚未執行，則也會啟動該子系統。

當您停止子系統時，其所有次伺服器也會停止。然而，當您停止次伺服器時，其母項子系統的狀態不會變更。

startsrc 及 **stopsrc** 指令都含有可以在本端或遠端主機上提出要求的旗標。請參閱 **srcmstr** 指令，以取得支援遠端 SRC 要求的配置需求。

啟動或停止子系統作業		
作業	***SMIT* 捷徑**	**指令或檔案**
啟動子系統	**smit startssys**	/bin/startsrc -s *SubsystemName*，或編輯 /etc/inittab
停止子系統	**smit stopssys**	/bin/stopsrc -s *SubsystemName*

相關資訊

stopsrc 指令

startsrc 指令

srcmstr 指令

顯示子系統的狀態

使用 **lssrc** 指令以顯示「系統資源控制器 (SRC)」資源的狀態，如子系統、子系統群組或次伺服器。

所有子系統都可以傳回短狀態報告， 包括子系統所屬的群組、子系統是否作用中及其處理程序 ID (PID)。如果子系統沒有使用信號通訊方法，則其程式為傳回完整狀態報告，包括其他狀態資訊。

lssrc 指令提供旗標與參數，以按照名稱或 PID 指定子系統、列出所有子系統、 要求短或完整狀態報告，以及要求本端環境或遠端主機上的 SRC 資源狀態。

請參閱 **srcmstr** 指令，以取得支援遠端 SRC 要求的配置需求。

顯示子系統狀態作業		
作業	***SMIT* 捷徑**	**指令或檔案**
顯示子系統狀態（長格式）	**smit qssys**	**lssrc -l -s** *SubsystemName*
顯示所有子系統的狀態	**smit lsssys**	**lssrc -a**
顯示特定主機上的所有子系統狀態		**lssrc -h***HostName* **-a**

重新整理子系統或子系統群組

使用 **refresh** 指令以告知「系統資源控制器 (SRC)」資源（如子系統或子系統群組）， 重新整理資源本身。

下列是重新整理子系統或子系統群組的必要條件：

· SRC 必須是執行中。

· 您要重新整理的資源不得使用信號通訊方法。

· 您要重新整理的資源必須程式設計為回應重新整理要求。

refresh 指令提供旗標與參數，可以按名稱或 PID 指定子系統。您也可以使用它來要求重新整理子系統或子系統群組， 不論是在本端環境或遠端主機上。請參閱 **srcmstr** 指令，以取得支援遠端 SRC 要求的配置需求。

重新整理子系統或子系統群組		
作業	**SMIT 捷徑**	**指令或檔案**
重新整理子系統	**smit refresh**	**refresh -s Subsystem**

相關工作

啟動系統資源控制器

在系統起始設定期間，會啟動「系統資源控制器 (SRC)」，並在 /etc/inittab 檔案中寫入 /usr/sbin/srcmstr 常駐程式的記錄。

子系統控制

traceson 指令可以用來開啟追蹤「系統資源控制器 (SRC)」資源（例如子系統、子系統群組或子伺服器），而 **traceoff** 指令則可以用來關閉這種追蹤。

使用 **traceson** 指令以啟用追蹤「系統資源控制器 (SRC)」資源，如子系統、子系統群組或次伺服器。

使用 **tracesoff** 指令以停用追蹤「系統資源控制器 (SRC)」資源，如子系統、子系統群組或次伺服器。

traceson 及 **traceoff** 指令可以用來在特定主機上遠端啟用或停用追蹤。請參閱 **srcmstr** 指令，以取得支援遠端 SRC 要求的配置需求。

必要條件

· 若要啟用或停用追蹤 SRC 資源，SRC 必須在執行中。

· 您要追蹤的資源不得使用信號通訊方法。

· 您要追蹤的資源必須以程式設計為回應追蹤要求。

啟用/停用子系統、子系統群組或次伺服器作業		
作業	***SMIT* 捷徑**	**指令或檔案**
啟用子系統追蹤（短格式）	**smit tracessyson**	**traceson -s Subsystem**
啟用子系統追蹤（長格式）	**smit tracessyson**	**traceson -l -s Subsystem**
停用子系統追蹤	**smit tracessysoff**	**tracesoff -s Subsystem**

相關工作

啟動系統資源控制器

在系統起始設定期間，會啟動「系統資源控制器 (SRC)」，並在 /etc/inittab 檔案中寫入 /usr/sbin/srcmstr 常駐程式的記錄。

作業系統檔案

檔案是用來在作業系統中進行資訊的輸入和輸出 (I/O)，將軟硬體的存取標準化。

輸入 動作是在修改或寫入檔案內容時發生。輸出 動作則是當讀取檔案內容，或是傳送檔案內容到另外一個檔案時發生。例如要將檔案列印出來時，系統會讀取文字檔中的資訊，並將該資訊寫到代表印表機的檔案。

檔案類型

系統可以辨識的檔案類型為**一般**、**目錄**或**特殊**。然而，作業系統使用這些基本類型的變化類型。

存在的基本檔案類型如下：

項目	說明
一般	儲存資料（文字、二進位以及可執行檔）
目錄	包含用來存取其他檔案的資訊
特殊	定義 FIFO（先進先出法）管線檔案或實體裝置

所有系統可辨識的檔案類型皆為這些種類的其中一個。然而，作業系統使用這些基本類型的變化類型。

一般檔案

一般檔案是最常見的檔案，用於容納資料。一般檔案的格式為文字檔或二進位檔。

文字檔

文字檔是包含以 ASCII 格式的文字來儲存且可讓使用者讀取之資訊的一般檔案。您可以顯示並且列印這些檔案。文字檔字行不可包含 NUL 字元，且長度不可超出 {LINE_MAX} 個位元組，包括換行字元。

文字檔一詞並不表示不包含控制或其他不可列印字元 （NUL 例外）。因此，將文字檔列為輸入或輸出的標準公用程式可以輕易地處理特殊字元，也可以在它們的個別區段內明確地說明其限制。

二進位檔

二進位檔是包含電腦可讀資訊的一般檔案。二進位檔可能是指示系統完成某項工作的可執行檔。指令與程式將儲存在可執行的 二進位檔案。特殊的編譯程式將轉換 ASCII 文字為二進位字碼。

文字及二進位檔之間的差異僅在於： 文字檔的字行少於 {LINE_MAX} 個位元組，且沒有 NUL 字元，每一行都是換行字元作為結束。

目錄檔

目錄檔包含系統存取所有類型的檔案時所需的資訊，但目錄檔不包含實際的檔案資料。結果，目錄比一般檔案佔用的空間較少，且提供檔案系統結構彈性及深度。每一個目錄項目代表一個檔案或子目錄。每一個項目包含檔名及檔案索引節點參考號碼（*i-node* 號碼）。i-node 號碼 指向指派至檔案之鍵值唯一的索引節點。i-node 號碼說明與檔案相關的資料位置。 目錄是由個別的指令集所產生和控制。

特殊檔案

特殊檔案定義系統裝置或處理程序所建立的暫存檔。特殊檔案的基本類型為 FIFO（先進先出法）、區塊及字元。FIFO 檔案亦稱為管線。管線是由某個程序建立，可暫時與其他程序通訊。 這些檔案在第一個程序完成時便會消失。區塊及字元檔則定義裝置。

每一個檔案都有一組許可權（稱為存取模式），用以決定誰可以讀取、修改或執行該檔。

相關概念

檔案和目錄存取模式

每一個檔案都有一個擁有者。對新檔案來說，建立此檔的使用者便是該檔的擁有者。擁有者會指派存取模式給此檔案。 存取模式會授與其他系統使用者許可權來讀取、修改或執行該檔。僅有檔案擁有者或具 root 權限的使用者可以變更此檔的存取模式。

檔案命名慣例
每一個檔案名稱在其所在的目錄中必須是唯一的。 這亦可以確保檔案在檔案系統中具有唯一的路徑名稱。

檔案命名的原則有：

- 檔名長度最多可到 255 字元，且可以包含字母、數字及底線。
- 作業系統是區分大小寫的，意即其區分檔名中的大小寫字母。因此，FILEA、FiLea 以及 filea 是三個不同的檔名，即使他們位於相同的目錄中。
- 檔名應該儘可能的具說明性及意義。
- 目錄的命名慣例與檔案相同。
- 某些字元對作業系統而言具有特殊意義。當您為檔案命名時，請避免使用這些名稱。這些字元包括下列各項：

```
/ \ " ' * ; - ? [ ] ( ) ~ ! $ { } &lt > # @ & | 空格 tab 新增一行
```

- 檔名若以點 (.) 作開頭，則會隱藏於一般目錄清單中。在輸入 **ls** 指令時若包含 **-a** 旗標，會將隱藏式檔案與一般檔案和目錄一起列出。

檔案路徑名稱
檔案系統中每一個檔案及目錄的路徑名稱，是由在目錄樹結構中位於其前的每一個目錄名稱組成。

因為檔案系統中的所有路徑皆起源於 /(root) 目錄，所以檔案系統中每一個檔案和根目錄之間的關係都是唯一的，這就是所謂的絕對路徑名稱。絕對路徑名稱是以斜線 (/) 符號開頭。例如，檔案 h 的絕對路徑名稱可能是 /B/C/h。請注意，同一系統中可以存在兩個名為 h 的檔案。因為這兩個檔案的絕對路徑不同（/B/h 及 /B/C/h），所以在系統中，每個名為 h 的檔案都有一個唯一的名稱。除了最終元件，路徑名稱的每一個元件是目錄。路徑名稱的最後一個組成部分可以是檔名。

註 : 路徑名稱的長度不能超出 1023 個字元。

以萬用字元及 *Meta* 字元做型樣相符
萬用字元提供了一個方便的方式來指定多個檔案名稱或目錄名稱。

萬用字元為星號 (*) 及問號 (?)。Meta 字元是開啟及關閉的中括弧 ([ ])、連字號 (-) 及驚嘆號 (!)。

使用 * 萬用字元的型樣相符
使用星號 (*) 可以找出相符的任何序列或字串。

(*) 代表任何字元，包括沒有字元。

請參閱下列範例：

- 如果在您的目錄中具有下列檔案：

```
1test 2test afile1 afile2 bfile1 file file1 file10 file2 file3
```

  而您只要參照以 file 開頭的檔案，請使用：

```
file*
```

  所選取的檔案會是：file、file1、file10、file2 及 file3。
- 若只要參考包含 file 字詞的檔案，請使用：

```
*file*
```

  所選取的檔案會是：afile1、afile2、bfile1、file、file1、file10、file2 及 file3。

使用 ? 萬用字元的型樣相符
使用 ? 可以比對任何一個字元。

? 表示單一字元。請參閱下列範例：

· 若只要參考以 **file** 開頭並以單一字元結束的檔案，請使用：

```
file?
```

所選取的檔案會是：file1、file2、file3。

· 若只要參考以 **file** 開頭並以兩個字元結束的檔案，請使用：

```
file??
```

所選取的檔案會是：file10。

使用 *[] shell Meta* 字元的型樣相符
Meta 字元提供另一種萬用字元表示法類型， 方法是把所要的字元含括在 [ ] 裡。它就像使用 ?， 但可讓您選擇要配對的特定字元。

[ ] 也可讓您使用連字號 (-) 來指定一定範圍內的值。若要指定英文字母的所有字母，請使用 [[:alpha:]]。若要指定英文字母的所有小寫字母，請使用 [[:lower:]]。

請參閱下列範例：

· 若只要參考以 1 或 2 結尾的檔案，請使用：

```
*file[12]
```

所選取的檔案會是：afile1、afile2、file1 及 file2。

· 若只要參照以任何數字開頭的檔案，請使用：

```
[0123456789]* or [0-9]*
```

所選取的檔案會是：1test 及 2test。

· 若只要參照不是以 a 開頭的檔案，請使用：

```
[!a]*
```

所選取的檔案會是：1test、2test、bfile1、file、file1、file10、file2 及 file3。

型樣相符對正規表示式
一般表示式可讓您自一組字元字串中 選取特定字串。一般而言，一般表示式的使用與文字處理程序相關。

一般表示式可以代表範圍很大的可能字串。 然而許多一般表示式依據供作上下文不變性之現行語言環境、國際化特性而有 不同解譯。

請參閱下列的比較範例：

型樣相符	正規表示式
*	.*
?	.
[!a]	[^a]
[abc]	[abc]

```
[[:alpha:]]             [[:alpha:]]
```

請參閱 *Commands Reference, Volume 1* 中的 **awk** 指令，以取得完整語法。

管理檔案

有許多在您的系統上使用檔案的方法。通常您會用文字編輯器來建立一個文字檔。

在 UNIX 環境中常用的編輯器為 vi 及 ed。因為有數種文字編輯器可使用，您可以選擇您習慣使用的編輯器。

您也可以使用輸入及輸出重新導向來建立檔案。您可以將指令的輸出傳送到新的檔案，或是將它附加到現有檔案。

在建立並且修改檔案之後，您可能必須複製或把檔案從一個目錄移動到另外一個目錄、 更改檔名以區分不同版本的檔案，或是為相同的檔案指定不同的名稱。當您處理不同的專案時，可能也會需要建立新的目錄。

您也可能會需要刪除某些檔案。隨著包含舊的或無用的檔案增加，您的目錄很快地會變得雜亂。若要釋放系統上的儲存體空間，請確定您刪除的檔案是不再需要的。

刪除檔案（*rm* 指令）

使用 **rm** 指令來移除您不再需要的檔案。

rm 指令會從目錄的清單中，將指定檔案的項目、檔案群組或特定的選取檔案移除。當您使用 **rm** 指令時，在檔案移除之前，並不需要使用者確認、讀取許可權及寫入許可權。然而，您必須對包含此檔案的目錄具有寫入權。

以下是 **rm** 指令的用法範例：

· 若要刪除檔案 myfile，請鍵入：

```
rm myfile
```

· 若要一個接一個刪除 mydir 目錄下的所有檔案，請鍵入：

```
rm -i mydir/*
```

在每個檔名顯示之後，輸入 y，並按 Enter 鍵，以刪除檔案。若要保留檔案時，只要按 Enter 鍵即可。

請參閱 *Commands Reference, Volume 4* 中的 **rm** 指令，以取得完整語法。

移動並重新命名檔案（*mv* 指令）

使用 **mv** 指令，將檔案和目錄從某個目錄移至另一個，或更改一個檔案或目錄的名稱。如果您並沒有為一個檔案或目錄指定新名稱就把它移至一個新的目錄，該檔案或目錄的原始名稱就會保留下來。

小心： 若沒有指定 **-i** 旗標，**mv** 指令會改寫許多現有檔案。**-i** 旗標會在改寫檔案之前，提示您做確認。 **-f** 旗標則不會提示您。如果 **-f** 和 **-i** 這兩個旗標被組合在一起，將優先採用最後指定的旗標。

以 mv 指令來移動檔案

以下是 **mv** 指令的用法範例：

· 若要將檔案移到另一個目錄，請給它一個新名稱，請鍵入：

```
mv intro manual/chap1
```

此動作會把 intro 檔案移至 manual/chap1 目錄。名稱為 **intro** 的檔案會從現行目錄中移除，而相同的檔案會以 chap1 出現在 manual 目錄中。

· 若要將檔案移到另一個目錄，並維持名稱不變，請鍵入：

```
mv chap3 manual
```

此動作會把 chap3 移至 manual/chap3。

以 mv 指令來更改檔名

使用 **mv** 指令來變更檔名，而不將該檔案移至其他目錄。

若要更改檔案名稱，請鍵入：

```
mv appendix apndx.a
```

此動作會把 appendix 檔案更名成 apndx.a。如果已經有名稱為 apndx.a 的檔案存在，它的舊內容會被 appendix 檔案的內容置換掉。

請參閱 *Commands Reference, Volume 3* 中的 **mv** 指令，以取得完整語法。

複製檔案（*cp* 指令）

使用 **cp** 指令，把 *SourceFile* 或 *SourceDirectory* 參數所指定的檔案或目錄內容副本，建立到 *TargetFile* 或 *TargetDirectory* 參數所指定的檔案或目錄。

如果指定作為 *TargetFile* 的檔案已經存在，複製的動作會不經警告就把原始檔案的內容寫入覆蓋掉。如果您正在複製一個以上的 *SourceFile*，目標端必須為一個目錄。

如果在新目的地已有相同名稱的檔案存在，所複製的檔案將會改寫新目的地上的檔案。因此，這是為檔案副本指定新名稱的好方法，這樣可以確保目的地目錄中沒有相同名稱的檔案存在。

若要把 *SourceFile* 的副本放置在一個目錄內，請對 *TargetDirectory* 參數指定一個已存在目錄的路徑。除非您在路徑的尾端指定好一個新檔名，否則在複製檔案到一個目錄時，會維持它們個別的名稱。如果您指定了 **-r** 或 **-R** 旗標，**cp** 指令也可以把整個目錄複製到其他目錄內。

您也可以用 **-R** 旗標來複製特殊裝置的檔案。指定 **-R** 旗標會在新的路徑名稱下重新產生特殊檔案。指定 **-r** 旗標會導致 **cp** 指令試圖把特殊檔案拷貝成一般檔案。

以下是 **cp** 指令的用法範例：

- 若要建立現行目錄中的檔案副本，請鍵入：

  ```
  cp prog.c prog.bak
  ```

  此動作會把 prog.c 複製成 prog.bak。如果 prog.bak 檔案不存在，**cp** 指令即會建立它。如果已存在，**cp** 指令即會將它置換成 prog.c 檔的副本。
- 若要將現行目錄中的檔案複製到另一個目錄，請鍵入：

  ```
  cp jones /home/nick/clients
  ```

  此動作會把 jones 檔案複製到 /home/nick/clients/jones。
- 若要將目錄中的所有檔案複製到新目錄，請鍵入：

  ```
  cp /home/janet/clients/* /home/nick/customers
  ```

  此動作只會把 clients 目錄中的檔案複製到 customers 目錄中。
- 若要複製特定的一組檔案到另一個目錄，請鍵入：

  ```
  cp jones lewis smith /home/nick/clients
  ```

  此動作會把在您現行工作目錄的 jones、lewis 和 smith 檔案複製到 /home/nick/clients 目錄中。
- 若要使用型樣相符的字元來複製檔案，請鍵入：

  ```
  cp programs/*.c .
  ```

  此動作會把 programs 目錄中以 .c 結尾的檔案複製到現行目錄（以單一點 (.) 表示）。您必須在 c 和最後的點之間鍵入一個空格。

請參閱 *Commands Reference, Volume 1* 中的 **cp** 指令，以取得完整語法。

尋找檔案（*find* 指令）

使用 **find** 指令，可以在目錄樹中遞迴地搜尋每一個指定的 *Path*， 同時使用下列文字提供的詞彙，尋找符合已寫好的布林表示式的檔案。

find 指令的輸出結果取決於 *Expression* 參數所指定的詞彙。

以下是 find 指令的用法範例：

- 若要列出檔案系統中名稱為 .profile 的所有檔案，請鍵入：

```
find / -name .profile
```

此指令會搜尋整個檔案系統，並寫入所有名稱為 .profile 的檔案之完整路徑名稱。斜線 (/) 告訴 **find** 指令去搜尋 /（根）目錄及其所有子目錄。

為節省時間，請指定您認為檔案可能會在的地方，以限制搜尋的範圍。

- 若要列出在現行目錄樹中具有特定許可權代碼 0600 的檔案，請鍵入：

```
find . -perm 0600
```

此動作會列出只具有擁有者讀取和擁有者寫入許可權的檔案名稱。點 (.) 會叫 **find** 指令去搜尋現行目錄及其子目錄。若需許可權代碼的說明，請參閱 **chmod** 指令。

- 若要搜尋數個目錄中含特定許可權代碼的檔案，請鍵入：

```
find manual clients proposals -perm -0600
```

此動作會列出具有檔案持有者讀取和寫入許可、和所有可能為其他許可權的檔案名稱。manual、clients 和 proposals 目錄及其子目錄都會被搜尋。在上一個範例中，-perm 0600 只會選取許可權代碼完全符合 0600 的檔案。在此範例中，-perm -0600 會選取許可權代碼允許 0600 所指出之存取權和高過 0600 層次之其他存取的檔案。此規則也適用於許可字元 0622 和 2744。

- 若要列出現行目錄中在過去 24 小時內有作變更的所有檔案，請鍵入：

```
find . -ctime 1
```

- 若要搜尋含多個連結的一般檔案，請鍵入：

```
find . -type f -links +1
```

此指令會列出具有一個以上鏈結 (-links +1) 的一般檔案名稱 (-type f)。

註：每一個目錄至少都有兩個鏈結：分別是在其母目錄及其本身的 .（點）項目。如需多個檔案鏈結的相關資訊，請參閱 **ln** 指令。

- 若要搜尋長度剛好是 414 位元組的所有檔案，請鍵入：

```
find . -size 414c
```

請參閱 *Commands Reference, Volume 2* 中的 **find** 指令，以取得完整語法。

顯示檔案類型（*file* 指令）

使用 **file** 指令，可以讀取由 *File* 或 **-f** *FileList* 參數指定的檔案，並嘗試依照類型替檔案分類。然後此指令會把檔案類型寫到標準輸出裝置上。

如果一個檔案被認為是 ASCII，此 **file** 指令會檢查前 512 位元組，來判斷它的語言。如果一個檔案不被認為是 ASCII（美國國家標準交換碼），此 **file** 指令會試圖進一步去判斷其是否為二進位資料檔案或是一個包含擴充字元的文字檔。

如果 *File* 參數指定了可執行檔或物件模組檔案，並且版本號碼大於零時，則此 **file** 指令會顯示版本戳記。

此 **file** 指令會使用 /etc/magic 檔案來識別具有識別碼的檔案；亦即，任何包含指出類型之數值或字串常數的檔案。

以下是 **file** 指令的用法範例：

· 若要顯示檔案 myfile 包含的資訊類型，請鍵入：

```
file myfile
```

這會顯示 myfile 的檔案類型（例如：目錄、資料、ASCII 文字、C 程式語言的原始檔或保存檔）。

· 若要顯示 filenames.lst 檔案中每個命名檔案類型，請鍵入：

```
file -f filenames.lst
```

此動作會顯示 filenames.lst 檔案中每個檔案的類型。每個檔案名稱都必須單獨出現在一行上。

· 若要建立包含所有現行目錄中檔案名稱的 filenames.lst 檔案，請鍵入：

```
ls > filenames.lst
```

編輯 filenames.lst 檔成為您想要的檔案。

請參閱 *Commands Reference, Volume 2* 中的 **file** 指令，以取得完整語法。

顯示檔案內容的指令（*pg*、*more*、*page* 及 *cat* 指令）
這些 **pg**、**more** 和 **page** 指令可讓您檢視某個檔案內容，並且控制顯示檔案內容的速度。

您也可以在您的畫面上使用 **cat** 指令來顯示一個或多個檔案內容。同時使用 **cat** 和 **pg** 這兩個指令，可讓您以一次一個全畫面的方式來閱讀檔案內容。

您也可以使用輸入及輸出重新導向來顯示檔案的內容。

相關概念
輸入及輸出重新導向
AIX 作業系統可讓您使用特定的 I/O 指令與符號，操作您的系統之輸入和輸出 (I/O) 資料。

使用 *pg* 指令
使用 **pg** 指令可以讀取 **File** 參數指名的檔案，並一次一個畫面，將它們寫入標準輸出。

如果您指定連字號 (-) 作為 **File** 參數，或執行 **pg** 但不加上選項，**pg** 指令會讀取標準輸入。每個畫面之後都會跟著一個提示。如果您按 Enter 鍵，另一個畫面就會接著顯示出來。與 **pg** 指令一起使用的次指令可讓您檢閱已傳遞的內容。

例如，若要以每次一頁的方式查看檔案 myfile 內容，請鍵入：

```
pg myfile
```

請參閱 *Commands Reference, Volume 4* 中的 **pg** 指令，以取得完整語法。

使用 *more* 或 *page* 指令
使用 **more** 或 **page** 指令， 一次一個畫面來顯示連續文字。

在每一個畫面顯示後，它會暫停一下，並且在畫面的最底端顯示檔名和完成的百分比（例如，myfile (7%)）。如果您在這個時候按下了 Enter 鍵，**more** 這個指令就會在畫面底端顯示附加的一行。如果您按空白鍵，**more** 指令就會顯示另一個文字畫面。

註： 在某些終端機機型上，**more** 指令會在顯示下一個畫面的文字之前清除畫面，而不是捲動畫面。

例如，若要檢視檔案 myfile，請鍵入：

```
more myfile
```

按空白鍵來檢視下一個畫面。

請參閱 *Commands Reference, Volume 3* 中的 **more** 指令，以取得完整語法。

cat 指令
使用 **cat** 指令，可以依序讀取每個 *File* 參數，並且將它寫入標準輸出。

請參閱下列範例：

- 若要顯示檔案 notes 的內容，請鍵入：

```
cat notes
```

如果檔案的行數多於 24，某些捲動的檔案內容會被畫面切掉。若要一次一頁地把檔案列出，請使用 **pg** 指令。

- 若要顯示檔案 notes、notes2 和 notes3 的內容，請鍵入：

```
cat notes notes2 notes3
```

請參閱 *Commands Reference, Volume 1* 中的 **cat** 指令，以取得完整語法。

尋找檔案內的字串（*grep* 指令）

使用 **grep** 指令，可以在指定的檔案中搜尋由 *Pattern* 參數所指定的型樣，然後將符合型樣的每一行寫入標準輸出中。

以下是 **grep** 指令的用法範例：

- 若要在檔案 pgm.s 中搜尋含部分型樣相符字元 *、^、?、[、]、\(、\)、\{ 和 \} 的型樣，在此例子中，資料行的開頭是任意大小寫字母，請鍵入：

```
grep "^[a-zA-Z]" pgm.s
```

此動作會顯示以字母開頭之 pgm.s 檔案中的所有指令行。

- 若要顯示檔案 sort.c 中與特定型樣不符的所有行，請鍵入：

```
grep -v bubble sort.c
```

此指令會顯示 sort.c 檔案中沒有包含 bubble 這個字的所有字行。

- 若要顯示 **ls** 指令輸出中與字串 staff 相符的字行，請鍵入：

```
ls -l | grep staff
```

請參閱 *Commands Reference, Volume 2* 中的 **grep** 指令，以取得完整語法。

排序文字檔（*sort* 指令）

使用 **sort** 指令，可以在 **File** 參數所指定的檔案中，照字母排列字行，並將結果寫入標準輸出。

如果 **File** 參數指定了多個檔案，**sort** 指令就會連結這些檔案，並且將它們照字母順序排列成一個檔案。

註： **sort** 指令會區分大小寫，並且先排序大寫字母再排序小寫字母（此動作視語言環境而定）。

在下列範例中，檔案 names 的內容為：

```
marta
denise
joyce
endrica
melanie
```

且 states 的檔案內容為：

```
texas
colorado
ohio
```

- 若要顯示檔案 names 排序後的內容，請鍵入：

```
sort names
```

系統會顯示如下的資訊：

```
denise
endrica
joyce
marta
melanie
```

· 若要顯示 names 和 states 檔案排序後的內容，請鍵入：

```
sort names states
```

系統會顯示如下的資訊：

```
colorado
denise
endrica
joyce
marta
melanie
ohio
texas
```

· 若要將檔案 names 的原內容取代為其經過排序的內容，請鍵入：

```
sort -o names names
```

此動作會將 names 檔案的內容置換成排序過的相同資料。

請參閱 *Commands Reference, Volume 5* 中的 **sort** 指令，以取得完整語法。

比較檔案（*diff* 指令）

使用 **diff** 指令來比較文字檔。它會比較單一檔案或目錄的內容。

當針對一般檔案執行 **diff** 指令，以及比較不同目錄的文字檔時， **diff** 這個指令會告知在檔案中的那一行需要變更，它們才會相符。

以下是 **diff** 指令的用法範例：

· 若要比較兩個檔案，請鍵入：

```
diff chap1.bak chap1
```

此指令會顯示 chap1.bak 與 chap1 兩個檔案之間的差異。

· 若要比較兩個檔案，並忽略空格數的差異，請鍵入：

```
diff -w prog.c.bak prog.c
```

如果兩個檔案之間只有空格及跳格字元數目不同，**diff -w** 指令會將這兩個檔案視為相同。

請參閱 *Commands Reference, Volume 2* 中的 **diff** 指令，以取得完整語法。

計算檔案中的字詞、字行及位元組的個數（*wc* 指令）

使用 **wc** 指令來計算由 *File* 參數所指定的檔案中的字行、 字詞及位元組的個數。

如果一個檔案並未指定 *File* 參數，就會使用到標準輸入。這個指令會把比較的結果寫到輸準輸出裝置上，並且為所有被指名的檔案保存一個加總計數。如果已指定了旗標，則旗標的次序會判斷輸出裝置的次序。字組是藉由空格、跳格字元、或換行字元的區隔，而定義為一連串的字元。

將檔案指定在指令行上的時候，檔案的名稱會連同計數一起印出。

請參閱下列範例：

· 若要顯示檔案 chap1 的行數、字數及位元組數，請鍵入：

```
wc chap1
```

此動作會把在 chap1 這個檔案的行數、字數、和位元組數顯示出來。

· 若只要顯示位元組和字數，請鍵入：

```
wc -cw chap*
```

此指令會把檔名以 chap 做為開頭的每個檔案的位元組數和字數顯示出來，並且會顯示出總計有幾個檔案。

請參閱 *Commands Reference, Volume 6* 中的 **wc** 指令，以取得完整語法。

顯示檔案的前幾行（*head* 指令）
使用 **head** 指令， 將每個指定的檔案或標準輸入的前幾行寫入標準輸出。

如果 **head** 這個指令沒有指定旗標，則依預設會顯示檔案的前 10 行。

例如，若要顯示 Test 檔案的前五行，請鍵入：

```
head -5 Test
```

請參閱 *Commands Reference, Volume 2* 中的 **head** 指令，以取得完整語法。

顯示檔案的最後幾行（*tail* 指令）
使用 **tail** 指令，可以將 *File* 參數所指定的檔案， 從指定的地方開始寫入標準輸出。

請參閱下列範例：

· 若要顯示 notes 檔案的最後十行，請鍵入：

```
tail notes
```

· 若要指定從 notes 檔案尾端開始讀取的行數，請鍵入：

```
tail -20 notes
```

· 若要一次一頁、從第 200 個位元組開始顯示 notes 檔案，請鍵入：

```
tail -c +200 notes | pg
```

· 若要追蹤檔案 accounts 的成長情況，請鍵入：

```
tail -f accounts
```

此指令會把 accounts 這個檔案內容的最後 10 行顯示出來。此 **tail** 指令會在字行附加到 accounts 檔案時，持續顯示它們。它會一直顯示到您按下 (Ctrl-C) 鍵順序，才會停止顯示。

請參閱 *Commands Reference, Volume 5* 中的 **tail** 指令，以取得完整語法。

剪下文字檔的區段（*cut* 指令）
使用 **cut** 指令，將所選取的位元組、字元或欄位， 從檔案的每一行寫到標準輸出。

請參閱下列範例：

· 若要顯示檔案每行的幾個欄位，請鍵入：

```
cut -f1,5 -d: /etc/passwd
```

此動作會把系統密碼檔案的登入名稱和完整的使用者名稱欄位顯示出來。這些是以冒號 (-d:) 來隔開的第一和第五欄 (-f1,5)。

· 如果 /etc/passwd 這個檔案的內容看起來像如下所示：

```
su:*:0:0: 具有特殊專用權的使用者 :/:/usr/bin/sh
daemon:*:1:1::/etc:
bin:*:2:2::/usr/bin:
sys:*:3:3::/usr/src:
adm:*:4:4:system administrator:/var/adm:/usr/bin/sh
```

```
pierre:*:200:200:Pierre Harper:/home/pierre:/usr/bin/sh
joan:*:202:200:Joan Brown:/home/joan:/usr/bin/sh
```

此 **cut** 指令會產生：

```
su:具有特殊專用權的使用者
daemon:
bin:
sys:
adm:system administrator
pierre:Pierre Harper
joan:Joan Brown
```

請參閱 *Commands Reference, Volume 1* 中的 **cut** 指令，以取得完整語法。

貼上文字檔的區段（*paste* 指令）

使用 **paste** 指令，把最多 12 個檔案的字行合併成一個檔案。

請參閱下列範例：

· 如果您有一個包含下列文字的 names 檔案：

```
rachel
jerry
mark
linda
scott
```

再者，還有另外一個 places 檔案包含下列文字：

```
New York
Austin
Chicago
Boca Raton
Seattle
```

再者，還有另外一個 dates 檔案包含下列文字：

```
February 5
March 13
June 21
July 16
November 4
```

若要將 names、places 及 dates 等檔案的文字貼在一起，請鍵入：

```
paste names places dates > npd
```

此指令會建立一個名為 npd 的檔案，其中一個直欄包含 names 的資料，另一直欄包含 places 檔案的資料，第三直欄則包含 dates 檔案的資料。npd 檔現在包含了下列資料：

```
rachel          New York        February 5
jerry           Austin          March 13
mark            Chicago         June 21
linda           Boca Raton      July 16
scott           Seattle         November 4
```

跳格字元會把每行的名稱、位置、和日期隔開。因為欄標停駐點設定為每隔八個字元一欄，所以這些直欄無法對齊。

· 若要以定位字元以外的字元隔開直欄，請鍵入：

```
paste -d"!@" names places dates > npd
```

此動作會把！和 @ 替代成直欄分隔字元。如果 names、places 和 dates 這三個檔案和範例 1 的相同，那麼 npd 檔案就會包含下列：

```
rachel!New York@February 5
jerry!Austin@March 13
mark!Chicago@June 21
```

```
linda!Boca Raton@July 16
scott!Seattle@November 4
```

· 若要在四個直欄中列出現行目錄，請鍵入：

```
ls | paste - - - -
```

每一個連字號 (-) 都會告知 **paste** 指令去建立一個直欄，並且包含了從標準輸入讀取的資料。第一行資料放置在第一個直欄、第二行資料放在第二個直欄，如此依序存放下去。

請參閱 *Commands Reference, Volume 4* 中的 **paste** 指令，以取得完整語法。

編寫文字檔中的行號（*nl* 指令）

使用 **nl** 指令來讀取已指定的檔案（根據預設值為標準輸入）， 替輸入的資料編上行號，並且將行號寫入標準輸出。

請參閱下列範例：

· 若只要對非空白的行編碼，請鍵入：

```
nl chap1
```

此動作會把 chap1 檔案的號碼列出來，而且在主體區段中，只為非空白行編號。

· 若要對所有行編碼，請鍵入：

```
nl -ba chap1
```

此動作會把 chap1 檔案中所有的行都加上編號，包括空白行。

請參閱 *Commands Reference, Volume 4* 中的 **nl** 指令，以取得完整語法。

移除文字檔中的直欄（*colrm* 指令）

使用 **colrm** 指令，從檔案中移除指定的直欄。輸入的資料是從標準輸入取得。輸出的資料會傳送到標準輸出裝置。

如果是使用一個參數來呼叫這個指令，就會將每一行中從指定直欄到最後一個直欄全部移除。如果使用兩個參數來呼叫這個指令，從第一到第二個指定的直欄都會被移除。

註： 直欄的編號會從直欄 1 開始。

請參閱下列範例：

· 若要將直欄從 text.fil 檔案中移除，請鍵入：

```
colrm 6 < text.fil
```

如果 text.fil 檔案包含：

```
123456789
```

那麼 **colrm** 這個指令會顯示：

```
12345
```

請參閱 *Commands Reference, Volume 1* 中的 **colrm** 指令，以取得完整語法。

檔案及目錄鏈結

鏈結是檔名和索引節點參考號碼（i-node 號碼）之間的連線，inode 號碼是檔案的內部表示。由於目錄項目包含檔名與 inode 號碼的配對，所以每一個目錄項目即為鏈結。

索引節點號碼實際上是定義檔案，而不是檔名。藉由鏈結不同的名稱可識別其 inode 號碼或檔案。例如，i-node 號碼 798 包含 Omaha 辦公室在六月份的銷售備忘錄。目前此備忘錄的目錄項目如下：

i-node 號碼	檔名
798	**memo**

由於此資訊與 sales 和 omaha 目錄中儲存的資訊相關，所以可依需要使用鏈結來共用資訊。使用 **ln** 指令，可建立鏈結至這些目錄。現在檔案有下列三個檔名：

i-node 號碼	檔名
798	memo
798	sales/june
798	omaha/junesales

當您使用 **pg** 或 **cat** 指令來檢視這三個檔名時，會顯示相同的資訊。如果您從這三個檔名中的任何一個來編輯 inode 號碼的內容，若資料內容有變更，則全部檔名所顯示的資料內容都會反應出這些變更。

鏈結類型
鏈結類型有兩種：固定及符號。

鏈結是使用 **ln** 指令來產生的，其類型如下：

項目	說明
固定鏈結	允許從新檔名來存取某個檔案的資料。固定鏈結確保檔案的存在。當移除最後一個固定鏈結時，亦會刪除 inode 號碼及其資料。固定鏈結只能建於位於相同檔案系統的檔案之間。
符號鏈結	允許從新檔名來存取其他檔案系統中的資料。符號鏈結是包含路徑名稱的一種特殊檔案類型。當處理程序發現符號鏈結時，處理程序會搜尋該路徑。符號鏈結無法防止檔案從檔案系統中遭到刪除。

註 : 不論建立多少鏈結，建立檔案的使用者仍會保有該檔案的所有權。僅檔案的擁有者或 root 使用者，才可以設定該檔案的存取模式。然而，透過適當的存取模式，仍可從鏈結的檔名來變更檔案。

只要有一個固定鏈結指向檔案的 inode 號碼，該檔案或目錄就會存在。在 **ls -l** 指令所顯示的長報表中，會提供通往每個檔案和子目錄的固定鏈結數目。作業系統對所有的固定鏈結皆一視同仁，而不管建立鏈結的先後次序。

鏈結檔案（*ln* 指令）
對於使用位於不同地點的相同資料來說，使用 **ln** 指令來鏈結檔案是很方便的方式。

建立鏈結的方式是提供替代名稱給原來的檔案。對於大型的檔案，例如資料庫或郵寄清單，經由使用鏈結，不需複製即可讓許多使用者共用該檔。鏈結不僅省節磁碟空間，對於檔案的變更，亦會自動反應在所有的鏈結檔案中。

ln 指令會將 **SourceFile** 參數中指定的檔案鏈結到 **TargetFile** 參數中指定的檔案，或鏈結到 **TargetDirectory** 參數中指定的另一個目錄中的相同檔名。根據預設值，**ln** 指令是建立固定鏈結。若要使用 **ln** 指令建立符號鏈結，請加入 **-s** 旗標。

註 : 若沒有使用 **-s** 旗標，就您不能跨越檔案系統鏈結檔案。

如果您要將檔案鏈結至一個新名稱，則只能列出一個檔案。如果您鏈結至目錄，則您可列出一個以上的檔案。

TargetFile 參數是選用性。如果您沒有指定目標檔，**ln** 指令就會在您的現行目錄中建立一個檔案。新檔案會繼承 **SourceFile** 參數中指定的檔名。

請參閱下列範例：

· 若要建立檔案 chap1 的鏈結，請鍵入：

```
ln -f chap1 intro
```

此指令會將 chap1 鏈結到新名稱 intro。有使用 **-f** 旗標時，即使檔名 intro 不存在，亦會建立該檔案。如果 intro 已存在，則以 chap1 的鏈結來置換該檔案。chap1 和 intro 這兩個檔名皆會參照相同的檔案。

- 若要將檔案 index 鏈結到另一個名為 manual 目錄中的相同名稱，請鍵入：

```
ln index manual
```

此動作會將 index 鏈結到新名稱 manual/index。

- 若要將數個檔案鏈結到另一個目錄中的名稱，請鍵入：

```
ln chap2 jim/chap3 /home/manual
```

這會將 chap2 鏈結到新名稱 /home/manual/chap2，並將 jim/chap3 鏈結到 /home/manual/chap3。

- 若要使用 **ln** 指令搭配型樣相符字元，請鍵入：

```
ln manual/* .
```

註： 您必須在星號和句點之間輸入一個空格。

如此會將所有位於 manual 目錄中的檔案鏈結至現行目錄中，點 (.) 讓它們具有與 manual 目錄中相同的名稱。

- 若要建立符號鏈結，請鍵入：

```
ln -s /tmp/toc toc
```

此動作在現行目錄中建立一個符號鏈結 **toc**。toc 檔案會指向 /tmp/toc 檔案。如果 /tmp/toc 檔案存在，**cat toc** 指令就會列出其內容。

- 若要保存相同結果，但不指定 **TargetFile** 參數，請鍵入：

```
ln -s /tmp/toc
```

請參閱 *Commands Reference, Volume 3* 中的 **ln** 指令，以取得完整語法。

移除鏈結的檔案的指令

rm 指令會從您指出的檔名移除鏈結。

當數個固定鏈結檔名的其中一個被刪除時，該檔案並沒有被完全刪除，因為它仍會以其他名稱存在。當移除 inode 號碼的最後一個鏈結時，亦同時移除資料。然後，該 i-node 號碼即可供系統重覆使用。

請參閱 *Commands Reference, Volume 3* 中的 **rm** 指令，以取得完整語法。

DOS 檔案

AIX 作業系統可讓您在系統上使用 DOS 檔案。

請將您要使用的 DOS 檔案複製到磁片上。您的系統可以依正確的格式，將這些檔案讀到基本作業系統目錄中，再以 DOS 格式讀回磁片中。

註： 萬用字元 * 及 ？（星號及問號）無法正確地用在本節所討論的指令上（雖然可用於基本作業系統 shell）。如果您未指定副檔名，仍將會使檔名符合，如同您 已指定一空白副檔名般。

將 *DOS* 檔複製到基本作業系統檔案

使用 **dosread** 指令，可以將指定的 DOS 檔複製到指定的基本作業系統檔。

註： 使用 DOS 檔案命名慣例時會有一個例外。由於反斜線 (\) 字元對於基本作業系統有特殊意義，因此請使用斜線 (/) 字元作為定界字元，來指定 DOS 路徑名稱中的子目錄名稱。

請參閱下列範例：

· 若要將文字檔 chap1.doc 從 DOS 磁片複製到基本作業檔案系統，請鍵入：

```
dosread -a chap1.doc chap1
```

這樣會將 /dev/fd0 預設裝置上的文字檔 \CHAP1.DOC 複製到現行目錄中的基本作業系統檔 chap1。

· 若要將二進位檔從 DOS 磁片複製到基本作業檔案系統，請鍵入：

```
dosread -D/dev/fd0 /survey/test.dta /home/fran/testdata
```

這會將 /dev/fd0 上的 \SURVEY\TEST.DTA DOS 資料檔複製到基本作業系統檔 /home/fran/testdata。

請參閱 *Commands Reference, Volume 2* 中的 **dosread** 指令，以取得完整語法。

將基本作業系統檔複製到 *DOS* 檔案

使用 **doswrite** 指令，可以將指定的基本作業系統檔複製到指定的 DOS 檔。

註： 使用 DOS 檔案命名慣例時會有一個例外。由於反斜線 (\) 字元對於基本作業系統有特殊意義，因此請使用斜線 (/) 字元作為定界字元，來指定 DOS 路徑名稱中的子目錄名稱。

請參閱下列範例：

· 若要將文字檔 chap1 從基本作業檔案系統複製到 DOS 磁片，請鍵入：

```
doswrite -a chap1 chap1.doc
```

這會將現行目錄中的基本作業系統檔 chap1 複製到 /dev/fd0 上的 DOS 文字檔 \CHAP1.DOC。

· 若要將二進位檔 /survey/test.dta 從基本作業檔案系統複製到 DOS 磁片，請鍵入：

```
doswrite -D/dev/fd0 /home/fran/testdata /survey/test.dta
```

這會將基本作業系統資料檔 /home/fran/testdata 複製到 /dev/fd0 上的 DOS 檔 \SURVEY\TEST.DTA。

請參閱 *Commands Reference, Volume 2* 中的 **doswrite** 指令，以取得完整語法。

刪除 *DOS* 檔案

使用 **dosdel** 指令可以刪除指定的 DOS 檔案。

註： 使用 DOS 檔案命名慣例時會有一個例外。由於反斜線 (\) 字元對於基本作業系統有特殊意義，因此請使用斜線 (/) 字元作為定界字元，來指定 DOS 路徑名稱中的子目錄名稱。

dosdel 指令將在檢查磁碟機前，轉換檔案或目錄中的小寫字元為大寫字元。由於所有檔名都假設是完整的（不是相對的）路徑名稱，因此，前面不需要加入起始斜線 (/)。

例如，若要刪除預設裝置 (/dev/fd0) 上的 DOS 檔 file.ext，請鍵入：

```
dosdel file.ext
```

請參閱 *Commands Reference, Volume 2* 中的 **dosdel** 指令，以取得完整語法。

顯示 *DOS* 目錄的內容

使用 **dosdir** 指令可以顯示指定的 DOS 檔案或目錄的相關資訊。

註： 使用 DOS 檔案命名慣例時會有一個例外。由於反斜線 (\) 字元對於基本作業系統有特殊意義，因此請使用斜線 (/) 字元作為定界字元，來指定 DOS 路徑名稱中的子目錄名稱。

dosdir 指令將在檢查磁碟機前，轉換檔案或目錄名稱中的小寫字元為大寫字元。 由於所有檔名都假設為完整的（而非相對的）路徑名稱，因此，前面不需要加上 /（斜線）。

例如，若要讀取 /dev/fd0 上 DOS 檔案的目錄，請鍵入：

```
dosdir
```

該指令會將檔案名稱及磁碟空間資訊傳回，類似下列的情形。

```
PG3-25.TXT
PG4-25.TXT
PG5-25.TXT
PG6-25.TXT
Free space: 312320 bytes
```

請參閱 *Commands Reference, Volume 2* 中的 **dosdir** 指令，以取得完整語法。

檔案的指令摘要

下列是檔案、檔案處理程序及 DOS 檔案的指令。此外也有鏈結檔案與目錄的指令清單。

表 *57.* 檔案的指令

項目	說明
*	萬用字元，與任何字元相符
?	萬用字元，與任何單一字元相符
[ ]	Meta 字元，符合括住的字元。

表 *58.* 檔案處理程序的指令

項目	說明
cat	連接或顯示檔案
cmp	比較兩個檔案
colrm	自檔案中擷取直欄
cp	複製檔案
cut	從檔案中的每一行寫出所選取的位元組、字元或欄位
diff	比較文字檔
file	判斷檔案類型
find	尋找具有相符表示式的檔案
grep	搜尋檔案中的型樣
head	顯示檔案的前幾行或位元組數
more	在顯示畫面上顯示連續文字，一次一個畫面
mv	移動檔案
nl	將檔案中的字行編號
pg	將檔案格式化至顯示器
rm	移除（解除鏈結）檔案或目錄
paste	合併數個檔案中的字行，或是一個檔案中的後續字行。
sort	將檔案排序，合併已排序的檔案，並檢查檔案，以判斷其是否已排序
tail	將檔案寫入至標準輸出，從所指定的點開始
wc	計算檔案中的行數、字數及位元組數

表 59. 鏈結檔案及目錄的指令

項目	說明
ln	鏈結檔案及目錄

表 60. DOS 檔案的指令

項目	說明
dosdel	刪除 DOS 檔案
dosdir	列出 DOS 檔案的目錄
dosread	將 DOS 檔案複製到「基本作業系統」檔案中
doswrite	將「基本作業系統」檔案複製到 DOS 檔案中

作業系統 shell

您的作業系統介面稱為 *shell*。

shell 指作業系統的最外層次。shell 將程式設計語言納入控制處理程序與檔案中，以及啟動和控制其他程式。shell 藉由提示您輸入，為作業系統解譯該輸入，來管理您與作業系統之間的互動。

shell 提供讓您與作業系統通訊的方法。此通訊以互動方式實行（來自鍵盤的輸入會立即生效）或作為 shell script。 *shell script* 是一連串的 shell 和作業系統指令，儲存於檔案中。

當登入系統時，系統會尋找執行的 shell 程式名稱。一旦執行後，shell 就會顯示指令提示。此提示通常是一個 $（錢幣符號）。當在提示上鍵入指令並按 Enter 鍵時，shell 會評估指令並試著完成。根據指令指示，shell 會將指令輸出寫入畫面或重新導向輸出。然後，傳回指令提示並等待您鍵入其他指令。

指令行即您鍵入的該行。它包含 shell 提示。每一行的基本格式如下：

```
$ Command Argument(s)
```

shell 會將指令行的第一個字（直到第一個空格為止）當作指令，並將其後的所有字當作引數。

註： 將 `libc.a` 移動或重新命名時，會從 shell 中顯示 `Killed` 的錯誤訊息，這是因為沒有 `libc.a` 檔案可供系統載入及執行公用程式。**recsh** 指令會呼叫復原 shell，如果是不小心被移動，這會提供重新命名 `libc.a` 的功能。

相關工作

清單之前輸入的指令（history 指令）
使用 **history** 指令來列出您之前已輸入過的指令。

shell 概念

在您開始使用可用於 AIX 的不同類型的 shell 之前，您需要瞭解基本術語及特性。

可用的 *shell*

下列是 AIX 隨附的 shell。

- Korn shell（以 **ksh** 指令來啟動）
- Bourne shell（以 **bsh** 指令來啟動）
- 限制 shell（受限的 Bourne shell 版本，並以 **Rsh** 指令來啟動）
- POSIX shell（即所謂的 Korn shell，它是以 **psh** 指令啟動）
- Korn Shell（**ksh** 和 **ksh93**）的限制 Shell。 **ksh** 和 **ksh93** Shell 隨附了其限制 Shell 對等項目 **rksh** 和 **rksh93**。
- 預設 shell（以 **sh** 指令啟動）
- C shell（以 **csh** 指令啟動）
- 授信 shell (受限的 Korn shell 版本， 以 **tsh** 指令來啟動)

· 遠端 shell（以 **rsh** 指令來啟動）

登入 *shell* 是指當您登入電腦系統時，所載入的 shell。您的登入 shell 在 /etc/passwd 檔案中設定。Korn shell 是標準作業系統登入 shell，而且可與 Bourne shell 舊版相容。

Korn Shell (/usr/bin/ksh) 已設定為預設 Shell。預設或標準 Shell 指的是鏈結到 /usr/bin/sh 指令並以它為開頭的 Shell。可將 Bourne shell (/usr/bin/sh) 替代為預設 Shell。POSIX shell 是以 /usr/bin/psh 指令來呼叫的，它就好像通往 /usr/bin/sh 指令的鏈結一樣。

相關概念

Bourne Shell
Bourne shell 為一互動式指令直譯器及指令程式設計語言。使用

Korn shell 或 POSIX shell 指令
Korn shell 為互動式指令直譯器及指令程式語言。它符合「電腦環境可攜性作業系統介面 (POSIX)」，是國際標準的作業系統。

***shell* 術語**

表格中的術語及定義有助於瞭解 shell。

項目	說明
blank	空白是定義在 **LC_CTYPE** 種類中的空白字元類別內之其中一個字元。 在 POSIX shell 中，空白指標籤或空格。
內建指令	指 shell 所執行但沒搜尋它來建立個別處理的一種指令。
指令	shell 語言語法中的字元順序。shell 讀取每一個指令後，直接或藉由呼叫個別的公用程式來執行預期動作。
註解	以井字符號 (#) 為開頭的任何字組。其後的字組和所有字元，直到下一個換行字元，都會被忽略。
ID	以字母或底線為開頭，可攜性字元集中字母、數字或底線的順序。ID 第一個字元不可是數字。ID 使用為別名、函數及具名參數的名稱。
清單	一或多個管線的序列，管線之間是以下列其中一個符號來分隔： 分號 (;)、&符號 (&)、兩個 & 符號 (&&) 或兩條縱線 (||)。清單會選擇性地以下列其中一個符號來結束： 分號 (;)、& 符號 (&) 或縱線加 & 符號 (|&)。 **;** 循序處理前述管線。shell 依序執行每一個指令，並等待最新的指令完成。 **&** 非同步處理前述管線。shell 依序執行每一個指令，於背景處理管線但不等待其完成。 **|&** 非同步處理前述管線，並對上代 shell 建立雙向管線。shell 依序執行每一個指令，於背景處理管線但不等待其完成。母項 Shell 可讀取自及寫入到已建立的指令之標準輸入與輸出，方法是使用 **read -p** 及 **print -p** 指令。任何時候，僅有其中一個指令可在作用中。 **&&** 僅當之前管線傳回結束值零 (0) 時，才處理此符號之後的清單。 **||** 僅當之前管線傳回非零結束值時，才處理此符號之後的清單。 分號 (;)、&符號 (&) 及管線加 & 符號 (|&) 的優先順序比兩個 & 符號 (&&) 及兩條管線 (||) 來得低。;、& 及 |& 符號彼此間的優先順序相同。&& 及 || 符號的優先順序相同。一或多個換行字元可代替分號來區隔清單中兩個指令。 **註：** |& 符號僅適用於 Korn shell。

項目	說明
meta 字元	每一個 meta 字元對 shell 而言，都有一個特殊意義，若沒有用引號括住，則會導致字組終止。Meta 字元包括：管線 (|)、& 符號 (&)、分號 (;)、 小於符號 (<)、大於符號 (>)、左括弧 (()、右括弧 ())、錢幣符號 ($)、左引號 (`)、反斜線 (\)、右引號 (')、雙引號 (")、換行字元、空格字元及跳格字元。以單引號括住的所有字元，shell 視為引號並以文字型式解譯。如果未以引號括住，則保留 meta 字元的特殊意義。（Meta 字元亦即 C shell 中所謂的剖析器 *meta* 字元。）
參數指派清單	將 *Identifier=Value* 格式的一或數個字併入，其中位於等號 (=) 兩旁的空格必須平衡。亦即，必須使用前置及尾置空白，或無空白。 **註 :** 在 C shell 中，參數指派清單的格式為 **set***Identifier=Value*。等號 (=) 兩邊必須保留空格。
管線	由管線 (|) 隔開的一連串指令。管線中每一個指令（前一指令除外），都以個別的處理執行。然而，被管線連接的每一個指令之標準輸出依序變成下一指令的標準輸入。如果清單以括弧括住，則執行為個別 subshell 中操作的簡式指令。 如果管線前沒有保留字 !，則結束狀態將是指定在管線中最後一個指令的結束狀態。否則， 結束狀態是前一指令結束狀態的邏輯 NOT。換句話說，如果前一指令傳回零，則結束狀態將是 1。如果前一指令傳回大於零，則結束狀態將是 0。 管線的格式如下： `[!] command1 [ | command2 ...]` **註 :** 舊版的 Bourne shell 使用脫字符號 (^) 來指出管線。
shell 變數	名稱或參數的值已指派。輸入變數名稱、等號 (=) 及值來指派變數。可以在變數名稱之前加一個錢幣符號 ($)，即可將變數名稱由指派值來替換。變數特別有用於建立長路徑名稱的簡短表示法，例如 $HOME 代表起始目錄。 預先定義的變數其值已由 shell 指派。使用者定義的變數其值已由使用者指派。
simple 指令	按任何順序的一連串選用性參數指派清單與重新導向。它們其後選用性地跟著指令、字組及重新導向。它們是以;、|、&、||、&&、|& 或換行字元等終止。將指令名稱當成參數 0 來傳送（如 exec 子常式所定義）。若簡式指令正常終止，則其值是結束狀態 0，若是異常終止，則值為非零。sigaction、sigvec 或 signal 子常式包括信號結束狀態值的清單。
子 shell	當作登入 shell 或現行 shell 子項執行的一種 shell。
萬用字元	亦稱為型樣相符字元。shell 以指派值連結它們。基本萬用字元為 ?、*、[set] 及 [!set]。在執行檔名替代時， 萬用字元特別有用。
字組	不含任何空白的字元順序。字組以一或多個 meta 字元隔開。

指定 *Script* 檔的 *Shell*

當您在 Korn（即 POSIX Shell）或 Bourne Shell 中執行可執行檔 Shell script 時， 若沒有指定不同的 Shell，則 script 中的指令會在現行 Shell（即 Script 從其中啟動的 Shell）的控制下執行。當您在 C Shell 中執行可執行檔 Shell script 時，若沒有指定不同的 Shell，script 中的指令就會在 Bourne Shell (/usr/bin/bsh) 的控制下執行。

您可以將 Shell 併入 Shell Script 中，以在特定的 Shell 中執行 Shell script。

若要在特定的 Shell 下執行可執行檔 Shell script，請在 Shell script 的第一行鍵入 *#!Path*，然後按 Enter 鍵。此 *#!* 字元可識別檔案類型。*Path* 變數會指定要用來執行 Shell Script 的 Shell 路徑名稱。

例如，若要在 Bourne Shell 中執行 **bsh** Script，請鍵入：

```
#!/usr/bin/bsh
```

當您在 Shell Script 檔名之前加上 Shell 指令時，指定在指令行上的 Shell 會置換 Script 檔本身中所指定的 Shell。因此，鍵入 `ksh myfile` 並按 Enter 鍵，便會在 Korn Shell 控制下執行檔案 `myfile`，即使 `myfile` 的第一行是 `#!/usr/bin/csh` 也一樣。

shell 特性

使用 shell 作為系統介面有不少的優點。

透過 shell 來作為與系統之間的介面，其主要優點是：

- **檔名中的萬用字元替代（型樣相符）**

  指定符合的型樣（而不是指定實際檔名）來完成一組檔案的指令。

  如需相關資訊，請參閱：

  - 第 198 頁的『Korn shell 或 POSIX shell 中的檔名替代』
  - 第 227 頁的『Bourne shell 中的檔名替代』
  - 第 241 頁的『C shell 中的檔名替代』

- **背景處理程序**

  設定冗長作業於背景執行，釋放終端機來進行並行互動式處理。

  詳細資訊，請參閱下列的 **bg** 指令：

  - 第 212 頁的『Korn shell 或 POSIX shell 中的工作控制』
  - 第 248 頁的『C shell 內建指令』

    註： Bourne shell 不支援工作控制。

- **指令別名化**

  將別名指定給指令或詞組。當 shell 發現指令行或 shell script 中有別名時，它會將文字替換成別名參考。

  如需相關資訊，請參閱：

  - 第 223 頁的『Korn shell 或 POSIX shell 中的指令別名』
  - 第 239 頁的『C shell 中的別名替代』

    註： Bourne shell 不支援指令別名化。

- **指令歷程**

  將輸入的指令記錄於歷程檔中。您可以使用此檔案輕易地存取、修改及重新發出任何列出的指令。

  詳細資訊，請參閱下列的 **history** 指令：

  - 第 223 頁的『Korn shell 或 POSIX shell 指令歷程』
  - 第 248 頁的『C shell 內建指令』
  - 第 256 頁的『C shell 中的歷程替代』

    註： Bourne shell 不支援指令歷程。

- **檔名替代**

  在使用型樣相符字元的指令行上，自動產生檔名清單。

  如需相關資訊，請參閱：

  - 第 198 頁的『Korn shell 或 POSIX shell 中的檔名替代』
  - 第 227 頁的『Bourne shell 中的檔名替代』
  - 第 241 頁的『C shell 中的檔名替代』

- **輸入及輸出重新導向**

  將輸入重新導向遠離鍵盤，並將輸出重新導向終端機以外的檔案或裝置。例如，從檔案提供程式的輸入，然後重新導向印表機或另一個檔案。

  如需相關資訊，請參閱：

- 第 199 頁的『Korn shell 或 POSIX shell 中的輸入及輸出重新導向』
- 第 227 頁的『Bourne shell 中的輸入及輸出重新導向』
- 第 258 頁的『C shell 中的輸入及輸出重新導向』

· **管線**

鏈結任何數目的指令來形成一複雜程式。一個程式的標準輸出變成下一個程式的標準輸入。

如需相關資訊，請參閱 第 177 頁的『shell 術語』中的管線定義。

· **shell 變數替代**

將資料儲存於使用者定義的變數及預先定義的 shell 變數中。

如需相關資訊，請參閱：

- 第 196 頁的『Korn shell 或 POSIX shell 中的參數替代』
- 第 235 頁的『Bourne shell 中的變數替代』
- 第 240 頁的『C shell 中的變數替代』

相關概念

指令

某些指令只需簡單鍵入一個字組即可。也可能合併指令，使一指令的輸出成為另一個指令的輸入。

字元類別

您可以使用字元類別來比對檔名。

您可以使用字元類別來比對檔名，如下所示：

```
[[:charclass:]]
```

此格式會指示系統以符合屬於指定類別的任一個單一字元。所定義的類別對應於 `ctype` 子常式，如下所示：

字元類別	定義
alnum	英數字元
alpha	大寫與小寫字母
blank	空格或水平欄標
cntrl	控制字元
digit	數字
graph	圖形字元
lower	小寫字母
print	可列印字元
punct	標點字元
space	空格、水平欄標、換行字元、換行、垂直欄標或換頁字元
upper	大寫字元
xdigit	十六位數

限制 *shell*

限制 shell 是用來設定其功能比那些一般 Bourne shell 更受控制的執行環境與登入名稱。

Rsh 或 **bsh -r** 指令會開啟限制 shell。除了下列這些系統不接受的動作外，這些指令的行為與 **bsh** 指令的行為相同：

· 變更目錄（使用 **cd** 指令）

· 設定 *PATH* 或 *SHELL* 變數的值

· 指定包含斜線 (/) 的路徑或指令名稱

· 重新導向輸出

如果限制 shell 判斷所要執行的指令是 shell 程序，則它會使用 Bourne shell 來執行該指令。依此方法，當強制使用有限的指令功能表時，可提供使用者用來存取 Bourne shell 完整功能的 shell 程序。這種情況會假設使用者在同一目錄中沒有寫入及執行許可權。

如果在啟動 Bourne shell 時指定了 *File* [*Parameter*] 參數，則 shell 會執行 *File* 參數（包括任何指定的參數）識別的 Script 檔。指定的 script 檔必須具備讀取許可權。Script 檔的任何 **setuid** 及 **setgid** 設定都會被忽略。然後，shell 讀取指令。若使用了 **-c** 或 **-s** 旗標，請不要指定 Script 檔。

當使用 **Rsh** 指令來啟動時，在解譯 `.profile` 及 `/etc/environment` 檔案之後，shell 會施行限制。因此，藉由執行設定動作及保存使用者於適當目錄（可能不是登入目錄）中，`.profile` 檔案的寫出器具有使用者動作的完全控制權。管理者可以在 **Rsh** 指令可使用的 `/usr/rbin` 目錄中，建立一個指令目錄，並變更 *PATH* 變數來包含該目錄。如果是以 **bsh -r** 指令來啟動它，則在解譯 `.profile` 檔案時，shell 會引用限制。

當使用名稱 **Rsh** 來呼叫時， 限制 shell 會讀取使用者的 `.profile` 檔案 (`$HOME/.profile`)。此時，它會扮演一般 Bourne shell，但若岔斷，則會導致立即跳出而不會返回指令層次。

Korn Shell 可以使用指令 **ksh -r** 來啟動為限制 Shell。

ksh 和 **rksh** 的 inode 相同，而 **ksh93** 和 **rksh93** 的 inode 相同。

建立及執行 *shell script*

shell script 是一種包含一或多個指令的檔案。shell script 提供一個簡單的方法，讓您執行冗長單調的指令、大量或複雜的指令順序，以及例行作業。當您輸入 shell script 檔的名稱時，系統會執行檔案所包含的指令序列。

您可以使用文字編輯器來建立 shell script。您的 script 可包含作業系統指令與 shell 內建指令。

下列步驟是撰寫 shell script 的一般指引：

1. 使用文字編輯器，建立及儲存檔案。您可以將 shell 與作業系統指令的任何組合併入 shell script 檔中。依照慣例，尚未設定給許多使用者使用的 shell script，會儲存於 `$HOME/bin` 目錄中。

   註： 作業系統不支援 Shell Script 內的 `setuid` 或 `setgid` 子常式。
2. 使用 **chmod** 指令，只允許擁有者執行檔案。例如，如果您的檔案名稱是 `script1`，請鍵入：

   ```
   chmod u=rwx script1
   ```
3. 在指令行輸入 script 名稱來執行 shell script。若要執行 `script1` shell script，請鍵入：

   ```
   script1
   ```

   註： 如果指令行上將 shell 指令（**ksh**、**bsh** 或 **csh**）置於 shell script 檔名之前，則不需要將 shell script 建立成可執行檔，也可以執行它。例如，若要在 Korn shell 中執行非執行檔 `script1`，請鍵入下列指令：

   ```
   ksh script1
   ```

相關概念

指令

某些指令只需簡單鍵入一個字組即可。也可能合併指令，使一指令的輸出成為另一個指令的輸入。

Korn shell

Korn shell（ksh 指令）與 Bourne shell（bsh 指令）的舊版相容，而且包含 Bourne shell 大部分的特性，以及 C shell 的數個良好特性。

Korn shell 或 *POSIX shell* 所設定的變數

下列是由 shell 設定的變數。

項目	說明
underscore (_)	初始指出在環境中執行傳送的 shell 或 Script 的絕對路徑名稱。後來，它被指派上一指令的最後一個引數。此參數不設定給非同步指令。 該參數亦可以用來保留檢查郵件時所符合的 `MAIL` 檔名。
ERRNO	指定最近失敗的子常式所設定的值。此值依系統而定，並且是為除錯目的而設計的。
LINENO	指定執行中 Script 或函數內現行行的行號。
OLDPWD	指出 **cd** 指令所設定的上一個工作目錄。
OPTARG	指定 **getopts** 一般內建指令所處理的最後一個選項引數的值。
OPTIND	指定 **getopts** 一般內建指令所處理的最後一個選項引數的索引。
PPID	定義 shell 母項的程序號碼。
PWD	指出 **cd** 指令所設定的上一個工作目錄。
RANDOM	產生隨機整數，平均地分散於 0 和 32767 之間。藉由指派一個數值給 *RANDOM* 變數，可以起始設定亂數的順序。
REPLY	當未提供任何引數時，由 **select** 陳述式及 **read** 一般內建指令設定。
SECONDS	指定自返回 shell 呼叫之後的秒數。如果已指派一個值給此變數，則根據參考而傳回的值，是已指派的值再加上指派之後經過的秒數。

***Korn shell* 或 *POSIX shell* 所使用的變數**

下列是由 shell 使用的變數。

項目	說明
CDPATH	指出 **cd**（變更目錄）指令的搜尋路徑。
COLUMNS	定義 shell 編輯模式及列印 **select** 清單的編輯視窗寬度。
EDITOR	如果此參數值以 emacs、gmacs 或 vi 結束，且未以 **set** 特殊內建指令設定 *VISUAL* 變數，則會開啟相對應的選項。
ENV	如果設定此變數，則會在該值上執行參數替代，以產生 shell 被呼叫時所執行之 Script 的路徑名稱。一般而言，此檔案使用於別名和函數定義。此旗標將會是被忽略的非互動式 shell。
FCEDIT	指定 **fc** 一般內建指令的預設編輯器名稱。
FPATH	指定函數定義的搜尋路徑。當具有 **-u** 旗標的函數被參考到，並且找不到指令時，則會搜尋此路徑。如果找到可執行檔，則會讀取它並且在現行環境中執行。
HISTFILE	如果當呼叫 shell 時設定此變數，則值是儲存指令歷程的檔案之路徑名稱。 `history` 檔的起始設定處理程序可視系統啟動檔而定，因為部分啟動檔包含可有效地先取得使用者為 HISTFILE 及 HISTSIZE 所指定的設定值。例如，函數定義指令會記錄在 `history` 檔案中。如果系統管理者在系統啟動檔中包含了函數定義，並且該啟動檔會在 ENV 檔之前或 *HISTFILE* 之前呼叫，或有設定 *HISTSIZE* 變數時，則會在使用者可以影響其性質之前，起始設定 `history` 檔案。
HISTSIZE	如果當呼叫 shell 時設定此變數，則先前輸入的指令（由此 shell 存取），其數目將大於或等於此數目。若是非 root 使用者，預設值為 128 個指令，若是 root 使用者，則為 512 個指令。
HOME	指出您的登入目錄名稱，該名稱成為登入完成時的現行目錄。**login** 程式會起始設定此變數。**cd** 指令會使用 *$HOME* 參數的值作為其預設值。在 shell 程序中使用此變數而非明確的路徑名稱，可不需切換目錄即可從另一個目錄執行程序。

項目	說明
IFS	指定 IFS（內部欄位分隔字元，通常是空格、跳格字元及換行字元），可用來它來隔開指令或參數替代所產生的指令字組，以及隔開具有一般內建指令 **read** 的字組。*IFS* 參數的第一個字元是用來隔開 $* 替代的引數。
LANG	提供 *LC_** 變數的預設值。
LC_ALL	置換 *LANG* 及 *LC_** 變數的值。
LC_COLLATE	決定型樣相符內範圍表示式的行為。
LC_CTYPE	定義字元分類、大小寫轉換、以及其他字元屬性。
LC_MESSAGES	決定撰寫訊息的語言。
LINES	決定列印選取清單所用的直欄長度。選取清單會垂直列印， 直到 *LINES* 變數所指定的行數被填滿三分之二為止。
MAIL	指定郵件系統為了偵測到達的新郵件所用的檔案路徑名稱。如果此變數設定為郵件的名稱，且未設定 *MAILPATH* 變數，則 shell 會向使用者通知指定檔中的新郵件。
MAILCHECK	指定 shell 多久檢查一次變更（秒），該變更是指 *MAILPATH* 或 *MAIL* 變數所指定之任何檔案在修改時的變更。預設值是 600 秒。當時間流逝，shell 會先檢查它之後再發出下一步提示。
MAILPATH	以冒號隔開來指定一連串檔名。如果已設定此變數，則 shell 會通知使用者在 *MAILCHECK* 變數指定的期間內（以秒為單位），指定的檔案已被修改。每一個檔名其後可能跟著一個 **?**， 並將列印出訊息。此訊息會進行變數替代，將 *$_* 變數定義為已變更的檔名。 預設訊息是 `you have mail in $_`。
NLSPATH	決定訊息型錄的位置，以處理 **LC_MESSAGES**。
PATH	指出指令的搜尋路徑，是以冒號隔開的目錄路徑名稱之依序清單。當 shell 尋找指令時，會依指定順序來搜尋這些目錄。清單中的空字串代表現行目錄。
PS1	指定用來作為主要系統提示的字串。參數的值會經由參數替代而擴充，以定義主要提示字串，依預設值而言是一個 $。! 字元在主要提示字串中，會取代為指令號碼。
PS2	指定第二個提示字串的值，根據預設值，這是 >。
PS3	指定 **select** 迴圈中使用的選項提示字串的值，根據預設值，它是 *#*?。
PS4	此變數的值會經由參數替代而擴充，並前置於每一行執行追蹤前面。如果省略，則執行追蹤提示是一個 +。
SHELL	指定 shell 的路徑名稱，保存於環境中。
SHELL PROMPT	當使用互動式時，在讀取某個指令之前 shell 會以 *PS1* 參數值提示。如果在任何時候輸入新的一行，且 shell 進一步要求輸入完整指令，則 shell 會發出第二次的提示（*PS1* 參數值）。
TMOUT	指定 shell 在結束之前，等待非作用中狀態的秒數。如果 *TMOUT* 變數設成一個大於零 (0) 的值，則在發出 *PS1* 提示之後，於規定的秒數之內未輸入指令的話， shell 將會結束。（請注意，可使用不超出此值的最大界限來編譯 shell。） **註 :** 在超過逾時期限之後，shell 在結束之前有 60 秒的暫停時間。
VISUAL	如果此參數的值以 emacs、gmacs 或 vi 結束，則會開啟相對應選項。

shell 會提供預設值給 *PATH*、*PS1*、*PS2*、*MAILCHECK*、*TMOUT* 及 *IFS* 參數，但是 *HOME*、*SHELL*、*ENV* 及 *MAIL* 等參數並不是由 shell 設定（雖然 *HOME* 參數是由 **login** 指令設定）。

Korn Shell 或 *POSIX Shell* 中的指令替代

Korn shell 或 POSIX shell 可讓您執行指令替代。指令替代中，shell 會在 subshell 環境中執行一指定指令，且以其輸出值取代該指令。

若要在 Korn shell 或 POSIX shell 中執行指令替代，請鍵入：

```
$(command)
```

或，若是英文反引號 (backquoted) 版本，請鍵入：

```
`command`
```

註： 雖然 **ksh** 可接受向後引用語法，但是 X/Open Portability Guide Issue 4 及 POSIX 標準已將其視為作廢的語法。這些標準將建議可攜性應用程式使用$ (指令) 語法。

shell 會在 subshell 環境中執行指令，以指令的標準輸出來取代指令替代（指令的文字加上 $( ) 或左引號），並移除替代文字結尾的一或數個換行字元，藉以擴充指令替代。

在下列範例中，包圍住指令的 $( ) 會指出要替代的 **whoami** 指令的輸出：

```
echo My name is: $(whoami)
```

您可以執行相同的指令替代：

```
echo My name is: `whoami`
```

上述兩個列中的使用者輸出為 dee：

```
My name is: dee
```

您也可以將算術運算式包括在 ( ) 中，即可替代它們。 例如，指令：

```
echo Each hour contains $((60 * 60)) seconds
```

將產生下列結果：

```
Each hour contains 3600 seconds
```

Korn shell 或 POSIX shell 會在執行指令替代時，移除尾端所有的換行字元。例如，如果您的現行目錄包含 file1 與 file2， 與 file3 檔案，指令：

```
echo $(ls)
```

將移除換行字元，並產生下列輸出：

```
file1 file2 file3
```

若要保留換行字元，請在 " " 中插入所替代的指令：

```
echo "$(ls)"
```

Korn shell 或 *POSIX shell* 中的算術評估

Korn shell 或 POSIX shell 一般內建 **let** 指令可讓您執行整數算術。

常數格式為 [**Base**]**Number**。**Base** 參數為在 2 和 36（含）之間的十進位數，代表算術的基底。**Number** 參數為該基底下的數字。如果您省略 **Base** 參數，則 shell 會使用 10 做基底。

算術運算式使用與 C 程式設計語言相同的語法、優先順序及表示式相關性。除雙加號 (++)、雙連字號 (—)、問號加冒號 (?:) 及逗號 (,) 之外，其他所有整數運算子都支援。下列表格以遞減的優先順序，列出有效的 Korn shell 或 POSIX shell 運算子：

運算子	定義
-	負單運算元
!	邏輯否定
~	按位元否定
*	乘法
/	除法
%	餘數
+	加法
-	減法
<<, >>	左移，右移
<=,>=, <>, ==, !=	比較
&	按位元 AND
^	按位元互斥 OR
|	按位元 OR
&&	邏輯 AND
||	邏輯 OR
= *=、/=、&= +=、-=、<<=、> >=、&=、^=、|=	指派

許多算術運算子（例如 *、&、< 及 >）對 Korn shell 或 POSIX shell 具有特殊的意義。這些字元必須要用引號含括。例如，若要將現行值 *y* 乘以 5，再將新值重新指派為 *y*，請使用表示式：

```
let "y = y * 5"
```

將表示式含括在引號內，會移除 * 字元的特殊意義。

您可以將運算子分組在 **let** 指令表示式中，以強迫分組。例如，在表示式：

```
let "z = q * (z - 10)"
```

此指令將 *q* 乘以 *z* 的減少值。

如果僅評估一個單一表示式，則 Korn shell 或 POSIX shell 會包括 **let** 指令的備用格式。shell 會將括在 (( )) 中的指令當作引號的表示式。因此，表示式：

```
((x = x / 3))
```

等於：

```
let "x = x / 3"
```

可在算術運算中以名稱查詢到具名參數，而不須使用參數替代語法。當查詢出具名參數時，它的值被評估為算術運算。

使用 **typeset** 特殊內建指令的 **-i** 旗標，可以指定具名參數的內部整數表示法。藉由使用 **-i** 標示，可在每一指派為具名參數的值上執行算術評估。如果您不指定算術基底，則參數的第一個指派決定算術基底。當參數替代發生時使用此基底。

相關概念

Korn shell 或 POSIX shell 指令

Korn shell 為互動式指令直譯器及指令程式語言。它符合「電腦環境可攜性作業系統介面 (POSIX)」，是國際標準的作業系統。

Korn shell 中的參數
底下將說明 Korn shell 參數。

Korn shell 或 *POSIX shell* 中的欄位分割
執行替代指令後，Korn shell 會尋找那些在 **IFS**（內部欄位分隔頁）變數中的欄位分隔字元，掃描出替代的結果。找到這種字元的位置所在，shell 會把替代分割成不同的引數。

shell 也會保留明確的空值（"" 或 ''），並且會移除隱含的空值引數（那些空值引數是由沒有值的引數所造成的）。

- 如果 *IFS* 的值是空格、跳格字元或換行字元，或是沒有設定，則位於輸入開頭或結尾的任何空格、跳格字元或換行字元序列都會被忽略，並且那些字元在輸入中的任何序列都會成為欄位區隔。例如，下列的輸入值會產生兩個欄位 **--** **school** 和 **days**：

  ```
  <newline><space><tab>school<tab><tab>days<space>
  ```

- 否則，如果 **IFS** 不是空值，則會依序套用下列規則。***IFS*** 空格是用來表示 **IFS** 值中任何序列的空格（零或多個出現次數）（例如，若 **IFS** 包含空格/逗點/跳格字元，任何序列的空格及跳格字元都會被視為 **IFS** 空格）。

  1. **IFS** 空格如果是在輸入值的開頭和尾端，就不會被系統處理。
  2. 輸入中出現的每個 **IFS** 字元若不是 **IFS** 空格，並緊鄰任何 **IFS** 空格，就會作為欄位區隔。
  3. 長度不是零的 **IFS** 空格會作為欄位區隔。

Korn shell 或 *POSIX shell* 特殊內建指令的清單
特殊指令內建在 Korn shell 及 POSIX shell 中，並且在 shell 處理程序中執行。

項目	說明
:（冒號）	只展開引數。
.（點）開頭。	讀取指定的檔案，然後執行指令。
break	自含括的 **for**、**while**、**until** 或 **select** 迴圈中結束。
continue	回復含括的 **for**、**while**、**until** 或 **select** 迴圈的下一個疊代。
eval	讀取引數作為 shell 的輸入，然後執行結果指令。
exec	執行 *Argument* 參數所指定的指令，而非此 shell 所指定的指令，但不建立新的處理程序。
exit	結束由 *n* 參數來指定結束狀態的 shell。
export	標記名稱以便自動匯出至後續執行指令的環境。
newgrp	相當於 exec/usr/bin/newgrp [*Group* ...] 指令。
readonly	標記指定的名稱為唯讀。
return	導致 shell 返回呼叫的 Script。
set	除非已指定選項或引數，否則以現行語言環境的對照順序來寫入全部 shell 的名稱和值。
shift	將位置參數更名。
times	針對 shell 以及自該 shell 執行的處理程序，列印累計的使用者和系統時間。
trap	當 shell 收到指定的信號時，執行指定的指令。
typeset	會設定 shell 參數的屬性和值。
unset	取消設定指定參數的值和屬性。

相關概念
Korn shell 或 POSIX shell 內建指令
內建在 Korn shell 和 POSIX shell 中的特殊指令並且在 shell 處理程序中執行。

Korn shell 或 *POSIX shell* 一般內建指令
下列是 Korn shell 或 POSIX shell 一般內建指令的清單。

項目	說明
alias	在標準輸出上列印一個別名清單。
bg	將指定的工作置於背景。
cd	將現行目錄改變成指定的目錄，或將現行字串替換成指定的字串。
echo	將字元字串寫入標準輸出。
fc	自最後在終端機上鍵入的 *HISTSIZE* 變數指令中選取一個指令範圍。在執行新舊替代之後，重新執行指定的指令。
fg	將指定的工作帶至前景。
getopts	檢查合法選項的 *Argument* 參數。
jobs	列出指定工作的資訊。
kill	傳送 **TERM**（終止）信號給指定的工作或處理程序。
let	評估指定的算術運算。
print	會將 shell 列印輸出。
pwd	相當於 **print -r -$PWD** 指令。
read	採取 shell 輸入。
ulimit	設定或顯示使用者處理程序資源限制，如同 /etc/security/limits 檔案中定義的一樣。
umask	決定檔案的許可權。
unalias	從別名清單中移除名稱清單的參數。
wait	等待指定的工作並且終止。
whence	指出若每一個指定的名稱被用來作為指令名稱，則該如何解譯。

如需相關資訊，請參閱 第 201 頁的『Korn shell 或 POSIX shell 內建指令』。

相關概念
Korn shell 或 POSIX shell 內建指令
內建在 Korn shell 和 POSIX shell 中的特殊指令並且在 shell 處理程序中執行。

Korn shell 或 *POSIX shell* 的條件表示式
條件表示式是與 [[的複合指令一起使用，以測試檔 案屬性與比較字串。

將不會執行出現在 [[及]] 之間的字組分割與檔名替代。每一表示式是由下列一或多個單運算元 或二元運算子表示式所建構的：

項目	說明
-a *File*	如果指定的檔案為指向另一實際存在檔案的符號鏈結，則為真。
-b *File*	如果指定的檔案存在，且為一區塊特殊檔案，則為真。
-c *File*	如果指定的檔案存在，且為一字元特殊檔案，則為真。
-d *File*	如果指定的檔案存在，且為一目錄，則為真。
-e *File*	如果指定的檔案存在，則為真。

項目	說明
-f *File*	如果指定的檔案存在，且為一般檔案，則為真。
-g *File*	如果指定的檔案存在，且已設定 **setgid** 位元，則為真。
-h *File*	如果指定的檔案存在，且為一符號鏈結，則為真。
-k *File*	如果指定的檔案存在，且已設定選定位元，則為真。
-n *String*	如果指定字串長度不是零值，則為真。
-o *Option*	如果已選取指定選項，則為真。
-p *File*	如果指定的檔案存在，且為一先進先出法 (FIFO) 特殊檔案或管線，則為真。
-r *File*	如果指定的檔案存在，且可由現行處理程序讀取，則為真。
-s *File*	如果指定的檔案存在，且大小大於零值，則為真。
-t *FileDescriptor*	如果已開啟指定的檔案描述子，且已連結終端機裝置，則為真。
-u *File*	如果指定的檔案存在，且已設定它的 **setuid** 位元，則為真。
-w *File*	如果指定的檔案存在，且已開啟寫入位元，則為真。然而，即使此測 試指出為真，檔案仍無法改寫在一唯讀檔案系統上。
-x *File*	如果指定的檔案存在，且已開啟**執行**旗標，則為真。如果指定的檔案 存在，且為一目錄，則現行處理程序許可在目錄中搜尋。
-z *String*	如果指定的字串長度為零值，則為真。
-L *File*	如果指定的檔案存在，且為一符號鏈結，則為真。
-O *File*	如果指定的檔案存在，且為此處理程序的有效使用者 ID 所擁有，則為真。
-G *File*	如果指定的檔案存在，且其群組符合此處理程序的有效群組 ID，則為真。
-S *File*	如果指定的檔案存在，且為一 Socket，則為真。
File1 **-nt** *File2*	如果 *File1* 存在，且比 *File2* 新，則為真。
File1 **-ot** *File2*	如果 *File1* 存在，且比 *File2* 舊，則為真。
File1 **-ef** *File2*	如果 *File1* 與 *File2* 存在，且參照相同檔案，則為真。
String1 **=** *String2*	如果 *String1* 等於 *String2*，則為真。
String1 **!=** *String2*	如果 *String1* 不等於 *String2*，則為真。
String **=** *Pattern*	如果指定的字串與指定的型樣相符，則為真。
String **!=** *Pattern*	如果指定的字串與指定的型樣不相符，則為真。
String1 **<** *String2*	如果根據美國國家標準交換碼 (ASCII) 數值的字元，字串 *1* 在字串 *2* 之前，則為真。
String1 **>** *String2*	如果根據美國國家標準交換碼 (ASCII) 數值的字元，字串 *1* 在字串 *2* 之後，則為真。
Expression1 **-eq** *Expression2*	如果表示式 *1* 等於表示式 *2*，則為真。
Expression1 **-ne** *Expression2*	如果表示式 *1* 不等於表示式 *2*，則為真。
Expression1 **-lt** *Expression2*	如果表示式 *1* 小於表示式 *2*，則為真。
Expression1 **-gt** *Expression2*	如果表示式 *1* 大於表示式 *2*，則為真。
Expression1 **-le** *Expression2*	如果表示式 *1* 小於或等於表示式 *2*，則為真。

項目	說明
Expression1 **-ge** *Expression2*	如果表示式 *1* 大於或等於表示式 *2*，則為真。

註 : 在先前的每一表示式中，如果 *File* 變數類似 /dev/fd/*n*，其中的 *n* 為整數，則測試可適用於其描述子號碼為 *n* 的開放檔案。

您可使用下列以遞減順序清單的任何表示式，從這些初始值或較小的部分來建構複合表示式。

項目	說明
(*Expression***)**	如果指定的表示式為真，則為真。用以組合表示式。
! *Expression*	如果指定的表示式為假，則為真。
Expression1 **&&** *Expression2*	如果表示式 *1* 及表示式 *2* 都為真，則為真。
Expression1 **||** *Expression2*	如果表示式 *1* 或表示式 *2* 其中之一為真，則為真。

在 *Korn shell* 或 *POSIX shell* 中以引號括住字元

當您希望 Korn shell 或 POSIX shell 將字元當作一般字元讀取，而非具有相關意義，則您必須以引號來括住它。

每一個 meta 字元對 shell 而言，都有一個特殊意義，若沒有以引號括住，則會導致字組終止。下列字元被 Korn shell 或 POSIX shell 視為 meta 字元，如果它們所代表的是自己，則必須以引號括住：

- 管線 (|)
- & 符號 (&)
- 分號 (;)
- 小於符號 (<) 及大於符號 (>)
- 左括弧 (() 及右括弧 ())
- 錢幣符號 ($)
- 左引號 (`) 及單引號 (')
- 反斜線 (\)
- 雙引號 (")
- 換行字元
- 空格字元
- 跳格字元

若要否定 meta 字元的特殊意義，請使用下列清單中其中一個引號機制。

項目	說明
反斜線	未以引號括住的反斜線 (\) 將會保留下列字元的文字值， 但換行字元除外。如果反斜線後緊接著換行字元，則 shell 會將此解譯為接續行。
單引號	以單引號 (' ') 括住字元，保留單引號內每一個字元的文字值。單獨的單引號無法存在於數個單引號中。 反斜線不可用來跳出設定在單引號中之字串的單引號。內含的引號可藉由撰寫建立， 例如：'a'\''b'，這會產生 a'b。

項目	說明
雙引號	以雙引號 (" ") 含括字元可保留雙引號內所有字元的文字值， 但錢幣符號、左引號及反斜線字元除外，如下所示： **$** 錢幣符號保留其特殊意義，引導參數展開、指令替代格式及算術展開。 引號內的字串之中的輸入字元同時含括在 $(及符合的) 之間，這些字元將不會受到雙引號的影響，但展開字詞時，定義其輸出會置換 $(...) 的指令。 在括住的 ${ 與符合的 } 之間的字元字串內，必須是未跳出雙引號或單引號的偶數（如果有的話）。前置反斜線字元必須用來跳出文字 { 或 }。 **`** 左引號保留其特殊意義，引導其他指令替代格式。從初始的左引號開始，一直到沒有前置反斜線的下一個左引號為止的字元，其中所含括的字串部分可將指令定義成：當字組展開時，以該指令的輸出來置換 ` ... ` 。 **** 只有當反斜線後面接著下列其中一個字元時，才會保留其為跳出字元的特殊意義：$、`、"、\ 或換行字元。

雙引號之前必須有一條反斜線，才能括在雙引號之內。當您使用雙引號時，如果反斜線後面緊接被解譯成有特殊意義的字元，則反斜線會被刪除且後續字元採文字型式。如果反斜線後面沒有具特殊意義的字元，則它會留置在原來的地方，並且緊接在後的字元也會留置原處。例如：

```
"\$"    ->      $
"\a"    ->      \a
```

下列條件適用 Korn 或 POSIX shell 中的 meta 字元及引號字元：

- 錢幣符號加星號 ($*) 和錢幣符號加 @ 符號 ($@) 若沒有用引號括住，或當作參數指派值或檔名來使用時，其意義皆相同。
- 作為指令引數來使用時，雙引號加錢幣符號加星號加雙引號 ("$*") 相當於 "$1*d*$2*d*..."，其中 *d* 是 IFS 參數的第一個字元。
- 雙引號加 @ 符號加雙引號 ("$@") 相當於 "$1" "$2" ...。
- 在一組左引號中 (` `)，以反斜線來引導下列字元：反斜線 (\)、 單引號 (') 及錢幣符號 ($)。如果左引號出現在雙引號 (" ") 內，則反斜線也會括住雙引號字元。
- 參數及指令替代會發生在雙引號 (" ") 內。
- 用引號括住保留字的任何字元，以移除保留字或別名的特殊意義。您不能以引號括住函數名稱或內建指令名稱。

受限制的 *Korn shell*

受限制的 Korn shell 可用來設定登入名稱與執行環境， 其功能比那些一般 Korn shell 的功能更加受到控制。

rksh 或 **ksh -r** 指令會開啟「受限制的 Korn shell」。這些指令的行為與 **ksh** 指令的行為相同，除了下列這些不接受的動作之外：

- 變更現行工作目錄
- 設定 *SHELL*、*ENV* 或 *PATH* 變數的值
- 指定包含 /（斜線）的指令之路徑名稱
- 以 >（右脫字符號）、>|（右脫字符號加管線符號）、 <>（左脫字符號加右脫字符號）或 >>（兩個右脫字符號）將指令的輸出重新導向

如果受限制的 Korn shell 判斷要執行的指令是 shell 程序，則它會使用 Korn shell 來執行該指令。依此方法，當強制使用有限的指令功能表時，可提供終端使用者用來存取 Korn shell 完整功能的 shell 程序。這種情況會假設使用者在同一目錄中沒有寫入及執行許可權。

如果在啟動 Korn shell 時指定了 **File** [*Parameter*] 參數，則 shell 會執行由 **File** 參數（包括任何指定的參數）識別的 Script 檔。指定的 script 檔必須具備讀取許可權。Script 檔的任何 `setuid` 及 `setgid` 設定都會被忽略。然後，shell 讀取指令。若使用了 **-c** 或 **-s** 旗標，請不要指定 Script 檔。

當使用 **rksh** 指令來啟動時，在解譯 `.profile` 及 `/etc/environment` 檔案之後，shell 會實行限制。因此，藉由執行設定動作及保存使用者於適當目錄（可能不是登入目錄）中，`.profile` 檔案的寫出器具有使用者動作的完全控制權。管理者可以在 **rksh** 指令可以使用的 `/usr/rbin` 目錄中建立一個指令目錄，並變更 *PATH* 變數來包含該目錄。如果是以 **ksh -r** 指令來啟動它，則在解譯 `.profile` 檔案時，shell 會引用限制。

當使用 **rksh** 指令來呼叫時，「受限制的 Korn shell」會讀取使用者的 `.profile` 檔案 (`$HOME/.profile`)。進行此動作時，它當成一般 Korn shell 來運作， 但若岔斷，則會導致立即跳出而不會返回指令層次。

Korn shell 或 *POSIX shell* 中的保留字

下列保留字對 Korn shell 或 POSIX shell 具有特殊的意義。

```
!               case    do
done    elif    else
esac    fi      for
function  if    in
select  then    time
until   while   {
}       [[      ]]
```

只有當保留字不帶引號，且以下列方式來使用時，才會辨識保留字：

- 指令的第一個字
- 不是 **case**、**for** 或 **in** 的其中一個保留字之後的第一個字
- **case** 或 **for** 指令中的第三個字（在此情況下，只有 **in** 有效）

強化的 *Korn shell (ksh93)*

除了預設系統 Korn shell (`/usr/bin/ksh`) 外， AIX 還以 `/usr/bin/ksh93` 提供一個強化的版本。此強化版本與現行預設版本大部分具有向上相容性，並包含一些無法在 `/usr/bin/ksh` 中使用的其他特性。

部分 Script 在 Korn shell `ksh93` 底下執行時，可能會和在預設的 shell 下執行時略有不同，這是因為兩種 shell 處理變數的方式多少會有點不同。

註 : 也可以使用一個名稱為 `rksh93` 且有受到限制的已強化 Korn shell 版本。

下列特性無法在 Korn shell `/usr/bin/ksh` 中使用， 但是可以在 Korn shell `/usr/bin/ksh93` 中使用：

項目	說明
算術加強功能	您可以在算術運算式中使用 **libm** 函數（可在 C 程式設計語言中找到的典型算術函數），例如 `$ value=$((sqrt(9)))`。此外還有其他算術運算子可以使用，包括單運算元 **+**、**++**、**--** 以及 **?:** 建構 (例如，"`x ? y : z`")，以及 **,**（逗點）運算子。最多可支援以 64 為底數的算術基底。同時亦支援 浮點算術。"**typeset -E**"（指數）可用來指定有效位數，而 "**typeset -F**"（浮點）則可用來指定算術變數的小數位數。**SECONDS** 變數現在顯示接近於 百分之一秒，而不是接近一秒。

複合變數	支援複合變數。複合變數可讓使用者在單一變數名稱中指定多重值。會為這裡的每一個值指派一個註標變數，並以句點 (.) 與母項變數隔開。例如：
	`$ myvar=( x=1 y=2 )`
	`$ print "${myvar.x}"`
	`1`

項目	說明

複合指派	起始設定陣列時支援複合指派，包含索引陣列及聯合陣列。指派值會以括弧括住，如下列範例所示：
	`$ numbers=( zero one two three )`
	`$ print ${numbers[0]} ${numbers[3]}`
	`zero three`

聯合陣列	聯合陣列是含有一個作為索引的字串的陣列。
	在 **typeset** 指令使用 **-A** 旗標時，可讓您在 ksh93 中指定聯合陣列。例如：
	`$ typeset -A teammates`
	`$ teammates=( [john]=smith [mary]=jones )`
	`$ print ${teammates[mary]}`
	`jones`

變數名稱參照	在 **typeset** 指令中使用 **-n** 旗標時，可讓您將一個變數名稱指派成另一個變數的參考資料。依此方法來修改變數值，將會依次修改所參考到的變數值。例如：
	`$ greeting="hello"`
	`$ typeset -n welcome=greeting     # establishes the reference`
	`$ welcome="hi there"     # overrides previous value`
	`$ print $greeting`
	`hi there`

參數擴充	可使用的參數擴充建構如下： · ${!varname} 是變數本身的名稱。 · ${!*varname*[@]} 會命名 *varname* 陣列的索引。 · ${*param*:*offset*} 是 *param* 的子字串，從 *offset* 開始。 · ${param:offset:num} 是 *param* 的子字串，開始於 *offset*，是 *num* 號碼的字元。 · ${@:*offset*} 指出以 *offset* 開頭的所有位置參數。 · ${@:*offset*:*num*} 指出從 *offset* 開始的位置參數 *num*。 · ${*param*/*pattern*/*repl*} 相當於 *param*，但它的第一個 *pattern* 必須以 *repl* 取代。 · ${*param*//*pattern*/*repl*} 相當於 *param*，但它的每一個 *pattern* 都必須以 *repl* 取代。 · ${*param*/#*pattern*/*repl*} 若 *param* 以 *pattern* 開頭，則 *param* 必須以 *repl* 取代。 · ${*param*/%*pattern*/*repl*} 如果 *param* 以 *pattern* 結尾，則 *param* 會換成 *repl*。

項目	說明

紀律函數	紀律函數是與特定變數相關的函數。這可讓您每次在參考、設定或取消設定該變數時，定義和呼叫函數。這些函數的格式為 *varname.function*，其中的 *varname* 是變數名稱，而 *function* 是紀律函數。預先定義的紀律函數有：**get**、**set** 及 **unset**。
	· 每次參照 *varname* 時，即會呼叫 ***varname*.get** 函數。如果在此函數中設定特殊變數 **.sh.value**，則 *varname* 值就會變為此值。日期時間即為一個簡單的範例：
	`$ function time.get`
	`> {`
	`>  .sh.value=$(date +%r)`
	`> }`
	`$ print $time`
	`09:15:58 AM`
	`$ print $time  # it will change in a few seconds`
	`09:16:04 AM`
	· 每次設定 *varname* 時，即會呼叫 ***varname*.set** 函數。且會將指派的值給予 **.sh.value** 變數。當函數完成時，指派給 *varname* 的值將是 **.sh.value** 的值。例如：
	`$ function adder.set`
	`> {`
	`>  let .sh.value="`
	`$ {.sh.value} + 1"`
	`> }`
	`$ adder=0`
	`$ echo $adder`
	`1`
	`$ adder=$adder`
	`$ echo $adder`
	`2`
	· 每次取消設定 *varname* 時，即會執行 ***varname*.unset** 函數。除非變數在其本身的函數中已取消設定，否則不會真正取消設定此變數；若未取消設定，則保留其值。
	在所有紀律函數內，特殊變數 **.sh.name** 是設為變數的名稱，而 **.sh.subscript** 是設為變數註標的值 (如果適用的話）。

函數環境	以 *function myfunc* 格式宣告的函數會在個別的函數環境下執行，而且支援區域變數。宣告為 *myfunc*() 的函數會使用與母項 shell 相同的環境執行。
變數	shell 將保留以 .sh. 開頭的變數，而且這些變數有特殊的意義。請參閱本表格中的紀律函數說明，以取得 **.sh.name** 、**.sh.value** 和 **.sh.subscript** 的說明。您也可取得代表 shell 版本的 **.sh.version**。
指令回覆值	指令的回覆值如下： · 若找不到要執行的指令，則會將回覆值設為 127。 · 若找到要執行的指令，卻無法執行該指令時，則回覆值為 126。 · 若指令可以執行，但卻為某個信號終止，則回覆值為 256 加上信號號碼。
PATH 搜尋規則	會先搜尋特殊的內建指令，再搜尋所有函數（包括 **FPATH** 目錄中的函數），然後是其他內建函數。
shell 歷程	**hist** 指令可讓您顯示和編輯 shell 指令歷程。在 ksh shell 中，會使用 **fc** 指令。**fc** 指令是 **hist** 的別名。變數包括 *HISTCMD*（在 shell 現行歷程中每執行一個指令，即增量一次）和 *HISTEDIT*（指定在使用 **hist** 指令時，要使用哪個編輯器）。

項目	說明
內建指令	強化的 Korn shell 包含下列內建指令： · **builtin** 指令會列出全部可用的內建指令。 · **printf** 指令的工作方式類似 **printf()** C 程式庫常式。請參閱 **printf** 指令。 · **disown** 會阻止 shell 將 SIGHUP 傳送到指定的指令。 · **getconf** 指令的運作模式與獨立式指令 **/usr/bin/getconf** 相同。請參閱 **getconf** 指令。 · **read** 內建指令具有下列旗標： – **read -d {*char*}** 可讓您指定定界字元，而非預設的新行。 – **read -t {*seconds*}** 可讓您指定一段時間限制（以秒為單位），在經過該段時間之後 **read** 指令將會逾時。如果 **read** 逾時，它會傳回 FALSE。 · **exec** 內建指令具有下列旗標： – **exec -a {*name*} {*cmd*}** 指定 *cmd* 的引數 0 將換成 *name*。 – **exec -c {*cmd*}** 告訴 **exec** 在執行 *cmd* 之前，先清除環境。 · **kill** 內建指令具有下列旗標： – **kill -n {*signum*}** 用來指定信號號碼，以傳送至處理程序，而 **kill -s {*signame*}** 是用來指定信號名稱。 – 不含引數的 **kill -l** 會列出全部信號名稱（但不會列出其號碼）。 · **whence** 內建指令具有下列旗標： – **-a** 旗標會顯示所有相符項，不只是所找到的第一個相符項。 – **-f** 旗標通知 **whence** 不要搜尋任何函數。 · 跳出字元順序已供 **print** 和 **echo** 指令使用。Esc（跳出）鍵可以順序 \E 來代表。 · 所有一般的內建指令都可以辨識 **-?** 旗標，其顯示指定指令的語法。 · **getopts** 內建需要 **optstring** 包含前導 **+**，來容許以 **+** 符號為開頭的選項。

項目	說明
Korn shell ksh 與 Korn shell ksh93 之間的其他差異	其他差異包括： · 對於 Korn shell ksh93，不能使用 **typeset -fx** 內建指令來匯出函數。 · 對於 Korn shell ksh93，不能使用 **alias -x** 內建指令來匯出別名。 · 對於 Korn shell ksh93，單引號後面的錢幣符號 ($') 會解譯為 ANSI C 字串。您必須以引號括住錢幣符號 (\"$\"')，才能取得舊的 (ksh) 行為。 · 已變更 Korn shell ksh93 內建指令的引數剖析邏輯。Korn shell ksh 內建指令的未記載引數剖析組合無法在 Korn shell ksh93 中運作。例如，**typeset -4i** 的運作類似 Korn shell ksh 中的 **typeset -i4**，但無法在 Korn shell ksh93 中運作。 · 對於 Korn shell ksh93，當展開時，指令替代及算術擴充是在特殊環境變數 PS1、PS3 及 ENV 上執行。因此，您必須使用反斜線 (\) 來跳出沈音符號 (`) 及錢幣符號與左括號 ($()，才能保留舊行為。例如，Korn shell ksh 會照字面將 x=$'*name*\t*operator*' 指派為 $*name*\t*operator*；Korn shell ksh93 會展開 \t，並將它指派為 *name*<\t expanded>*operator*。若要保留 Korn shell ksh 行為，您必須用引號括住 $。例如，x="$"'*name*\t*operator*'。 · Korn shell ksh93 已移除 *ERRNO* 變數。 · 在 Korn shell ksh93 中，對於非互動式 shell，不會展開重新導向符號後面的檔名。 · 對於 Korn shell ksh93，您必須使用 **alias** 指令的 **-t** 選項，才能顯示已追蹤的別名。追蹤別名特性現在已作廢，因此顯示的別名可能不會被追蹤。 · 對於 Korn shell ksh93，在 emacs mode 中，Ctrl+T 會交換現行與前一個字元。對於 ksh，Ctrl+T 會交換現行字元與下一個字元。 · Korn shell ksh93 不允許 ${*name operator value*} 內有不對稱的括弧。例如，${*name*-(} 需要如 ${*name*-\(} 的跳出符號，才能在這兩個版本中運作。 · 對於 Korn shell ksh93，**kill -l** 指令只會列出信號名稱，而非它們的數值。

Korn shell 或 *POSIX shell* 中的結束狀態

由 shell 所偵測到的錯誤，如語法錯誤，將 導致 shell 傳回一非零值的結束狀態。否則，shell 將傳回最後完成指令的結束 狀態。

shell 會列印指令或函數名稱與錯誤狀況，以報告偵測到的執行時期錯誤。如果發生錯誤的指令行數目大於 1， 還會將行號列印在指令或函數名稱後面的 **[ ]**（方括弧）之中。

就非互動式的 shell 而言，特殊內建或其他類型指令所遇到的錯誤，將導致 shell 寫入診斷訊息，如下表所示：

錯誤	特殊內建	其他公用程式
shell 語言語法錯誤	將結束	將結束
公用程式語法錯誤（選項或運算元錯誤）	將結束	將不會結束
重新導向錯誤	將結束	將不會結束
變數指派錯誤	將結束	將不會結束
擴充錯誤	將結束	將結束
指令找不到	不適用	可能結束
點 Script 找不到	將結束	不適用

如果任何顯示為「將（可能）結束」的錯誤 發生在 subshell 中，subshell 將（可能）以非零值狀態結束，但包含 subshell 的 Script 不會因為此錯誤而結束。

在所有顯示在表格中的情況中，互動式 shell 將寫入一則診斷訊息至標準錯誤，而不會結束程式。

Korn shell 中的參數

底下將說明 Korn shell 參數。

參數的定義如下：

- 以下列任一字元來作為 ID：星號 (*)、@ 符號 (@)、井字符號 (#)、問號 (?)、連字號 (-)、錢幣符號 ($) 及驚嘆號 (!)。這些稱為特殊的參數。
- 以數字來表示的引數（位置參數）
- 以 ID 來表示的參數，其具有一個值和零個或數個屬性（指名的參數/變數）。

typeset 特殊的內建指令可指派值和屬性給指名參數。Korn shell 支援的屬性是由 **typeset** 特殊的內建指令來說明。匯出參數將值和屬性傳送至環境。

指名參數的值以下列方式來指派：

```
Name=Value [ Name=Value ] ...
```

如果 **Name** 參數有設定 **-i** 整數屬性，則 **Value** 參數會受制於算術評估。

shell 支援一維陣列機能。陣列的元素是以註標來參考的。註標是以方括弧 ([ ]) 所括住的算術運算來表示。若要將值指派給陣列，請使用 `set -A` *Name Value* ... 全部註標的值必須在 0 至 511 此範圍內。陣列不需要宣告。以有效註標對指名參數的任何參考皆為合法，且需要的話，會建立一個陣列。 不以註標來參考陣列，相當於參考元素 0。

位置參數是以 **set** 特殊指令來指派值的參數。當呼叫 shell 時，會從引數 0 設定 **$0** 參數。$ 字元是用來引介可替代的參數。

相關概念

shell 啟動
您可以用 **ksh** 指令、**psh** 指令 (POSIX shell) 或 **exec** 指令來啟動 Korn shell。

Korn shell 函數
function 保留字會定義 shell 的函數。shell 會讀取並儲存成內在函數。別名名稱會在讀取函數時分辨出來。shell 會以和指令相同的方式中執行函數，與引數一起傳送為位置參數。

Korn shell 或 POSIX shell 中的算術評估
Korn shell 或 POSIX shell 一般內建 **let** 指令可讓您執行整數算術。

相關參考

Korn shell 複合指令
複合指令可以是簡式指令的清單、管線，或它可以使用保留字作為開頭。當您撰寫 shell Script 時，大部分都會使用複合指令，例如：**if**、**while** 及 **for**。

Korn shell 或 *POSIX shell* 中的參數替代
Korn shell 或 POSIX shell 可讓您執行參數替代。

下列為可替代的參數：

項目	說明
${*Parameter*}	shell 會讀取從 ${ 到成對之 } 中間的所有字元，以作為相同字組的部分，即使該字組包含大括弧或 meta 字元。替換指定參數的值。當 *Parameter* 參數其後跟著字母、數字或底線，但皆不作為其名稱的一部分來解譯時，或指名參數加上註標時，則不能省略大括弧。 如果指定的參數包含一或多個數字，則它是位置參數。一個以上數字的位置參數必須以大括弧括住。如果變數值為 * 或 @，則會替換以 $1 開頭的每一個位置參數（以欄位分隔字元來隔開）。如果使用具有註標 * 或 @ 的陣列 ID，則會替換每一個元素（以欄位分隔字元來隔開）的值。
${#*Parameter*}	如果 *Parameter* 參數的值是 * 或 @，則會替換位置參數的數目。否則，會替換 *Parameter* 參數所指定的長度。

項目	說明
${#*Identifier*[*]}	替換由 *Identifier* 參數所指定之陣列的元素數目。
${*Parameter*:-*Word*}	如果 *Parameter* 參數已設定且非空值，則會替換其值；否則會替換 *Word* 參數的值。
${*Parameter*:=*Word*}	如果 *Parameter* 參數未設定或為空值，則會設為 *Word* 參數的值。不能以此方式來指派位置參數。
${*Parameter*:?*Word*}	如果 *Parameter* 參數已設定且非空值，則會替換其值。否則，會列印 *Word* 變數的值，然後結束 shell。 如果省略 *Word* 變數，則會列印標準訊息。
${*Parameter*:+*Word*}	如果 *Parameter* 參數已設定且非空值，則會替換 *Word* 變數的值。
${*Parameter*#*Pattern*} | ${*Parameter*##*Pattern*}	如果指定的 shell *Pattern* 參數符合 *Parameter* 參數的開頭值，則此替代的值是 *Parameter* 參數的值再刪除符合的部分。否則，會替換 *Parameter* 參數的值。以第一個格式而言，會刪除最小符合型樣。以第二個格式而言，會刪除最大符合型樣。
${*Parameter*%*Pattern*} | ${*Parameter*%%*Pattern*}	如果指定的 shell *Pattern* 符合 *Parameter* 變數值的尾端，則此替代的值是 *Parameter* 變數的值再刪除符合的部分。否則，會替換 *Parameter* 變數的值。以第一個格式而言，會刪除最小符合型樣。以第二個格式而言，會刪除最大符合型樣。 在之前的表示式中，除非將 *Word* 變數當作被替換的字串使用，否則不會評估。因此，在下列範例中，只有在未設定 **-d** 旗標或是空值時，才會執行 **pwd** 指令： `echo ${d:-$(pwd)}`

註： 如果之前的表示式中省略 **:**，則 shell 僅會檢查是否已設定 *Parameter* 參數。

相關概念

無人式終端機

如果終端機已登入且無人操作，所有系統就很容易受到攻擊。當系統管理人員讓使用 root 授權啟用的終端機處於無人看管的狀態時，會發生最嚴重的問題。一般而言，使用者只要離開終端機，就應該要登出。

Korn shell 或 *POSIX shell* 中預先定義的特殊參數

Korn shell 或 POSIX shell 會自動設定某些參數。

shell 會自動設定下列參數：

項目	說明
@	展開以 $1 開頭的位置參數。以一個空格隔開每一個參數。 如果您在 $@ 兩側加上 "，則 shell 會將每一個位置參數視為個別的字串。如果位置參數不存在，則 shell 會將陳述式展開成無引號的空字串。
*****	展開以 $1 開頭的位置參數。 shell 以 **IFS** 參數值的第一個字元來隔開每一個參數。 如果您在 $* 兩側加上 "，則 shell 會以雙引號括住位置參數值。每一個值是以 **IFS** 參數的第一個字元來隔開。
#	指定傳給 shell 的位置參數數目（以十進位表示），但不計算 shell 程序本身的名稱。因此，**$#** 參數會產生最高編號之位置參數的數目。 此參數的其中一個主要用途是檢查必要數目的引數是否存在。
-	在呼叫或使用 **set** 指令時，將旗標提供給 shell。
?	指定上次執行之指令的結束值。它的值是一個十進位字串。大部分指令傳回 0 來表示順利完成。shell 本身會傳回 **$?** 參數作為它的結束值。

項目	說明

$	定義此 shell 的程序號碼。由於程序號碼是全部現存處理程序中唯一的，所以此字串（最多 5 數位）通常用來產生暫用檔的唯一名稱。
	下列範例說明建議您練習的動作：僅為此目的之目錄中建立暫用檔：
	`temp=$HOME/temp/$$`
	`ls >$temp`
	`.`
	`.`
	`.`
	`rm $temp`

!	指定最近呼叫之背景指令的程序號碼。
zero (0)	展開 shell 或 shell Script 的名稱。

***Korn shell* 或 *POSIX shell* 中的檔名替代**

Korn shell 或 POSIX shell 可掃描 *Word* 變數所指定的每一個指令字詞以找出某些字元，藉以執行檔名替代。

如果指令字組包含 ***)**、**?** 或 **[** 字元，且未設定 **-f** 旗標，則 shell 將把此字組視為型樣。shell 將以符合此型樣、 且已根據現行語言環境作用的對照順序排序的檔名置換此字組。若 shell 找不到一個符合型樣的檔名，它不會改變該字組。

當 shell 使用型樣來執行檔名替代時，**.** 及 **/** 字元必須明確地符合。

註 : Korn shell 不會在其他型樣相符的案例中特別處理這些字元。

這些型樣相符字元指示下列的替代：

項目	說明
*****	符合任何字串，其中包括空字串。
?	符合任何單一字元。
[...]	符合括號內的其中一個字元。根據現行語言環境下作用的對照順序，一對以連字號 (-) 隔開的字元在詞彙上符合此對包含範圍內的字元。如果在開頭的 [之後的第一個字元是 ！，則任何未含括的字元皆符合。連字號 (-) 可以包含在字集中，作為第一個或最後一個字元。

您也可以使用 `[:charclass:]` 表示法，在某範圍指示中比對檔名。這個格式指示系統找出任何符合屬於類別 的單一字元。字元組成特定字元類別的定義， 是藉由 setlocale 子常式的 **LC_CTYPE** 種類來呈現。現行語言環境中指定的所有字元類別皆可為系統所辨識。

某些字元類別的名稱如下：

- **alnum**
- **alpha**
- **cntrl**
- **digit**
- **graph**
- **lower**
- **print**
- **punct**
- **space**
- **upper**
- **xdigit**

例如，`[[:upper:]]` 將符合任一大寫字母。

Korn shell 是根據對照元素、符號或等值類別來支援檔名擴充。

PatternList 是一個以 | 隔開的一或多個型樣清單。組合型樣是由一或多個下列項目所形成：

項目	說明
?(*PatternList***)**	選擇性地符合任何一個給定的型樣
(PatternList***)**	符合零或多個給定型樣
+(*PatternList***)**	符合一或多個給定型樣
@(*PatternList***)**	僅符合一個給定型樣
!(*PatternList***)**	符合給定型樣以外的其他情況

型樣相符具有一些限制。如果檔名的第一個字元是點 (.)，則只有開頭也是一個點的型樣才能符合它。例如，* 符合檔名 myfile 及 yourfile， 但不符合檔名 .myfile 及 .yourfile。若要符合這些檔案，請使用下列型樣：

```
.*file
```

如果型樣不符合任何檔名，則傳回該型樣本身作為試圖符合的結果。

檔案及目錄名稱應不包含 *、?、[或] 字元，因為在嘗試型樣相符期間會導致無限遞迴（亦即，無限迴圈）。

引號除去
某些字元如果沒有用引號括住，將會被移除。

在原始字詞中出現的引號字元、反斜線 (\)、單引號 (') 以及雙引號 (") 將會被移除（除非是位於引號內）。

***Korn shell* 或 *POSIX shell* 中的輸入及輸出重新導向**
在 Korn shell 執行指令前，它會先掃描指令行尋找重新導向字元。這些特殊的表示法會將 shell 導入重新導向輸入及輸出。

重新導向字元可出現在簡式指令的任何地方，或是可在之前加上指令或是隨著一個指令。它們不會傳送至呼叫的指令中。

此 shell 會在使用 **Word** 或 **Digit** 參數之前，執行指令及參數替代，除非另有說明。只有當型樣與單一檔案相符，且沒有執行空白解譯的情況下，才能執行檔名替代。

項目	說明
<*Word*	使用 **Word** 參數所指定的檔案來作為標準輸入（檔案描述子 0）。
>*Word*	使用 **Word** 參數所指定的檔案來作為標準輸出（檔案描述子 1）。如果此檔不存在，則 shell 會建立該檔。如果檔案存在且沒有 **noclobber** 選項，則會產生錯誤；否則，此檔會被截短成零長度。 **註 :** 當多個 shell 設定 **noclobber** 選項，而且它們將輸出重新導向到相同的檔案，則會有競爭狀況，這會造成這些一個以上的 shell 處理程序寫入檔案。shell 不會去偵測或防止這種競爭狀況。
>|*Word*	和 **>***Word* 指令一樣，但是這個重新導向陳述式會置換 **noclobber** 選項。
> >*Word*	使用 **Word** 參數所指定的檔案來作為標準輸出。如果目前存在此檔案，則 shell 會將輸出附加至此檔（藉由第一次找到的檔案尾字元）。如果此檔不存在，則 shell 會建立該檔。
<>*Word*	會開啟由 **Word** 參數所指定的檔案，以讀取或寫入成標準輸入。
<<[**-**]*Word*	會讀取 shell 輸入的每一行，直到它找到只含有 **Word** 參數值或檔案結尾字元的那一行為止。此 shell 不會在指定的檔案中執行參數替代、指令替代或是檔案名稱替代。所產生的文件（稱為 here 文件）成為標準輸入。一旦引用 **Word** 參數的任何字元，則不會判讀文件中的字元。

here 文件被視為單一字詞，它是從下一個換行字元之後開始， 一直到只含有定界字元的字行為止（沒有尾端的空白字元）。然後，如果有下一個 here 文件的話，則啟動它。格式如下所示：

```
[n]<<word
   here document
delimiter
```

如果 *word* 中有任何字元被引號括起來，則會移除 *word* 上的引號，以形成定界字元。將不會展開 *here* 文件行。否則，此定界字元將是 *word* 本身。如果 *word* 中沒有字元被引號括起來，則 here 文件中的所有字行都將會擴充，以用於參數擴充、指令替代及算術擴充。

shell 會執行重新導向資料的參數替代。若要避免解譯 \、$ 及單引號 (') 字元，以及 **Word** 參數的第一個字元，請在字元之前加上 \ 字元。

如果連字號 (-) 附加在 << 之後，則 shell 會刪除 **Word** 參數和文件中的所有前導標籤。

項目	說明
<&*Digit*	從 **Digit** 參數所指定的檔案描述子中複製標準輸入
>& *Digit*	從 **Digit** 參數所指定的檔案描述子中複製標準輸出
<&-	關閉標準輸入
>&-	關閉標準輸出
<&p	將輸入從共同處理程序移至標準輸入
>&p	將輸出從共同處理程序移至標準輸出

如果這些重新導向選項的其中一個之前加上數字，則所參照的檔案描述子的編號由數字（代替預設的 0 或 1）來指定。下列範例中，shell 會開啟檔案描述子 2 以寫成檔案描述子 1 的副本。

```
... 2>&1
```

重新導向所被指定的順序是有其意義的。shell 會根據在評估時期的（*FileDescriptor*、*File*）連結，來評估每個重新導向。例如，在此陳述式中：

```
... 1>File 2>&1
```

檔案描述子 1 會與 **File** 參數所指定的檔案連結。shell 會使檔案描述子 2 和與檔案描述子 1 (*File*) 連結的檔案連結。如果重新導向的次序被反轉，則檔案描述子 2 將會與終端機連結（假設檔案描述子 1 先前已執行），且檔案描述子 1 會與 **File** 參數指定的檔案連結。

如果指令之後跟隨著 & 符號 (&)，且工作控制不在作用中，則指令的預設標準輸入是空的檔案 /dev/null。否則，指令的執行環境，會含有由輸入和輸出規格所啟動修正的 shell 檔案描述子。

相關概念

輸入及輸出重新導向

AIX 作業系統可讓您使用特定的 I/O 指令與符號，操作您的系統之輸入和輸出 (I/O) 資料。

相關工作

將輸出重新導向到列入輸入 (here) 文件

您可以將輸出重新導向至列入輸入 (here) 文件。

共同處理機能

Korn shell 或是 POSIX shell，可讓您執行一或多個指令作為背景處理。這些從 shell Script 內部執行的指令，稱為共同處理。

在指令之後放置 |& 運算子，以指定共同處理程序。指令的標準輸入及輸出會被導入您的 Script 中。

共同處理必須符合下列的限制：

- 在每個訊息的結尾加入換行字元
- 將每個輸出訊息傳送至標準輸出

· 在每個訊息之後清除它的標準輸出

下列範例說明輸入如何傳送至共同處理，以及如何由共同處理傳回：

```
echo "Initial process"
./FileB.sh |&
read -p a b c d
echo "Read from coprocess: $a $b $c $d"
print -p "Passed to the coprocess"
read -p a b c d
echo "Passed back from coprocess: $a $b $c $d"
```

```
FileB.sh
      echo "The coprocess is running"
   read a b c d
   echo $a $b $c $d
```

所產生的標準輸出如下：

```
Initial process
Read from coprocess: The coprocess is running
Passed back from coprocess: Passed to the coprocess
```

使用 **print -p** 指令來寫入共同處理。使用 **read -p** 指令，從共同處理中讀取。

相關概念

Korn shell 或 POSIX shell 指令

Korn shell 為互動式指令直譯器及指令程式語言。它符合「電腦環境可攜性作業系統介面 (POSIX)」，是國際標準的作業系統。

重新導向共同處理的輸入與輸出

共同處理的標準輸入及輸出是藉由使用 I/O 重新導向，被重新指派到一個已編號的檔案描述子。

例如，指令：

```
exec 5>&p
```

會將共同處理程序的輸入移至檔案描述子 5。

完成此共同處理程序之後，您就可以使用標準重新導向語法，將指令輸出重新導向至共同處理程序。您也可以啟動其他的共同處理。從兩種共同處理中的輸出是連接至相同管線且會讀取具有 **read -p** 的指令。若要停止共同處理，請鍵入：

```
read -u5
```

***Korn shell* 或 *POSIX shell* 內建指令**

內建在 Korn shell 和 POSIX shell 中的特殊指令並且在 shell 處理程序中執行。

除非有另外的指示，否則會將輸出寫入檔案描述子 1 ，而且如果指令不含任何錯誤的語法，則結束的狀態是零 (0)。輸入及輸出的重新導向是被允許的。內建指令類型有兩種，特殊內建指令 及一般內建指令。

特殊內建指令與一般指令不同的使用方法如下列：

· 特殊指令中語法錯誤可能導致 shell 結束執行指令。如果您在一般內建指令中語法錯誤，則這種情形將不會發生。如果特殊內建指令中的語法錯誤沒有結束 shell 程式，則結束值為非零值。

· 指令完成之後變數會指定與特殊內建指令一起保持生效狀態。

· I/O 重新導向是在參數指派之後處理。

此外，在參數指派格式中的字組， 跟隨著 **export**、**readonly** 及 **typeset** 的特殊指令會與相同參數指派規則一起展開。顎化符號替代會在 = 之後執行，而字詞分割及檔名替代則不會執行。

相關概念

Korn shell 或 POSIX shell 指令

Korn shell 為互動式指令直譯器及指令程式語言。它符合「電腦環境可攜性作業系統介面 (POSIX)」，是國際標準的作業系統。

Korn shell 函數
function 保留字會定義 shell 的函數。shell 會讀取並儲存成內在函數。別名名稱會在讀取函數時分辨出來。shell 會以和指令相同的方式中執行函數，與引數一起傳送為位置參數。

相關參考

Korn shell 或 POSIX shell 特殊內建指令的清單
特殊指令內建在 Korn shell 及 POSIX shell 中，並且在 shell 處理程序中執行。

Korn shell 或 POSIX shell 一般內建指令
下列是 Korn shell 或 POSIX shell 一般內建指令的清單。

Korn shell 或 *POSIX shell* 的特殊內建指令說明
特殊指令內建在 Korn shell 及 POSIX shell 中，並且在 shell 處理程序中執行。

底下將說明 Korn shell 的特殊內建指令：

`: eval newgrp shift`
`. exec readonly times`
`break exit return trap`
`continue export set typeset`
`unset`

項目	說明
: [*Argument* ...]	只展開引數。它在指令為必要時使用，如同 **if** 指令的 *then* 條件，但是此指令並不會完成任何動作。
. *File* [*Argument* ...]	會讀取完整指定的檔案然後執行此指令。此指令會在 shell 的環境中執行。搜尋的路徑是由 *PATH* 變數指定，用來尋找包含指定檔案的目錄。如果指定了任何引數，則它們會成為位置參數。否則，此位置參數是不變的。這個結束狀態是最近期執行之指令的結束狀態。請參閱 第 196 頁的『Korn shell 或 POSIX shell 中的參數替代』，以取得位置參數的詳細資訊。 **註 :** *.File* [*Argument* ...] 指令會在執行任何指令之前，讀取整個檔案。因此，在此檔案中的 **alias** 和 **unalias** 指令不會套用到任何檔案中所定義的函數。
break [*n*]	自含括的 **for**、**while**、**until** 或 **select** 迴圈中結束。如果您指定 *n* 變數，則此指令會岔斷 *n* 變數所指定的層次編號。*n* 的值是任一等於或大於 1 的整數。
continue [*n*]	回復含括的 **for**、**while**、**until** 或 **select** 迴圈的下一個疊代。如果有指定 *n* 變數，則指令會在第 n^{th} 個含括迴圈處回復。*n* 的值是任一等於或大於 1 的整數。
eval [*Argument* ...]	會將特殊引數讀取為輸入至 shell 並執行結果指令。
exec [*Argument* ...]	執行由此 shell（未建立新處理程序）位置中的引數所指定的指令。會出現輸入及輸出引數並會影響現行處理程序。如果您不指定引數，則 **exec** 指令會依照輸入及輸出重新導向清單的規定來修正檔案描述子。在此情況下，任何大於 2 的檔案描述子編號會與呼叫其他程式時關閉此機制一起開啟。
exit [*n*]	依照 *n* 變數所指定的結束狀態，來結束 shell。*n* 變數必須為無正負號介於 0-255 範圍的十進位整數。如果您省略 *n* 參數，則結束狀態為最近所執行之指令的狀態。檔案結尾字元也會跳出 shell，除非開啟 **set** 特殊指令的 **ignoreeof** 選項。
export -p [*Name*[**=** *Value*]] ...	會標記自動匯出至後來執行的指令環境之特殊名稱。 **-p** 會在標準輸出寫入名稱以及所有匯出數數的值，格式如下： `"export %s= %s\n", <name> <value>`
newgrp [*Group*]	相當於 **exec/usr/bin/newgrp** [*Group*] 指令。 **註 :** 這個指令不會返回。

項目	說明
readonly -p [*Name*[**=** *Value*]] ...	會標記由 *Name* 參數所指定為唯讀的名稱。這些名稱無法由後續指派來變更。 **-p** 會在標準輸出寫入名稱以及所有匯出數數的值，格式如下： `"export %s= %s\n", <name> <value>`
return [*n*]	導致某個 shell 函數傳回呼叫 Script。此傳回狀態由 *n* 參數來指定。如果您省略 *n* 參數，則傳回狀態為最近所執行之指令的狀態。如果您在某個函數或是 Script 之外呼叫 **return** 指令，則它與 **exit** 指令相同。
set [**+** |**-abCefhkmnostuvx**] [**+** |**-o** *Option*]... [**+** |**-A***Name*] [*Argument* ...]	如果沒有指定的選項或是引數，則 **set** 指令會在現行位置的對照序列中寫入此名稱以及所有 shell 的變數值。當指定選項時，它們會設定或取消設定 shell 的屬性，說明如下： **-A** 陣列指派。取消設定 *Name* 參數並從指定的 *Argument* 參數清單中循序指派值。如果使用 **+A** 旗標，則不會先取消設定 *Name* 參數。 **-a** 自動匯出定義的所有後續參數。 **-b** 非同步的通知使用者背景工作完成。 **-C** 相當於 `set -o noclobber`。 **-e** 執行 ERR 設陷（如果設定的話），並於指令為非零結束狀態時結束，除非簡式指令為： + 包含於 && 或 || 清單中 + 指令緊跟在 if、while 或 until 之後 + 包含於跟在 ! 之後的管線中 此模式在讀取設定檔時停用。 **-f** 停用檔名替代。 **-h** 當第一次被發現時，將每個指令指定為磁軌別名。 **-k** 將所有參數指派引數放置在指令的環境中，而不只是指令名稱之前的那些引數。 **-m** 在個別的處理程序中執行背景工作並在完成之後列印一行。背景工作的結束狀態會在完成訊息中報告出來。在具有工作控制的系統中，會針對互動式 shell 自動啟用此旗標。如需相關資訊，請參閱 第 212 頁的『Korn shell 或 POSIX shell 中的工作控制』。 **-n** 讀取指令並檢查語法錯誤，但不會執行。此旗標是被忽略的互動式 shell。

項目	說明
	-o *Option* 列印現行選項設定及錯誤訊息，如果您不指定引數的話。您可以在單一的 **ksh** 指令行中設定一個以上的選項。如果 **+o** 旗標已使用，則指定的選項為取消設定。若指定引數，引數會導致位置參數設定或取消設定。引數，就像是 *Option* 變數所指定的一樣，可以是下列其中之一： **allexport** 與 **-a** 旗標相同。 **bgnice** 在較低的優先順序中執行所有背景工作。這是預設的模式。 **emacs** 輸入 emacs-樣式列入指令輸入的編輯器。 **errexit** 與 **-e** 旗標相同。 **gmacs** 輸入 gmacs 樣式列入指令輸入的編輯器。 **ignoreeof** 當它發現檔案尾字元時不會結束 shell。若要結束 shell，您必須使用 **exit** 指令，或是按 Ctrl-D 鍵順序達 11 次以上。 **keyword** 與 **-k** 旗標相同。 **註 :** 這個旗標只適用於和 Bourne shell 舊版相容。我們強烈建議不要使用。 **markdirs** 將反斜線 / 附加至為檔名替代結果的所有目錄名稱。 **monitor** 與 **-m** 旗標相同。 **noclobber** 從截短現存的檔案中阻止重新導向。當您指定這個選項時， 重新導向符號後面必須有一條垂直線 (>|) 來截斷檔案。 **noexec** 與 **-n** 旗標相同。 **noglob** 與 **-f** 旗標相同。 **nolog** 防止 `.profile` 及 `$ENV` 檔案中的函數定義被儲存在歷程檔中。 **nounset** 與 **-u** 旗標相同。 **privileged** 與 **-p** 旗標相同。

項目	說明
	trackall 與 **-h** 旗標相同。 **verbose** 與 **-v** 旗標相同。 **vi** 輸入 vi-樣式列入編輯器的插入模式作為指令輸入。輸入跳離字元 033 將編輯器放置在移動模式中。一項傳回會傳送此行。 **viraw** 會處理每個鍵入的 vi 模式字元。 **xtrace** 與 **-x** 旗標相同。 **-p** 停用 $HOME/.profile 檔案的處理程序，並使用 /etc/suid_profile 檔來代替 ENV 檔。這個模式在每次有效的使用者 ID (UID) 或群組 ID (GID) 不等於實際的 UID 或 GID 時會停用。關閉這個選項將生效的 UID 或 GID 設定為實際的 UID 和 GID。 **註 :** 系統不支援 **-p** 選項，因為作業系統不支援 **setuid** shell Script。 **-s** 將位置參數按詞彙排序。 **-t** 在讀取和執行一個指令之後結束。 **註 :** 這個旗標只適用於和 Bourne shell 舊版相容。我們強烈建議不要使用。 **-u** 當替換時會將取消設定參數視為錯誤。 **-v** 會將 shell 輸入行列印成已讀取。 **-x** 會將指令和它們的引數列印成已執行。 **-** 關閉 **-x** 和 **-v** 旗標並停止檢查旗標的引數。 — 阻止任何旗標開始變更。將 *$1* 參數設為以 - 開頭的值時， 這個選項就非常有用。如果沒有引數跟隨這個旗標，則位置參數未設定。 在任何 **set** 指令旗標之前加上 + 而不是 - 來關閉此旗標。當呼叫 shell 時您可以使用這些旗標。呼叫 'set +o' 而不使用任何引數時，它會對 shell 以適合重新輸入的格式來顯示現行選項設定，作為達到相同選項設定的指令。現行的旗標集是在 **$-** 參數中找到的。除非您指定 **-A** 旗標，否則剩餘的引數會是位置參數，並且會依序指派為 $1、$2、... 等等。如果沒有給定引數，則這些名稱及所有名稱參數值會列印成標準輸出。
shift [*n*]	將位置參數更名成開頭具有 $n+1 ... 一直到 $1 ...。*n* 參數的預設值為 1。*n* 參數是評估非負數小於或是等於 *$#* 參數的任何一個算術運算式。
times	列印累計 shell 以及從 shell 中處理執行的使用者和系統次數。

項目	說明
trap [*Command*] [*Signal*] ...	當 shell 接收指定的信號時會執行指定的指令。當設陷已設定且被採用時，*Command* 參數就會讀取。*Signal* 參數會指定為一個數字或是信號的名稱。設陷指令以信號號碼的次序執行。若要在進入現行 shell 時被忽略的信號上設定設陷，任何嘗試都是無效的。 如果指令是 -，所有的設陷會重設為它們的原始值。如果您省略這個指令，並且第一個信號是以數值編號的信號，則 **ksh** 指令會將 *Signal* 參數值重設為原始值。 **註 :** 如果您省略此指令並且第一個信號是符號名稱，則信號會被解譯為指令。 如果 *Signal* 參數值是 **ERR** 信號，則指定的指令在每次以非零狀態結束時會執行。如果此信號是 **DEBUG**，則指定的指令會在每一個 指令之後執行。如果 *Signal* 參數值是 **0** 或是 **EXIT** 信號，並且 **trap** 指令是在函數的主體中執行，則指定的指令會在函數完成之後執行。如果在任何函數以外設定的 **trap** 指令之 *Signal* 參數為 **0** 或 **EXIT**，則指定的指令會在從 shell 中跳出時執行。 **註 :** 如果 Script 在函數內收到 SIGINT 信號，當 Shell 結束時無法設陷捕捉 EXIT 信號。 沒有引數的 **trap** 指令，會列印出與每個信號編號關聯的指令清單。如果指定的指令為 NULL（以 ""（空引號）指示），則 **ksh** 指令將忽略信號。如需 Korn Shell 或 POSIX Shell 如何將字元讀成一般字元的相關資訊，請參閱第 189 頁的『在 Korn shell 或 POSIX shell 中以引號括住字元』。 如需用於 **trap** 指令，但沒有加上 **SIG** 字首的 *Signal* 參數值之完整清單，請參閱 *Technical Reference: Base Operating System and Extensions, Volume 2* 中的 **sigaction**、**sigvec** 或 **signal** 子常式。

項目	說明
typeset [**+HLRZfilrtux** [*n*]] [*Name*[**=***Value*]] ...	會設定 shell 參數的屬性和值。當在函數內部呼叫時，會建立一個新的 *Name* 參數的案例。此參數值和類型在函數完成時會被還原。您可以使用 **typeset** 指令指定下列旗標： **-H** 在非 AIX 的機器上提供 AIX 對主電腦檔案的對映。 **-L** 從 *Value* 參數中填滿左邊並移除前導的空白。如果 *n* 參數值為非零，則它會定義欄位的寬度；否則，它會由第一個引數值的寬度來決定。當已指派參數，它會以空白或是截短將右邊填滿（如果必要的話)，以適用於此欄位。如果也設定了 **-Z** 旗標，則前導零會被移除。**-R** 旗標會關閉。 **-R** 填滿右邊並填滿前導的空白。如果 *n* 參數值為非零，則它會定義欄位的寬度；否則，它會由第一個引數值的寬度來決定。如果參數是重新指派的，則此欄位仍保持填滿空白或是從尾端截短。**L** 旗標會關閉。 **-Z** 如果第一個非空白字元為數字，且未設定**-L** 旗標，則會填滿右邊並填滿前導零。如果 *n* 參數值為非零，則它會定義欄位的寬度；否則，它會由第一個引數值的寬度來決定。 **-f** 指出名稱會參考函數名稱，而不是參數名稱。不能作任何指派，而有效的旗標為 **-t**、**-u**、以及 **-x**。**-t** 旗標會開啟執行追蹤此函數。**-u** 旗標會導致此函數標記為未定義的。*FPATH* 變數是用來搜尋尋找當此函數為被參考時的函數定義。**-x** 旗標可讓函數定義在和 **ksh** 指令一起呼叫的 shell Script 中，維持有效。 **-i** 將參數定義為一個整數，讓算術較為快速。 如果 *n* 變數值為非零，則它會定義基本輸出算術；否則，第一個指派的會決定基本的輸出。 **-l** 將所有大寫字元轉換為小寫。**-u** 大寫轉換旗標會關閉。 **-r** 會標記由 *Name* 參數所指定為唯讀的名稱。這些名稱無法由後續指派來變更。 **-t** 具名的參數標籤。標籤可由使用者定義且對 shell 沒有特殊的意義。 **-u** 將所有小寫字元轉換為大寫字元。 **-l** 小寫旗標會關閉。 **-x** 會由 *Name* 參數標記指定的名稱，作為自動匯出至後續執行指令的環境。 使用 + 而非 - 來關閉 **typeset** 指令旗標。如果您未指定 *Name* 參數但是指定旗標，則會印出具有這些旗標集的參數名稱（以及選用性的值）清單。(使用 + 而非 - 以避免將值列印出來。）如果您未指定任何名稱或旗標，則此名稱及所有參數的屬性會被列印出來。
unset [**-fv**] *Name* ...	取消設定此值及由名稱清單給定的參數屬性。如果有指定 **-v**，則 *Name* 是指某個變數名稱，而且 shell 會將其取消設定，並從環境中將它移除。唯讀變數無法取消設定。取消設定 *ERRNO*、*LINENO*、*MAILCHECK*、*OPTARG*、*OPTIND*、*RANDOM*、*SECONDS*、*TMOUT* 以及底線 (_) 變數，會移除它們的特殊意義，即使已在後續指派它們。 如果有設定 **-f** 旗標，則 *Name* 會指向某個函數名稱，並且 shell 將會取消設定該函數定義。

Korn shell 或 *POSIX shell* 的一般內建指令說明

這裡將說明 Korn 或 POSIX shell 的內建指令。

Korn shell 提供下列一般內建指令：

```
alias fg print ulimit
bg getopts pwd umask
cd jobs read unalias
command
kill setgroups
wait
echo let setsenv
test whence
fc
```

項目	說明
alias [**-t**] [**-x**] [*AliasName*[**=** *String*]] ...	建立或重新定義別名定義，或者將存在的別名寫入標準的輸出中。 如需相關資訊，請參閱 **alias** 指令。
bg [*JobID*...]	將每一個指定的工作放置在背景中。如果 *JobID* 參數未指定，則現行工作會放在背景中。請參閱 第 212 頁的『Korn shell 或 POSIX shell 中的工作控制』，以取得工作控制的詳細資訊。 如需在背景執行工作的相關資訊，請參閱 **bg** 指令。
cd [*Argument*]	
cd *Old New*	這個指令可以是兩種不同的格式。在第一個格式中，將現行目錄變更為 *Argument* 參數所指定的一個指令。如果此 *Argument* 參數的值是連字號 (-) ，則此目錄會變更為上一個目錄。HOME shell 變數是 *Argument* 參數的預設值。 PWD 變數是設定為現行的目錄。 CDPATH shell 變數會定義含有 *Argument* 參數值目錄的搜尋路徑。另一個目錄名稱會由冒號 (**:**) 隔開。預設的路徑為空值，將指定現行目錄。現行目錄是由空值路徑名稱指定的，在等號或是在路徑清單中介於冒號定界字元之間的任何位置中，會立即出現。如果指定的引數是以斜線 (/) 作為開頭，則不使用搜尋路徑。否則，每個路徑中的目錄會搜尋引數。 第二種 **cd** 指令格式，會使用 *New* 變數指定的字串來取代目前目錄 PWD 中，*Old* 變數指定的字串，並嘗試切換到這個新目錄。
command [**-p**] *CommandName* [*Argument* ...]	
command [**-v** | **-V**] *CommandName*	促使 shell 將所指定的指令和引數當作單純的指令，阻止 shell 函數查閱動作。 如需相關資訊，請參閱 **command** 指令。
echo [*String* ...]	將字元字串寫入標準輸出。請參閱 **echo** 指令，以取得用法及說明。**-n** 旗標未受支援。
fc [**-r**] [**-e** *Editor*] [*First* [*Last*]]	
fc -l [**-n**] [**-r**] [*First* [*Last*]]	

項目	說明
fc -s [*Old*=*New*] [*First*]	會顯示您的指令歷程檔案的內容或是呼叫編輯器，以修正並重新執行在 shell 中輸入的上一個指令。 如需相關資訊，請參閱 **fc** 指令。
fg [*JobID*]	會將每個指定的工作帶至前端。如果您不指定任何的工作，則此指令會將現行工作帶至前端中。 如需在前景執行工作的相關資訊，請參閱 **fg** 指令。
getopts *OptionString Name* [*Argument* ...]	檢查合法選項的 *Argument* 參數。 如需相關資訊，請參閱 **getopts** 指令。
jobs [**-l** | **-n** | **-p**] [*JobID* ...]	會顯示現行 shell 環境中啟動的工作狀態。如果沒有指定具有 *JobID* 參數特定的工作，則會顯示所有作用中的工作資訊。如果某個終止工作被報告，則 shell 會從那些現行 shell 環境中所瞭解的清單移除該工作的處理程序 ID。 如需相關資訊，請參閱 **jobs** 指令。
kill [**-s** { *SignalName* | *SignalNumber* }] *ProcessID...* **kill** [-*SignalName* | -*SignalNumber*] *ProcessID...*	傳送某個信號（根據預設值，是 SIGTERM 信號）至執行中的處理程序。這個預設動作會正常的停止處理程序。如果您想要停止某個處理程序，則在 *ProcessID* 變數中指定處理程序 ID (PID)。shell 會報告每一個在背景處理程序的 PID（除非您以管線啟動一個以上的處理程序，在這樣的情況下 shell 會報告最後處理程序的編號）。您也可以使用 **ps** 指令來尋找該指令的處理程序 ID 編號。
kill -l [*ExitStatus*]	列出信號名稱。 如需相關資訊，請參閱 **kill** 指令。
let *Expression* ...	評估指定的算術運算。如果最後表示式是非零則結束狀態是 0，不然就是 1。請參閱第 184 頁的『Korn shell 或 POSIX shell 中的算術評估』，以取得進一步資訊。
print [**-Rnprsu** [*n*]] [*Argument* ...]	會將 shell 列印輸出。如果未指定任何旗標，或是指定 了連字號 (-) 或雙連字號 (--) 旗標，那麼就會如 **echo** 指令所述，將引數列印至標準輸出。旗標的作用如下： **-R** 在原始資料模式中列印（**echo** 指令的跳出慣例會被忽略）。**-R** 旗標會列印 **-n** 以外的所有後續引數以及旗標。 **-n** 阻止換行字元新增至輸出。 **-p** 將引數寫到執行 |& 的處理程序的管線，而不寫到標準輸出。 **-r** 在原始資料模式列印。**echo** 指令的跳出慣例會被忽略。 **-s** 將引數寫入歷程檔案代替標準輸出。 **-u** 在所放置的輸出中，指定一個一位數字的檔案描述子單元號碼 *n*。預設值是 1。
pwd	相當於 `print -r - $PWD`。 **附註：**內部 Korn shell **pwd** 指令不支援符號鏈結。

項目	說明
read [**-prsu** [**n**]] [*Name?Prompt*] [*Name...*]	採取 shell 輸入。使用 IFS 變數中的字元作為分隔字元，讀取一行並將它分散至欄位中。 如需相關資訊，請參閱 **read** 指令。
setgroups	執行 **/usr/bin/setgroups** 指令，其會執行為一個個別的 shell。請參閱 **setgroups** 指令，以取得這個指令的資訊。然而，這裡有一點差異。 **setgroups** 內建指令會呼叫 subshell，但 **setgroups** 指令會置換現行執行的 shell。因為內建的指令僅支援相容的指令，建議該 Script 要使用絕對路徑名稱 /usr/bin/setgroups，不要使用 shell 內建指令。
setsenv	執行 /usr/bin/setsenv 指令，取代目前執行的 shell。請參閱 **setsenv** 指令，以取得此指令的相關資訊。
test	與 [*expression*] 相同。請參閱 第 187 頁的『Korn shell 或 POSIX shell 的條件表示式』，以取得用法及說明。

項目	說明

ulimit [**-HSacdfmst**] [*Limit*]	設定或顯示使用者處理程序資源限制，如同 /etc/security/limits 檔中的定義。這個檔案包含下列預設限制：
	`fsize = 2097151`
	`core = 2048`
	`cpu = 3600`
	`data = 131072`
	`rss = 65536`
	`stack = 8192`
	`threads = -1`
	當有使用者新增至系統時，會使用這些值作為預設設定。這些值會在新增使用者至系統時利用 **mkuser** 指令來設定，或是利用 **chuser** 指令來變更。
	限制分為軟性和硬性。使用者可以變更他們的軟性限制，一直到硬性限制所設定的最大值，使用 **ulimit** 指令。您必須有 root 使用者權限來變更硬體資源限制。
	許多系統不包含一個或一個以上的這些限制。指定的資源限制會在 *Limit* 參數被指定時設定。*Limit* 參數的值可以是每個指定資源單元中的號碼，或是 unlimited 值。您可以指定下列 **ulimit** 指令旗標：
	-H
	指定設定該硬體給定的限制。如果您有 root 使用者權限，則您可以增加硬體的限制。任何使用者都可以縮減它。
	-S
	指定設定該軟體給定的限制。軟體限制可增加一定硬體限制的值。如果不指定 **-H** 或 **-S** 選項，則限制會引用這兩個選項。
	-a
	列出所有現行資源限制。
	-c
	在核心傾出的大小中指定 512 位元區塊的數字。
	-d
	以 KB 指定資料區域的大小。
	-f
	指定 512 位元區塊的數字作為可寫入子處理程序的檔案（任何可讀取的檔案大小）。
	-m
	指定 KB 數目作為實體記憶體的大小。
	-n
	指定處理程序可開啟的檔案描述子數目限制。
	-r
	指定每個處理程序的執行緒數目限制。
	-s
	指定 KB 數目作為堆疊區域的大小。
	-t
	指定以秒計的數字來計算一個處理程序使用的時間。
	現行資源限制在您省略 *Limit* 變數時會被列印。軟體限制除非您指定 **-H** 旗標否則會列印出來。當您指定一個以上的資源時，限制的名稱及單元會在列印值之前被列印出來。如果沒有給定選項，**-f** 旗標就是假設的。當您變更此值時，會將硬體和軟體限制設定為 *Limit*，除非您指定 **-H** 或 **-S**。
	如需使用者及系統資源限制的相關資訊，請參閱 getrlimit、setrlimit 或 vlimit subroutine。

項目	說明
umask [**-S**] [*Mask*]	決定檔案的許可權。在檔案建立時，此值與建立處理程序之許可權將決定檔案的許可權。預設值為 022。如果沒有指定 *Mask* 參數，**umask** 指令就會將現行 shell 環境的檔案模式建立遮罩顯示到標準輸出。 如需檔案許可權的相關資訊，請參閱 **umask** 指令。
unalias { **-a** | *AliasName...* }	移除每個指定的別名名稱定義，或移除所有別名定義（如果 **-a** 旗標已使用的話）。別名定義會從現行 shell 環境中移除。 如需相關資訊，請參閱 **unalias** 指令。
wait [*ProcessID...*]	等待指定的工作並且終止。如果您不指定工作，則指令會等待所有現行作用中的子處理程序。這個指令中的結束狀態是該等待的處理程序。 如需相關資訊，請參閱 **wait** 指令。
whence [**-pv**] *Name ...*	指出每一個指定的名稱，它是如何在使用作為指令名稱時被解譯的。在不使用任一旗標的情況下，**whence** 將會顯示對應於各個名稱的絕對路徑名稱（如果有的話）。 **-p** 針對指定的名稱執行路徑搜尋，即使這些是別名、函數或是保留字。 **-v** 產生更詳細的報告（其中指定了每個名稱的類型）。

Korn shell 或 *POSIX shell* 中的工作控制

Korn shell 或 POSIX shell，會提供控制指令序列或是 *jobs* 的機能。

當您執行 **set -m** 的特殊指令時， Korn shell 會連結每一個具有管線符號的工作。它會保留現行工作的表格、由 **jobs** 指令列印並將它們指派為短整數的數字。

當以 & 符號 (&) 在背景啟動工作時，shell 會印出類似下列的字行：

```
[1] 1234
```

此輸出指出工作（在背景啟動）為工作號碼 1。它亦顯示出此工作具有一個（頂層）處理程序，其處理程序 ID 為 1234。

如果您執行一項工作並想要另外執行某些工作時，請使用 Ctrl-Z 鍵順序。這個鍵順序會將 **STOP** 信號傳送給現行的工作。shell 通常會指出工作已停止，然後顯示一個 shell 提示。您可以處理此工作的狀態（用 **bg** 指令來將它放在背景）， 執行其他指令，最後用 **fg** 指令來使工作返回前景。Ctrl-Z 鍵順序會立即生效，而且會像捨棄擱置輸出 shell 中的岔斷一樣，並且會在您鍵入此鍵順序時無法讀取輸入。

如果嘗試從終端機中讀取，則會停止一項在背景中執行的工作。背景工作會正常的允許產生輸出。您可以發出 **stty tostop** 指令來停用這個選項。如果您設定了這個終端機的選項，則背景工作會在它們嘗試產生輸出或是讀取輸入時停止工作。

您可以使用數個方法來參照 Korn shell 中的工作。一項參照的工作是由它的處理程序中的任何處理程序 ID，或是以下列之一的方法來參照：

項目	說明
%*Number*	指定具有指定編號的工作。
%*String*	指定任何以 *String* 變數作為指令行開頭的工作。
%?*String*	指定其指令行中含有 *String* 變數的任何工作。
%%	指定現行工作。
%+	相當於 %%。
%-	指定上一個工作。

此 shell 會立刻辨識出處理狀態中的變更。每次成為暫停執行時它會正常的通知您，以致於沒有可能的進一步進度。shell 只在列印一個提示之前執行此動作，以致於它不會用其他方式分散您的工作。

當開啟監視模式，每個已完成的背景工作會觸發 **CHLD** 信號的設陷集。

如果您在工作已停止或是執行中時嘗試離開 shell（藉由鍵入 `exit` 或是使用 Ctrl-D 鍵順序），系統會以 `There are stopped (running) jobs.` 的訊息來警告您。請使用 **jobs** 指令來察看那一項工作受到影響。如果您立即重試結束，則 shell 會不警告您而終止停止的及執行中的工作。

信號處理
如果被呼叫的指令後面有 & 符號 (&)，而且工作的 **monitor** 選項不在作用中，則會忽略這個指令的 `SIGINT` 及 `SIGQUIT` 信號。否則，這些信號會令 shell 從它的母項中繼承此值。

已設定設陷的信號在 shell 等待前景指令完成時接收到信號，則與該信號有關的設陷會直到前景指令完成之後才執行。因此，`CHILD` 信號中的設陷直到前景工作終止才會執行。

***Korn shell* 或 *POSIX shell* 中的列入編輯**
一般來說，您會從終端機裝置中輸入每一個指令行，再跟著一個換行字元（**RETURN** 或 **LINE FEED**）。當您啟動 emacs、gmacs、或 vi 列入編輯選項時，則您可以編輯指令行。

下列指令會輸入編輯模式：

項目	說明
set -o emacs	進入 emacs 編輯模式，並起始 emacs 樣式的列入編輯器。
set -o gmacs	進入 emacs 編輯模式，並起始 gmacs 樣式的列入編輯器。
set -o vi	進入 vi 編輯模式，並起始 vi 樣式的列入編輯器。

每當指派給 *VISUAL* 或 *EDITOR* 變數的值是在這些選項名稱中結束，就會自動選取編輯選項。

註： 若要使用編輯特性，您的終端機必須接受 **RETURN** 作為沒有換行的換行字元。在此畫面中空格必須改寫現行的字元。

每一個編輯模式會開啟現行行中的視窗。這個視窗寬度是 *COLUMNS* 變數的值（如果它有定義的話），否則寬度為 80 個字元的空格。如果此行長度比視窗寬度減兩個字元還長，則系統會在視窗尾端顯示標示來通知您。就像是游標移動並到達視窗的界限一樣，此視窗會靠中對齊這個游標。所顯示的標示如下：

項目	說明
>	指出此行會向視窗右邊延伸。
<	指出此行會向視窗左邊延伸。
*	指出此行會向視窗兩邊延伸。

在每個編輯模式中的搜尋指令提供 Korn shell 歷程檔案的存取權。只有字串是符合的。如果字串中的前導字元為 ^，則必須自這一行的第一個字元開始符合。

相關概念
Korn shell 或 POSIX shell 指令
Korn shell 為互動式指令直譯器及指令程式語言。它符合「電腦環境可攜性作業系統介面 (POSIX)」，是國際標準的作業系統。

emacs 編輯模式
當您啟用 **emacs** 或 **gmacs** 選項時，便會進入 emacs 編輯模式。在這兩種模式之間僅有處理 Ctrl-T 編輯指令方式的差異。

如果要編輯，請移動游標至需要更正的指標並插入或刪除字元或字組（如果需要的話）。所有編輯指令會控制字元或是 ESC 序列。

編輯指令會從指令行的任何位置來操作（不只是從開始的地方）。在編輯指令之後不要按 Enter 鍵或是換行（下移鍵）鍵，除了要當作附註之外。

項目	說明
Ctrl-F	會將游標向前（右）移動一個字元。
Esc-F	會將游標向前移動一個字組（字元字串是由唯一的字母、數值或是底線所組成的）。
Ctrl-B	會將游標向後（左）移動一個字元。
Esc-B	會將游標向後移動一個字組。
Ctrl-A	會將游標向行的開頭移動。
Ctrl-E	會將游標向行的尾端移動。
Ctrl-] *c*	會在現行行將游標向前移動至指示的字元。
Esc-Ctrl-] *c*	會在現行行將游標向後移動至指示的字元。
Ctrl-X Ctrl-X	會交換游標並標記。
ERASE	會刪除上一個字元。（消除字元作為 **stty** 指令所定義的使用者定義，通常是 Ctrl-H 鍵順序。）
Ctrl-D	會刪除目前的字元。
Esc-D	會刪除目前的字組。
Esc-Backspace	會刪除上一個字組。
Esc-H	會刪除上一個字組。
Esc-Delete	會刪除上一個字組。如果您的岔斷字元是 Delete 鍵的話，則這個指令無效。
Ctrl-T	會在 emacs 模式中，將目前字元與下一個字元調換。在 gmacs 模式中調換上兩個字元。
Ctrl-C	會使用大寫書寫目前字元。
Esc-C	會使用大寫書寫目前字組。
Esc-L	會將目前字組變更為小寫。
Ctrl-K	會從游標處向行的尾端刪除。如果之前加上數值參數，其值會小於目前游標所在的位置，則此編輯指令會從最接近游標的位置刪除。如果之前加上數值參數，其值會大於目前游標所在位置，則此編輯指令會從最接近游標的位置刪除。
Ctrl-W	會從游標處刪除至標記處。
Esc-P	會從游標處向堆疊中的標記處擴張範圍。
KILL	使用者定義的 kill 字元，如同 **stty** 指令所定義的， 通常是 Ctrl-G 鍵順序或是 @。會刪除目前一整行。如果連續輸入兩個刪除的字元，所有後續的刪除的字元會導致換行（在使用報表終端機時非常有用）。
Ctrl-Y	會從本行中還原最後一個移除的項目。（將這個項目拉回此行。）
Ctrl-L	會換行並列印目前這一行。
Ctrl-@	（空值字元）會設定一個標記。
Esc-space	會設定一個標記。
Ctrl-J	（換行）會執行現行行。
Ctrl-M	(Return)會執行現行行。
EOF	會處理檔案尾字元，一般而言只有現行行是空值時「Ctrl-D」鍵順序才能當做是檔案尾。

項目	說明
Ctrl-P	會提取上一個指令。每次輸入「Ctrl-P」鍵順序，上一個指令就及時向後存取。當不在多行指令的第一行時會向後移動一行。
Esc-<	會提取最少的新近（最舊的）歷程行。
Esc->	會提取最多的新近（最初的）歷程行。
Ctrl-N	會提取下一個指令行。每次輸入「Ctrl-N」鍵順序，下一個指令行就及時向前存取。
Ctrl-R *String*	會針對上一個包含 **String** 參數所指定之字串的指令行，將其搜尋行歷程反轉。如果提供一個 0 值，則會向前搜尋。指定的字串會被 Enter 或是換行字元所終止。如果在字串之前加上 ^，則符合的行開頭必須具有 **String** 參數。如果省略 **String** 參數，則下一個包含最近 **String** 參數的指令行會被存取。在此情形下，0 的值會反轉搜尋的方向。
Ctrl-O	（作業）會執行現行行並提取與歷程檔案中現行行相關的下一行。
Esc *Digits*	（跳出）會定義數值參數。這個取得的數值會做為下一個指令的參數。會接受參數的指令是 **Ctrl-F**、**Ctrl-B**、**ERASE**、**Ctrl-C**、**Ctrl-D**、**Ctrl-K**、**Ctrl-R**、**Ctrl-P**、**Ctrl-N**、**Ctrl-]**、**Esc-.**、**Esc-Ctrl-]**、**Esc-_**、**Esc-B**、**Esc-C**、 **Esc-D**、**Esc-F**、 **Esc-H**、**Esc-L** 以及 **Esc-Ctrl-H**。
Esc *Letter*	(Soft-key) 會在別名清單中搜尋別名為 *_Letter* 的別名。如果搜尋的別名已定義，則它的值會放在輸入佇列之中。*Letter* 參數不可指定跳出函數。
Esc-[*Letter*	(Soft-key) 會在別名清單中搜尋別名為 *__Letter*（雙底線加字母）的別名。如果搜尋的別名已定義，則它的值會放在輸入佇列之中。這個指令可用來作為許多終端機上的程式功能鍵。
Esc-.	會在本行中插入上一個指令的最後一個字組。如果之前加上數值參數，則這個參數值會決定那一個字組要插入而不是插入最後一個字組。
Esc-_	與 **Esc-.** 鍵順序相同。
Esc-*	嘗試對現行字組進行檔名替代。如果字組不符合任何檔案或不包含任何特殊型樣字元，則會附加一個星號 (*)。
Esc-Esc	檔名完成。以現行字組替代所有檔名的最長共同字首，其符合具有附加星號的現行字組。如果符合的字詞是唯一的， 若檔案是目錄，會加上斜線 (/)，若檔案不是目錄，則會加上空格。
Esc-=	列出符合現行字組型樣的檔案，就像有附加星號 (*) 一樣。
Ctrl-U	會將下一個指令參數乘 4。
****	會跳出下一個字元。如果在編輯字元以及 **ERASE**、**KILL** 和 **INTERRUPT**（通常是刪除鍵）字元之前加上反斜線 (\)， 即可在指令行或是搜尋字串中輸入它們。反斜線會移除下一個字元的編輯特性（如果有的話）。
Ctrl-V	會顯示 shell 的版本。
Esc-#	會在此行的開頭插入井字號 (#)，然後執行此行。這會導致某個註解插入歷程檔案中。

vi 編輯模式
vi 編輯模式有兩種輸入模式。

模式包括：

· **輸入模式**。 當您輸入指令時，vi 編輯器會處於輸入模式中。

· **控制模式**。 按 Esc 鍵以輸入控制模式。

大部分控制指令會在指令之前接受一個可選用的重複 **Count** 參數。大部分的系統在 vi 模式時，一般的處理程序會最先啟用。若有下列任何情形，此指令會重覆：

· 速度是 1200 傳輸速率或是更大。

· 指令含有任一控制字元。

· 自提示列印至所經歷過的時間低於一秒。

Esc 字元會一般終止指令餘數的處理程序，且您可以在稍後修改指令行。這個組合具有鍵入之前回應原始資料模式之一般處理程序的優點。如果也設定了 **viraw** 選項，一般處理程序會一直停用。這個模式被不支援兩種替代的行尾定界字元的系統所隱含，而且可能會有助於某些終端機。

將可用的 vi 編輯指令分類如下。種類如下：

輸入編輯指令
底下將說明 Korn shell 的輸入編輯指令。

註 : 根據預設值，編輯器會處於輸入模式。

項目	說明
ERASE	會刪除上一個字元。（使用者定義的消除字元，如同 **stty** 指令所定義的， 通常是 Ctrl-H 或 #。）
Ctrl-W	會刪除上一個空白隔開的字組。
Ctrl-D	會終止 shell。
Ctrl-V	會跳出下一個字元。如果在編輯字元（如 **ERASE** 或 **KILL** 字元）之前加上 Ctrl-V 鍵順序， 即可在指令行或是搜尋字串中輸入它們。「Ctrl-V」鍵順序會移除下一個字元的編輯特性（如果有的話）。
****	跳出下一個 **ERASE** 或 **KILL** 字元。

移動編輯指令
底下將說明 Korn shell 的移動編輯指令。

「移動」編輯指令用來移動游標的方式如下：

項目	說明
[*Count*]**l**	會將游標向前（右）移動一個字元。
[*Count*]**w**	會將游標向前移動一個英數字組。
[*Count*]**W**	會將游標移至下一個隨著一個空白的字組開頭。
[*Count*]**e**	會將游標移至現行字組的尾端。
[*Count*]**E**	會將游標移至現行空白區隔字組的尾端。
[*Count*]**h**	會將游標向後（左）移動一個字元。
[*Count*]**b**	會將游標向後移動一個字組。
[*Count*]**B**	會將游標移至上一個空白區隔的字組。
[*Count*]**|**	會將游標移至由 *Counter* 參數所指定的直欄中。
[*Count*]**f***c*	會尋找現行行中的下一個 *c* 字元。
[*Count*]**F***c*	會尋找現行行中的上一個 *c* 字元。
[*Count*]**t***c*	相當於 **f** 後接 **h**。
[*Count*]**T***c*	相當於 **F** 後接 **l**。
[*Count*]**;**	依照 *Count* 參數所指定的次數， 重複最後一個單字元的尋找指令：**f**、**F**、**t** 或 **T**。

項目	說明
[*Count*]**,**	依照 *Count* 參數所指定的次數，反轉最後一個單字元的尋找指令。
0	會將游標移至某一行的開始處。
^	將游標移至某一行中第一個非空白字元。
$	會將游標移至某一行的尾端。

搜尋編輯指令
搜尋編輯指令用來存取您指令歷程的方式如下所示：

項目	說明
[*Count*]**k**	會提取上一個指令。
[*Count*]**-**	相當於 **k** 指令。
[*Count*]**j**	會提取下一個指令行。每次輸入 **j** 指令時，下一個指令會被存取。
[*Count*]**+**	相當於 **j** 指令。
[*Count*]**G**	提取號碼是由 *Counter* 參數來指定的指令。預設值為最少的新近歷程指令。
/*String*	會在歷程中反向搜尋，以尋找上一個包含指定字串的指令。字串會由 RETURN 或換行字元來終止。如果在指定的字串之前加上 ^，則符合的行開頭必須具有 *String* 參數。如果 *String* 參數值是空值，則上一個字串已被使用。
?*String*	與 **/***String* 相同，不過它是向前搜尋。
n	搜尋下一個符合最後型樣為 **/***String* 或 **?** 的指令。
N	搜尋下一個符合最後型樣為 **/***String* 或 **?** 的指令，但是方向相反。會搜尋由上一個 **/***String* 指令所輸入的字串歷程。

文字修改編輯指令
「修改文字」編輯指令修改指令行的方式如下：

項目	說明
a	會輸入輸入模式並且會在現行字元之後輸入文字。
A	會將文字附加在此行的尾端。相當於 **$a** 指令。
[*Count*]**c***Motion* **c**[*Count*]*Motion*	會刪除從現行字元開始，到 *Motion* 參數指定的字元，來移動游標，並進入輸入模式。如果 *Motion* 參數的值是 **c**，則會刪除整行，並進入輸入模式。
C	會透過此行尾端刪除現行字元並輸入輸入模式。相當於 **c$** 指令。
S	相當於 **cc** 指令。
D	會透過行的尾端刪除現行字元。相當於 **d$** 指令。

項目	說明
[*Count*]**d***Motion* **d**[*Count*]*Motion*	會刪除 *Motion* 參數所指定的字元及該字元之前的現行字元。如果 *Motion* 為 **d**，將會刪除整行。
i	會輸入輸入模式並且會在現行字元之前插入文字。
I	會在此行的開頭插入文字。相當於 **0i** 指令。

項目	說明
[*Count*]**P**	會在游標之前放置上一個修改的文字。
[*Count*]**p**	會在游標之後放置上一個修改的文字。
R	會輸入輸入模式並在畫面中改寫字元。
[*Count*]**r***c*	從現行游標所在的位置開始，將 *Count* 參數所指定的字元數目，置換成 *c* 參數所指定的字元。這個指令也會使游標在此字元被置換後向前移動。
[*Count*]**x**	會刪除目前的字元。
[*Count*]**X**	會刪除之前的字元。
[*Count*]**.**	會重複上一個修改文字指令。
[*Count*]~	反轉由 *Count* 參數所指定的字元數目，從現行游標所在位置開始，並使游標向前移動。
[*Count*]_	附加前一個指令的 *Count* 參數所指定的字組，並進入輸入模式。若省略 *Count* 參數，則會使用最後一個字。
*****	對現行字組加上星號 (*)，然後嘗試進行檔名替代。如果不符合所尋找的，則它會搖鈴。否則，此字組會由符合的型樣所置換並且輸入輸入模式。
****	檔名完成。以現行字組替代所有檔名的最長共同字首， 符合具有附加星號 (*) 的現行字組。如果符合的字詞是唯一的，若檔案是目錄，會加上 /。如果檔案不是目錄則會附加空格。

雜項編輯指令
下列是常用的編輯指令。

項目	說明
[*Count*]**y***Motion*	
y[*Count*]*Motion*	從現行字元一直拉到 *Motion* 參數所指定之游標位置標示的字元（含該字元），並將這所有的字元放在刪除緩衝區中。文字和游標不會被變更。
Y	會從現行位置拉至此行的尾端。相當於 **y$** 指令。
u	會還原最後一個文字修正的指令。
U	會還原所有在此行執行的文字修正指令。
[*Count*]**v**	將指令 `fc -e ${VISUAL:-${EDITOR:-vi}}` *Count* 傳回輸入緩衝區中。如果省略 *Count* 參數，則現行行為已使用。
Ctrl-L	會換行並列印目前這一行。這個指令只在控制模式中生效。
Ctrl-J	（換行）會執行現行行，而不管在什麼模式中。
Ctrl-M	（返回）會執行現行行，而不管在什麼模式中。
#	在指令行前面插入井字號 (#) 之後，傳送該指令行。如果您想將現行行插入不執行的歷程中，這是非當有用的。 如果指令行包含管線或分號或換行字元，就會在這些符號的每一個之前插入額外的井字號 (#)。如果要刪除所有的井字符號，請從歷程中擷取指令行，並輸入其他的井字號 (#)。
=	會符合現行字組型樣的檔名清單，就像是附加星號一樣。
@*Letter*	在別名清單中搜尋名為 *_Letter* 的別名。如果搜尋的別名已定義，則它的值會放在處理程序的輸入佇列中。

Korn shell 或 *POSIX shell* 指令

Korn shell 為互動式指令直譯器及指令程式語言。它符合「電腦環境可攜性作業系統介面 (POSIX)」，是國際標準的作業系統。

POSIX 不是作業系統，但就來源層次、跨多重系統而言，它是旨於建立應用程式可攜式的一種標準。POSIX 特性是建立在 Korn shell 的頂端中。Korn shell（通常也稱為 POSIX shell）提供許多與 Bourne 和 C shell 相同的特性，例如 I/O 重新導向功能、變數替代以及檔名替代。它也會包括數個其他指令和程式語特性：

註： 可使用一個名稱為 `rksh` 且有受到限制的 Korn shell 版本。如需相關資訊，請參照 **rksh** 指令。

項目	說明
算術評估	Korn shell 或 POSIX shell 可以使用內建的 **let** 指令來執行整數算術（使用 2 至 36 中任何一個數字做為基底）。 為了能夠在 Korn shell 辨識從 0（八進位）及 0x（十六進位）開始的數字，請執行下列指令： **export XPG_SUS_ENV=ON** 匯出 XPG_SUS_ENV 變數會使得已執行的指令與它們使用的程式庫，完全與 POSIX 相容。 **註：** 因為整個程式庫系統變成與 POSIX 相容，所以給定的指令預設值預期的行為可能會變更。 **export OCTAL_CONST=ON** 就辨識八進位及十六進位常數常數而言，匯出這個變數會導致 Korn shell 中宣告的常數解譯為與 POSIX 相容。
指令歷程	Korn shell 或 POSIX shell，會儲存一個記錄您輸入所有指令的檔案。您可以使用文字編輯器，來改變歷程檔中的指令然後重新發出該指令。
共同處理機能	可讓您在背景執行程式，並傳送及接收這些背景處理程序的資訊。
編輯	Korn shell 或 POSIX shell，會提供列入編輯選項讓您編輯指令行。編輯器就像 emacs、 gmacs 以及 vi 一樣是可用的。

Korn shell 指令是下列其中之一：

- 簡式指令
- 管線
- 清單
- 複合指令
- 函數

當您在 Korn shell 或 POSIX shell 中發出指令時，shell 會評估該指令，並執行下列作業：

- 執行所有指出的替代作業。
- 判斷指令是否有包含斜線 (/)。如果有，則 shell 會執行指定的路徑名稱所命名的程式。

如果指令不含斜線 (/)，則 Korn shell 或 POSIX shell 會繼續下列的動作：

- 判斷指令是否為特殊的內建指令。如果是，則 shell 會執行現行 shell 處理程序內的指令。
- 將指令與使用者定義的函數做比較。如果指令符合使用者定義的函數，則會儲存位置函數， 然後重設為 *function* 呼叫的引數。當此函數完成或是發出傳回時，位置函數清單會還原，且任何在函數之中的 EXIT 設陷集會完成。函數的值為最後指令執行的值。某個函數會在現行 shell 處理程序中完成。
- 如果指令名稱符合一般內建指令的名稱，則將會呼叫該一般內建指令。
- 建立處理程序，並嘗試使用 **exec** 指令來執行指令（如果指令不是內建指令，也不是使用者定義的函數的話）。

Korn shell 或 POSIX shell 會在每個目錄中的指定路徑搜尋可執行檔。*PATH* shell 變數會定義包含指令之目錄的搜尋路徑。另一個目錄名稱會以冒號 (:) 來隔開。預設的路徑是 `/usr/bin:`（以該順序來指

定 /usr/bin 目錄，以及現行目錄）。現行目錄是由兩個或兩個以上相鄰的冒號所指定的，或由路徑清單開頭或尾端的冒號所指定。

如果檔案具有執行許可權但不是一個目錄或是一個 a.out 檔，則 shell 會假設它含有 shell 指令。現行 Shell 處理程序會建立一個 Subshell 以讀取檔案。所有非匯出的別名、函數以及指名的參數會從檔案中移除。如果 shell 指令檔具有讀取權，或是在檔案中設定 **setuid** 或 **setgid** 位元，則 shell 會執行設定許可權的代理程式，並以向下傳送為開啟檔案的 shell 指令來完成 shell。括入括弧內的指令會在 subshell 中執行但不移除非匯出數量。

相關概念

可用的 shell
下列是 AIX 隨附的 shell。

共同處理機能
Korn shell 或是 POSIX shell，可讓您執行一或多個指令作為背景處理。這些從 shell Script 內部執行的指令，稱為共同處理。

Korn shell 或 POSIX shell 中的列入編輯
一般來說，您會從終端機裝置中輸入每一個指令行，再跟著一個換行字元（**RETURN** 或 **LINE FEED**）。當您啟動 emacs、gmacs、或 vi 列入編輯選項時，則您可以編輯指令行。

Korn shell 或 POSIX shell 中的算術評估
Korn shell 或 POSIX shell 一般內建 **let** 指令可讓您執行整數算術。

Korn shell 或 POSIX shell 內建指令
內建在 Korn shell 和 POSIX shell 中的特殊指令並且在 shell 處理程序中執行。

Korn shell 複合指令
複合指令可以是簡式指令的清單、管線，或它可以使用保留字作為開頭。當您撰寫 shell Script 時，大部分都會使用複合指令，例如：**if**、**while** 及 **for**。

下列是 Korn shell 或 POSIX shell 複合指令的清單：

指令語法	說明
for *Identifier* [**in** *Word* ...] **;do** *List* **;done**	每次執行 **for** 指令時，**Identifier** 參數就會設為下一個從 **in** *Word* ... 清單中取出的字組。如果省略 **in** *Word* ... 指令，則 **for** 指令就會針對已設定的每個位置參數， 執行一次 **do** *List* 指令。當清單中沒有多餘的字組時執行會結束。
select *Identifier* [**in** *Word* ...] **;do** *List* **;done**	**select** 指令會在發生標準錯誤（檔案描述子 2）時，列印出指定的字詞集，每個集之前都加上編號。如果省略 **in** *Word* ... 指令，那麼會改用位置參數。會列印出 **PS3** 提示，並且會從標準輸出中讀取一行。如果此行由列出的字組編號所組成，則 *Identifier* 參數值會設定為字組對應此編號的字組。 如果此行從標準輸出讀取是空的話，則選取的清單會重新列印。否則，*Identifier* 參數的值會設定為空值。從標準輸入中讀取行的內容會儲存在 **REPLY** 的參數中。*List* 參數會針對每一個選項，直到發生岔斷或是遇到檔案結尾字元。
case *Word* **in** [[**(**] *Pattern* [| *Pattern*] ... **)** *List* **;;**] ... **esac**	**case** 指令會執行與第一個符合 *Word* 參數之 *Pattern* 參數相關聯的 *List* 參數。此型樣的格式與用於檔名替代的格式相同。
if *List* **;then** *List* [**elif** *List* **;then** *List*] ... [**;else** *List*] **;fi**	*List* 參數指定所要執行之指令的清單。shell 會先執行 **if** *List* 指令。如果傳回零結束狀態，則它會執行 **then** *List* 指令。否則，就會執行 **elif** 指令後面的 *List* 參數所指定的指令。 如果由 **elif** *List* 指令中的最後一個指令傳回的值是零，則會執行 **then** *List* 指令。如果由 **then** *List* 指令中的最後一個指令所傳回的值是零，則會執行 **else** *List* 指令。如果沒有針對 **else** 或 **then** 指令執行 *List* 參數指定的指令，則 **if** 指令會傳回零結束狀態。

指令語法	說明
while *List* **;do** *List* **;done** **until** *List* **;do** *List* **;done**	*List* 參數指定所要執行之指令的清單。**while** 指令會重複執行 *List* 參數所指定的指令。如果在 **while** *List* 指令中的最後一個指令的結束狀態為零，則會執行 **do** *List* 指令。如果在 **while** *List* 指令中的最後一個指令的結束狀態不是零，則迴圈會終止。如果沒有執行 **do** *List* 指令中的任何一個指令，則 **while** 指令會傳回零結束狀態。**until** 指令可能會被用來取代 **while** 指令，以取消迴圈終止測試。
(*List***)**	*List* 參數會指定所要執行之指令的清單。shell 會在個別的環境中執行 *List* 參數。 **註：** 如果需要在巢狀結構中使用兩個相鄰的左括弧，您必須在兩個左括弧之間插入一個空格，來區分指令與算術評估。
{ *List***;}**	*List* 參數會指定所要執行之指令的清單。*List* 參數會簡單地執行。 **註：** 不同於 meta 字元（ ），{ } 表示保留字（用於特殊目的，和使用者宣告的 ID 不同）。如果要辨識，則這些保留字必須出現在行的開頭或是在分號 (;) 後面。
[[*Expression*]]	評估 *Expression* 參數。如果此表示式為真，則指令會傳回零結束狀態。
function *Identifier* **{** *List* **;}** or **function** *Identifier* **() {***List* **;}**	定義由 *Identifier* 參數所參考的函數。函數的主體是以 { } 來含括指定指令的清單。( ) 是由兩個運算子所組成，所以將空格字元與 *identifier*、(及) 混合使用是允許的，但不是必要的。
time *Pipeline*	執行 *Pipeline* 參數。使用時間、使用者以及系統計時會列印至標準錯誤。

相關概念

Korn shell 中的參數

底下將說明 Korn shell 參數。

shell 啟動

您可以用 **ksh** 指令、**psh** 指令 (POSIX shell) 或 **exec** 指令來啟動 Korn shell。

如果 shell 是由 **exec** 指令啟動，且零引數 (**$0**) 的第一個字元是連字號 (-)，則此 shell 會視為登入 shell。shell 會先讀取 `/etc/profile` 檔案中的指令，然後從現行目錄中的 `.profile` 檔或是從 `$HOME/.profile` 檔讀取（如果兩個檔案都存在的話）。然後 shell 會從檔案中讀取指令（如果該檔案存在的話），該檔案是由 ENV 環境變數值中所執行的參數替代來命名的。

如果您在呼叫 Korn shell 或 POSIX shell 時，指定 *File* [*Parameter*] 參數，則 shell 會執行由 *File* 參數（包括任何指定的參數）識別的 Script 檔。指定的 Script 檔必須具有讀取許可權；否則任何的 **setuid** 及 **setgid** 設定均會被忽略。然後，shell 讀取指令。

附註： 在呼叫 Korn shell 或 POSIX shell 時，請勿以 **-c** 或 **-s** 旗標來指定 Script 檔。

如需位置參數的詳細資訊，請參閱 第 196 頁的『Korn shell 中的參數』。

相關概念

Korn shell 中的參數

底下將說明 Korn shell 參數。

Korn shell 環境

指令開始執行時，它的所有已知變數（帶有其相關值）組成該指令的環境。

這個環境包括指令自其上代程序中繼承的變數，以及在呼叫指令的指令行中被指定為關鍵字參數的變數。shell 以數種方式與環境相互作用。當它啟動時，shell 會掃描環境並為找到的每個名稱建立參數， 賦予參數對應值並將它標記為匯出。執行指令繼承環境。

如果您修正 shell 參數值或使用 **export** 或 **typeset -x** 指令建立一個新的參數，則此參數會變成環境的一部分。由任一執行的指令來觀察的環境，因此會包含原先由 shell 所承繼的名稱一值配對，它的值可由現行的 shell 來修正，再加上任何其他使用 **export** 或 **typeset -x** 指令所產生的結果。所執行的指令

(subshell) 會察看其對所繼承之環境變數所做的任何修改， 但其子項 shell 或處理程序若要察看修改過的值，subshell 就必須將這些變數匯出。

任何簡式指令或函數的環境，會藉由在前面加上一個或一個以上的參數指派，以作變更。參數指派引數是採用 *Identifier==Value* 格式的字組。因此，下列兩種表示式是相等的（就指令的執行而言）：

```
TERM=450 Command arguments
```

```
(export TERM; TERM=450; Command arguments)
```

Korn shell 函數
function 保留字會定義 shell 的函數。shell 會讀取並儲存成內在函數。別名名稱會在讀取函數時分辨出來。shell 會以和指令相同的方式中執行函數，與引數一起傳送為位置參數。

Korn shell 或 POSIX shell 會在呼叫的函數環境中執行函數。下列全部是函數及呼叫中的 Script 所共用的，並會產生負面影響：

· 變數值及屬性（除非您在函數中使用 **typeset** 指令來宣告本端變數）

· 工作目錄

· 別名、函數定義及屬性

· 特殊參數 $

· 開啟檔案

下列項目不為函數及呼叫中的 Script 所共用，而且沒有負面影響。

· 位置參數

· 特殊參數 #

· 呼叫函數時，變數指派清單中的變數

· 使用函數中的 **typeset** 指令來宣告的變數

· Options

· 設陷。然而，信號會由呼叫中的 Script 所忽略也將會由此函數所忽略。

註 : 在較早的 Korn shell 版本中，除了 **EXIT** 和 **ERR** 之外， 設陷會由函數以及呼叫 Script 共用。

如果對 **0** 或 **EXIT** 的設陷是在函數的主體內部執行，則此動作會在函數完成之後，在呼叫函數的環境中執行。如果設陷是在函數主體外部執行，則此動作會在 Korn shell 結束後立即執行。在較早 Korn shell 的版本中，沒有 **0** 或 **EXIT** 中的設陷在函數主體外部，會在此函數結束後立即執行。

執行函數時，會有如 Korn shell 或 POSIX shell 內建指令中所說明的相同語法錯誤及變數指派內容。

每次在函數名稱被指定為簡式指令名稱時就會執行複合指令。運算元當作暫時的指令，在複合指令執行期間將會成為 positional 參數。特殊參數 # 也將變更為反應運算元的編號。特殊參數 0 將不會變更。

return 特殊指令是用來從呼叫函數中返回的。在函數內部發生錯誤，控制權會返回至呼叫程式。

函數 ID 是利用 **typeset** 特殊指令的 **-f** 或 **+f** 選項列出。**-f** 選項也會列出函數的文字。函數是使用 **unset** 特殊指令的 **-f** 選項取消定義。

通常，函數會在 shell 執行 shell Script 時取消設定。**typeset** 特殊指令的 **-xf** 選項允許函數匯出至執行不具個別 shell 呼叫的 Script 中。必須在個別的 shell 呼叫之間定義的函數，應該利用 **typeset** 特殊指令的 **-xf** 選項，於 ENV 檔案中指定。

如果函數未順利完成宣告，則函數定義的結束狀態為零。否則，它將會大於零。函數呼叫的結束狀態是函數最新執行之指令的結束狀態。

相關概念
Korn shell 中的參數
底下將說明 Korn shell 參數。

Korn shell 或 POSIX shell 內建指令

內建在 Korn shell 和 POSIX shell 中的特殊指令並且在 shell 處理程序中執行。

Korn shell 或 *POSIX shell* 指令歷程
Korn shell 或 POSIX shell 會從您的終端機裝置上，將輸入的指令儲存為歷程檔。

如果設定了，則此 *HISTFILE* 變數值是歷程檔的名稱。如果未設定或無法寫入 *HISTFILE* 變數，則會使用 `$HOME/.sh_history` 歷程檔案。如果歷程檔不存在而 Korn shell 無法建立該檔，或是如果該檔存在而 Korn shell 沒有附加至該檔的許可權，則 Korn shell 是將暫用檔使用為歷程檔。shell 會使用具有適當許可權，相同命名的歷程檔來存取所有互動式 shell 的指令。

根據預設值，若是非 root 使用者，Korn shell 或 POSIX shell 會儲存最後 128 個指令的文字，若是 root 使用者，則會儲存 512 個指令。歷程檔的大小（由 *HISTSIZE* 變數指定）是沒有限制的， 儘管相當大的歷程檔可導致 Korn shell 的啟動變慢。

指令歷程替代
使用 **fc** 內建指令來列出或編輯部分的歷程檔。如果要選取編輯或列出檔案的一部分，請指定編號或指令的第一個或前幾個字元。

您可指定單一指令或指令範圍。

如果您沒有指定編輯器器作為 **fc** 一般內建指令的引數，則會使用由 *FCEDIT* 變數指定的編輯器。如果沒有定義 *FCEDIT* 變數，將會使用 `/usr/bin/ed` 檔案。編輯後的指令將在您結束編輯器時列印與執行。

編輯器名稱的連字號 (-) 是用來略過編輯階段並重新執行該指令。在此情形下，可在執行之前，使用格式為 `Old=New` 的替代參數來修改指令。比方說，如果 `r` 是 `fc -e -` 的別名，則輸入 `r bad=good c` 時，就會執行最近一次以字母 c 作為開頭的指令，並且會將第一個出現的 bad 字串置換成 good 字串。

相關工作
清單之前輸入的指令（history 指令）
使用 **history** 指令來列出您之前已輸入過的指令。

***Korn shell* 或 *POSIX shell* 中的指令別名**
Korn shell 或 POSIX shell 可讓您建立別名，以自訂指令。

alias 指令定義文字格式 `Name=String` 為一別名。當您使用別名來作為指令行的第一個字時，Korn shell 會檢查它是否已處理過相同名稱的別名。若有，Korn shell 不會置換該別名。如果沒有處理過具有相同名稱的別名，Korn shell 就會用該別名的值來置換別名。

別名的第一個字元可以是除了 meta 字元以外的任何可列印字元。剩餘字元則必須同樣是有效的 ID。置換字串可以包含任何有效的 shell 文字，包含 meta 字元。

如果別名值的最後一個字元為一空格，則 shell 將檢查替代別名後的字組。您可以使用別名重新定義特殊的內建指令， 但無法重新定義保留字。別名定義無法透過 **ksh** 的呼叫而繼承。然而，如果您指定 **alias -x**，對以名稱來呼叫而不呼叫個別 shell 的 Script 來說，別名仍然有效。若要匯出別名定義，並使子程序能夠存取它們，您必須在您的環境檔中指定 **alias -x** 以及別名定義。

使用 **alias** 指令來建立、列出及匯出別名。

使用 **unalias** 指令來移除別名。

用來建立別名的格式如下：

```
alias Name=String
```

其中 **Name** 參數會指定別名的名稱， **String** 參數則會指定別名的值。

下列匯出的別名是由 Korn shell 預先定義的， 但可以取消設定或重新定義它們。建議您不要變更，因為這樣，以後若有任何人預期別名是 Korn shell 預先定義的，便會產生混淆。

```
autoload='typeset -fu'
false='let 0'
functions='typeset -f'
hash='alias -t'
history='fc -l'
integer='typeset -i'
nohup='nohup '
```

```
r='fc -e -'
true=':'
type='whence -v'
```

Korn shell (**ksh**) 的非互動呼叫不支援別名；例如在 shell script 中，或是在 **ksh** 中使用 **-c** 選項，如下所示：

```
ksh -c alias
```

相關工作

建立指令別名（alias shell 指令）
alias 可讓您建立指令、檔名或任何 shell 文字的快速鍵名稱。藉由使用別名，在執行經常性工作時，您可節省許多時間。您可以建立指令別名。

追蹤別名
別名經常使用作為完整路徑名稱的簡稱。別名機能選項可讓您自動設定別名值為相對應指令的完整路徑名稱。此別名的 特殊的類型為一追蹤別名。

追蹤別名可藉由刪除 shell 搜尋 *PATH* 變數的完整路徑名稱的需求，以加速執行。

set -h 指令會啟用指令 *tracking*，如此一來，每當參照某個指令時，shell 就會定義所追蹤之別名的值。此數值每當您重設 *PATH* 變數時，將會取消定義。

這些別名仍將繼續追蹤，如此下一後續參 照將會重新定義該數值。數個追蹤的別名將編譯至 shell 內。

顎化符號字元替代
在 shell 執行別名替代之後，它會檢查每一個字，察看該字是不是以未引用的顎化符號 (~) 作為開頭。如果是，shell 會檢查這個字（直到第一個斜線 (/)），以查看它是不是符合 `/etc/passwd` 檔案中的使用者名稱。如果發現符合項目，shell 會以符合的使用者之登入目錄來置換 ~ 字元與名稱。此 處理程序稱為顎化符號字元替代。

如果 shell 未發現符合資料，則不會變更 原始文字。如果 ~ 字元為字詞中的唯一字元，或其後跟著加號 (+) 或連字號 (-)，則 Korn shell 亦會作特殊的置換：

項目	說明
~	被 *HOME* 變數的值置換
~+	被 *$PWD* 變數（現行目錄的完整路徑名稱）置換
~-	被 *$OLDPWD* 變數置換（前一個目錄的完整路徑名稱）

此外，當變數指派參數的值是以顎化符號 (~) 字元開始時，shell 將嘗試顎化符號替代作業。

Bourne Shell
Bourne shell 為一互動式指令直譯器及指令程式設計語言。使用

bsh 指令可以執行 Bourne shell。

Bourne shell 可執行作為登入 shell 或在登入 shell 之下的 subshell。僅**登入**指令可以呼叫 Bourne shell 作為登入 shell。為了要這麼做，它會使用 **bsh** 指令的特殊格式名稱：`-bsh`。以首字母連字號 (-) 來呼叫時，shell 會先讀取並執行在系統 `/etc/profile` 檔案中找到的指令，以及 `$HOME/.profile`（若有的話）。`/etc/profile` 檔案設定了所有使用者需要的變數。最後，shell 備妥從您的標準輸入中讀取指令。

如果在啟動 Bourne shell 時指定了 **File** [*Parameter*] 參數，則 shell 會執行 **File** 參數（包括任何指定的參數）識別的 Script 檔。指定的 Script 檔必須具有讀取許可權；否則任何的 `setuid` 及 `setgid` 設定均會被忽略。然後，shell 讀取指令。若使用了 **-c** 或 **-s** 旗標，就請不要指定 script。

相關概念

可用的 shell
下列是 AIX 隨附的 shell。

Bourne shell 環境

指令開始執行時，它的所有已知變數（帶有其相關值）組成該指令的環境。這個環境包括指令自其上代程序中繼承的變數，以及在呼叫指令的指令行中被指定為關鍵字參數的變數。

shell 將名稱為內建**匯出**指令引數的變數，傳送到它的子程序。這個指令將名稱變數置於 shell 及其所有未來子程序兩者的環境中。

關鍵字參數為以指派格式出現的變數與值配對，通常在指令行上的程序名稱之前（但亦請參閱 **set** 指令的旗標）。這些變數被放在所要呼叫之程序的環境中。

請參閱下列範例：

· 請考慮下列的程序，它顯示了兩個變數的值（儲存在名稱為 key_command 的指令檔中）：

```
# key_command
echo $a $b
```

下列指令行產生顯示如後的輸出：

```
輸入                                  輸出
a=key1   b=key2  key_command       key1 key2
a=tom    b=john  key_command       tom john
```

程序的關鍵字參數未包含在儲存於 $# 的參數計數之內。

程序可存取在其環境中任何變數的值。 然而，如果它變更這些值的任一個，則變更不會在 shell 環境中反應。就上述程序而言，這些變更是區域性的。若要將變更放置於程序會傳送到其子程序的環境中，您須將新的值匯出在該程序之內。

請參閱下列範例：

· 若要取得從現行 shell 匯出的變數清單，請鍵入：

```
export
```

· 若要取得現行 shell 中的唯讀變數清單，請鍵入：

```
readonly
```

· 若要取得現行環境中變數與值的配對清單，請鍵入：

```
env
```

如需使用者環境的詳細資訊，請參閱 第 277 頁的『/etc/environment 檔案』。

Bourne shell 中的條件性替代

一般來說，shell 會以指派給 *Variable* 變數的字串值來取代表示式 $*Variable*（若有的話）。然而，根據變數是否已設定或非空值，或兩種情況皆存在，有一種特殊的表示法可允許條件性替代。

根據定義，如果某變數曾被指派一個值，則表示已設定該變數。一個變數的值可以是空字串，您可使用下列任何一種方法來將該值指派給變數：

項目	說明
A=	
bcd=""	
Efg=''	將空字串指派給 A、bcd 及 Efg。
set '' ""	將第一個和第二個位置參數設為空字串，並取消設定其他所有的位置參數。

下列是您可以用來執行條件性替代的可用的表示式清單：

項目	說明
${*Variable***-** *String***}**	如果已設定此變數，請替換 *Variable* 值以取代該表示式。否則，使用 *String* 值來置換此表示式。
${ *Variable***:-***String* **}**	如果已設定此變數而且不是空值，請替換 *Variable* 值以取代該表示式。否則，使用 *String* 值來置換此表示式。
${ *Variable***=***String* **}**	如果已設定此變數，請替換 *Variable* 值以取代該表示式。否則，請將 *Variable* 值設定成 *String* 值，然後替換 *Variable* 值以取代該表示式。您不能以此方式來指派值給位置參數。
${ *Variable***:=***String* **}**	如果已設定此變數而且不是空值，請替換 *Variable* 值以取代該表示式。否則，請將 *Variable* 值設定成 *String* 值，然後替換 *Variable* 值以取代該表示式。您不能以此方式來指派值給位置參數。

${ *Variable***?***String* **}**	如果已設定此變數，請替換 *Variable* 值以取代該表示式。否則會顯示下列格式的訊息：
	`Variable: String`
	並結束現行 shell（除非 shell 為登入 shell）。 若沒有替 *String* 變數指定一個值，則 shell 會顯示下面訊息：
	`Variable: parameter null or not set`

${ *Variable***:?***String* **}**	如果已設定此變數而且不是空值，請替換 *Variable* 值以取代該表示式。否則會顯示下列格式的訊息：
	`Variable`
	`: String`
	並結束現行 shell（除非 shell 為登入 shell）。如果您沒有指定 *String* 值，shell 就會顯示下列訊息：
	`Variable: parameter null or not set`

${ *Variable***+***String* **}**	如果已設定變數，請用 *String* 值來取代此表示式。否則替換空字串。
${ *Variable***:+***String* **}**	如果已設定此變數而且不是空值，請替換 *String* 值以取代該表示式。否則替換空字串。

在條件性替代中，除非 shell 使用該變數作為替換的字串，否則不會評估 *String* 變數。因此，在下列範例中，只有在未設定 d 或為空值時，shell 才會執行 **pwd** 指令：

```
echo ${d:-`pwd`}
```

相關概念

Bourne shell 中使用者定義的變數
Bourne shell 會辨識可指派字串值的英數變數。

***Bourne shell* 中的位置參數**

當執行 shell 程序時，此 shell 會悄悄地建立位置參數，這些參數會依據其在指令行的位置來參照每一個字組在指令行上的位置。

位於位置 0 的字（程序名稱）稱為 $0，下一個字（第一個參數）稱為 $1，依此類推最多至 $9。若要參照編號大於 9 的指令行參數，請使用內建的 **shift** 指令。

您可以使用內建的 **set** 指令，明確地重設位置參數。

註： 未指定位置的引數時，該位置的位置參數會設定為空值。位置參數是廣域的，而且可傳送至巢狀 shell 程序。

相關概念

Bourne shell 中使用者定義的變數
Bourne shell 會辨識可指派字串值的英數變數。

相關參考

Bourne shell 中預先定義的特殊變數
有數個變數具有特殊的意義。下列變數只能由 Bourne shell 設定：

Bourne shell 中的檔名替代

Bourne shell 允許您執行檔名替代。

指令參數通常是檔名。您可自動產生一連串的檔名，作為指令行上的參數。欲執行此動作，請指定 shell 可辨識的字元作為型樣相符字元。當某個指令含有這種字元，則 shell 會以目錄中的檔名來置換該字元。

註： Bourne shell 不支援以同等字元分類為基礎的檔名擴充。

這種型樣中的大部分字元都符合自己，但您也可以在您的型樣中使用某些特殊型樣相符字元。這些特殊字元包括：

項目	說明
*****	符合任何字串，包括空字串
?	符合任何單一字元
[. . .]	符合任何以方括弧 ([) 含括的字元
[! . . .]	符合位於方括弧中的任何字元，而不是驚嘆號後的任何字元

在方括弧 ([) 內，由連字號 (-) 隔開的一對字元，根據字元值的二進位次序，在該配對的含括範圍內詞彙性地指定全部字元集。

型樣相符具有一些限制。如果檔名的第一個字元是點 (.)，則只有開頭也是一個點的型樣才能符合它。例如，* 符合檔名 *myfile* 及 *yourfile*， 但不符合檔名 *.myfile* 及 *.yourfile*。若要符合這些檔案，請使用下列型樣：

```
.*file
```

如果型樣不符合任何檔名，則傳回該型樣本身作為試圖符合的結果。

檔案及目錄名稱應不包含 *、?、[或] 字元，因為在嘗試型樣相符期間會導致無限遞迴（亦即，無限迴圈）。

Bourne shell 中的輸入及輸出重新導向

在指令中可以使用一些重新導向選項。

一般而言，大部分的指令並不知道它們的輸入或輸出是否與鍵盤、顯示畫面或檔案有關聯。因此，指令是適於使用在鍵盤或是 pipeline 中。

下列的重新導向選項可以出現在任何的簡式指令中。它們也可以在之前或之後加上某個指令，但是不傳送指令。

項目	說明
<File	使用指定的檔案作為標準輸入。
>File	使用指定的檔案作為標準輸入。如果檔案不存在則會建立該檔案；否則，會將它截短為零長度。
> >File	使用指定的檔案作為標準輸入。如果檔案不存在則會建立該檔案；否則，會新增輸出至檔案尾端。

項目	說明
<<[-]*eofstr*	將下列範圍中的所有指令行讀成標準輸入：從 *eofstr* 變數一直到只包含 *eofstr* 的指令行，或是一直到檔案結尾字元。如果把 *eofstr* 變數中的任何字元使用引號，shell 就不會展開或解譯任何輸入行中的字元。否則，它會執行變數和指令替代，並忽略有前置引號的換行字元 (**\newline**)。使用反斜線 (\) 來引用 *eofstr* 變數或輸入行中的字元。 如果您將連字號 (-) 加到 << 重新導向選項，那麼所有前置定位字元會從 *eofstr* 變數和從輸入行中刪除。
<&*Digit*	由 *Digit* 變數指定的與標準輸入有關的檔案描述子。
>&*Digit*	使標準輸出與 *Digit* 變數指定的檔案描述子產生關聯。
<&-	關閉標準輸入。
>&-	關閉標準輸出。

註： 已重新導向的 shell 不允許將輸出重新導向。

有關重新導向的詳細資訊，請參閱第 304 頁的『輸入及輸出重新導向』。

Bourne shell 內建指令的清單

以下列出 Bourne shell 內建的指令。

項目	說明
:	傳回零結束值。
.	從檔案參數讀取和執行指令，然後再傳回。
break	自含括的 **for**、**while** 或 **until** 指令迴圈跳出（如果有的話）。
cd	將現行目錄變更至指定的目錄。
continue	回復含括的 **for**、**while** 或 **until** 指令迴圈的下一個疊代。
echo	將字元字串寫入標準輸出。
eval	讀取引數作為 shell 的輸入，然後執行結果指令。
exec	執行 **Argument** 參數所指定的指令，而非此 shell 所指定的指令，但不建立新的處理程序。
exit	結束由 **n** 參數來指定結束狀態的 shell。
export	標記名稱以便自動匯出至後續執行指令的環境。
hash	尋找和記住指定的指令，其搜尋路徑中的位置。
pwd	顯示現行目錄。
read	從標準輸入讀取一行。
readonly	將 **Name** 參數所指定的名稱標示為唯讀。
return	使函數結束，並傳回指定的傳回值。
set	控制各不同參數在標準輸出的顯示。
shift	將指令行引數向左移位。
test	評估條件表示式。
times	顯示執行 shell 之處理程序的使用者和系統累計次數。
trap	當 shell 收到指定的信號時，執行指定的指令。
type	解譯 shell 會以何種方式將指定的名稱解譯為指令名稱。
ulimit	顯示或調整配置的 shell 資源。

項目	說明
umask	決定檔案的許可權。
unset	移除指定的名稱所對應的變數或函數。
wait	等待指定的子程序結束，或報告它的終止狀態。

相關參考
Bourne shell 內建指令
特殊指令內建在 Bourne shell 中，並且在 shell 處理程序中執行。

***Bourne Shell* 指令**
您可以在 Bourne Shell 發出指令。

當您在 Bourne Shell 中發出指令時，它將先評估該指令並完成所有指示替代。 然後在下列情況執行該指令：

· 指令名稱為 Bourne Shell 特殊內建指令。

或

· 指令名稱符合已定義功能的名稱。若是如此，Shell 將設定位置參數為功能參數。

若指令名稱不符合內建指令或已定義功能名稱，且指令名稱為一個已編譯（二進位）程式的可執行檔，此 Shell（作為母項）會建立一個立即執行程式的新（子項）處理程序。若此檔標記為可執行但並非編譯過的程式，Shell 將假設此為 Shell 程序。在這情況下，此 Shell 會建立本身的另一個實例 (*subshell*)，讀取此檔並執行其內含指令。Shell 亦在 subshell 中執行括弧中的指令。對使用者而言，編譯程式的執行方法與 Shell 程序完全相同。一般來說，Shell 會以下列次序在檔案系統目錄中搜尋指令：

1. /usr/bin
2. /etc
3. /usr/sbin
4. /usr/ucb
5. $HOME/bin
6. /usr/bin/X11
7. /sbin
8. 現行目錄

此 Shell 將依序搜尋每一個目錄，若無法找到該指令則將繼續尋找下一個目錄。

註： *PATH* 變數會決定 Shell 搜尋目錄的順序。您可以藉重設 *PATH* 變數來變更搜尋目錄之特殊順序。

如果您在執行指令時提供了特定的路徑名稱（例如，/usr/bin/sort）， Shell 就只會搜尋您指定的目錄，而不會搜尋其他目錄。如果指令名稱包含斜線 (**/**)，Shell 就不會使用搜尋路徑。

您可提供以根目錄開始的完整路徑名稱（如/usr/bin/sort）。 您亦可以指定與現行目錄相對的路徑名稱。例如，如果您指定：

```
bin/myfile
```

Shell 將在現行目錄下尋找一個名為 bin 的目錄，然後又在此目錄下尋找檔案 myfile。

註： 受限制的 Shell 不執行含有斜線 (/) 的指令。

Shell 將記得每一個執行指令的搜尋路徑位置（避免稍候之不必要的 **exec** 指令）。如果它在相關目錄中找到指令（名稱未以 **/** 開頭），則每當現行目錄改變時，Shell 就必須重新判斷指令的位置。每次您變更 *PATH* 變數或執行 **hash -r** 指令時，Shell 就會忘記所有先前記得的位置。

字元引號
許多字元對 shell 來說都有某個特殊的意義。有時候您必須隱藏那個意義。包圍住字串的單引號 (') 及雙引號 (")，或單一字元前的反斜線 (\)，可讓您隱藏字元的意義。

所有的字元（用來含括的單引號除外）都會逐字採用，並除去任何特殊的意義。因此，指令：

```
stuff='echo $? $*; ls * | wc'
```

指派了文字字串 `echo $? $*; ls * | wc` 給變數 `stuff`。shell 不會執行 **echo**、**ls** 及 **wc** 指令，也不會展開 **$?** 及 **$*** 變數與星號 (*) 特殊字元。

在雙引號內，錢幣符號 ($)、反引號 (`) 以及雙引號 (") 字元仍然有效，而其他字元則要採取文字方式。因此，在雙引號內，指令與變數替代皆可能出現。此外，引號不會影響 括起來的字串部分中之指令替代中的指令，所以那裡的字元將保留 它們特殊意義。

請考量下列結果：

```
ls *
file1 file2 file3
message="This directory contains `ls * ` "
echo $message
此目錄包含了 file1 file2 file3
```

這顯示已展開位在指令替代內的星號 (*) 特殊字元。

若要隱藏雙引號內的錢幣符號 ($)、反引號 (`) 及雙引號 (") 字元之特殊意義，請在這些字元的前面加上反斜線 (\)。當您不在一個具有反斜線的字元前使用雙引號時，就相當於將其 置於單引號中。因此，換行字元前的反斜線（亦即，行末的反斜線）會隱藏換行字元，並可讓您在下一個實體行中繼續指令行。

信號處理

如果指令以 & 符號終止（亦即，如果是在背景執行）， shell 會忽略所呼叫指令的 **INTERRUPT** 及 **QUIT** 信號。否則，信號具有 shell 自其母項繼承的值（但 **SEGMENTATION VIOLATION** 信號除外）。

如需相關資訊，請參閱 Bourne shell 內建指令 **trap**。

Bourne shell 複合指令

複合指令是下列其中一項。

- 管線（由管線 (|) 符號所隔開的一或多個簡式指令）
- 單一指令清單
- 以保留字開頭的指令
- 以控制運算子左括弧 (() 開頭的指令

除非另有提及，複合指令的傳回值是 其最後一個執行的簡式指令傳回值。

保留字

Bourne shell 的下列保留字僅在出現作為指令的第一個字，並且不帶引號時，才會被辨識為保留字。

```
for             do              done
case            esac
if              then             fi
elif            else
while           until
{               }
(               )
```

項目	說明
for*Identifier* [**in***Word. . .*] **do***List* **done**	將 *Identifier* 參數設定成 *Word* 參數所指定的字（一次一個），並執行在 *List* 參數中指定的指令。若您要省略 **in** *Word. . .*，則 **for** 指令會針對所設定的每一個位置參數執行 *List* 參數，而且處理作業會在所有位置參數都已使用過後結束。
case *Word* **in** *Pattern* [| *Pattern*] . . . **)** *List***;;** [*Pattern* [|*Pattern*] . . . **)** *List***;;**] **...esac**	在與符合 *Word* 參數值的第一個 *Pattern* 參數相關的 *List* 參數中執行指定的指令。在用於檔名替代的型樣中使用相同的符合字元表示法， 只有斜線 (/)、前導點 (.) 或接在斜線後面的點 (/.) 不需要明確地符合。

項目	說明
if *List* **then** *List* [**elif** *List* **then** *List*] . . . [**else** *List*] **fi**	執行 **if** 指令後的 *List* 參數中所指定的指令。如果指令傳回零結束值，shell 會執行第一個 **then** 指令後的 *List* 參數。否則，它會執行 **elif** 指令後的 *List* 參數（如果存在的話）。如果傳回的結束值為零，shell 會執行在下一個 **then** 指令後的 list 參數。如果指令傳回非零的結束值，shell 會執行 **else** 指令後的 *List* 參數（如果存在的話）。如果未執行 **else** *List* 或 **then** *List*，則 **if** 指令會傳回零結束值。
while *List* **do** *List* **done**	執行 **while** 指令後的 *List* 參數中所指定的指令。如果在 **while** *List* 中的最後一個指令之結束值為零，則 shell 會執行 **do** 指令後的 *List* 參數。它會以迴圈的方式繼續執行整個清單， 直到 **while** *List* 中的最後一個指令之結束值不為零為止。如果沒有執行 **do** *List* 中的任何一個指令， **while** 指令會傳回零結束值。
until *List* **do** *List* **done**	執行 **until** 指令後的 *List* 參數中所指定的指令。如果在 **until** *List* 中的最後一個指令之結束值不為零，則執行 **do** 指令後的 *List*。迴圈並繼續執行整個清單直到 **until** *List* 中之前一指令的結束值為零。如果沒有執行 **do** *List* 中的任何指令，則 **until** 指令會傳回零結束值。
(*List* **)**	執行 subshell 中的 *List* 參數中的指令。
{*List***;}**	在現行 shell 處理程序中執行 *List* 參數中的指令，而且不會啟動 subshell。
Name **() {** *List* **}**	定義一個由 *Name* 參數參照的函數。此函數主體是由 *List* 參數 所指定、在大括弧之間的指令清單。

Bourne shell 內建指令
特殊指令內建在 Bourne shell 中，並且在 shell 處理程序中執行。

除非另有指定，在指令沒有任何語法錯誤的情況下，輸出值都會寫入檔案描述子 1（標準輸出），且結束狀態為 0（零）。輸入及輸出的重新導向是被允許的。

下列特殊指令與其他內建指令有些不同：

```
:（冒號）     exec            shift

.（點）          exit            times
break         export          trap
continue      readonly        wait
eval          return
```

Bourne shell 以下列方式處理這些指令：

· 指令完成時，指令前之關鍵字參數指派清單仍然有效。

· I/O 重新導向是在參數指派之後處理。

· shell Script 的錯誤將導致 Script 停止處理。

相關參考
Bourne shell 內建指令的清單
以下列出 Bourne shell 內建的指令。

特殊指令說明
Bourne shell 提供下列特殊內建指令。

項目	說明
:	傳回一個零結束值。
. *File*	自 *File* 參數讀取並執行指令，然後返回。不要啟動 subshell。shell 使用 *PATH* 變數指定的搜尋路徑尋找包含指定檔案的目錄。

項目	說明
break [*n*]	自含括的 **for**、**while** 或 **until** 指令迴圈跳出（如果有的話）。如果您指定 *n* 變數，則 **break** 指令會岔斷 *n* 變數指定的層次數目。
continue [*n*]	回復含括的 **for**、**while** 或 **until** 指令迴圈的下一個疊代。如果您指定 *n* 變數，則指令會在第 *n* 個含括迴圈處回復。
cd *Directory*]	將現行目錄變更為 *Directory*。如果您不指定 *Directory*，則使用 *HOME* shell 變數值。*CDPATH* shell 變數會定義 *Directory* 的搜尋路徑。*CDPATH* 為 用冒號分隔的替代目錄名稱清單。空的路徑名稱指定現行目錄（預設路徑）。此空的路徑名稱會立即出現在指派的等號之後，或冒號定界字元之間的任何位置，或是路徑清單中。如果 *Directory* 是以斜線 (/) 開頭，shell 就不會使用搜尋路徑。否則，shell 將會 在 *CDPATH* shell 變數中搜尋每一個目錄。 **註 :** 受限制的 shell 無法執行 **cd** shell 指令。
echo *String* . . .]	將字元字串寫入標準輸出。請參閱 **echo** 指令，以取得用法及參數資訊。**-n** 旗標未受支援。
eval [*Argument* . . .]	讀取作為輸入至 shell 的引數然後執行結果指令或指令。
exec [*Argument* . . .]	執行 *Argument* 參數所指定的指令，以取代這個 shell，而不建立新處理程序。可能會出現輸入和輸出引數，如果沒有出現其他引數，則會導致 shell 輸入或輸出受到修改。 不建議在您的登入 shell 中執行此動作。
exit [*n*]	導致 shell 以 *n* 參數所指定的結束值來結束。 如果您省略此參數，則此結束值為前一指令執行後的值（ Ctrl-D 鍵順序亦導致 shell 結束）。*n* 參數的值介於 0 至 255（含）。
export [*Name* . . .]	標示指定的名稱以自動匯出至後來執行的指令環境中。如果您沒有指定 *Name* 參數，**export** 指令會顯示 shell 中所有匯出名稱的清單。您無法匯出函數名稱。
hash [**-r**] [*Command* . . .]	尋找並記住每個指定 *Command* 的搜尋路徑位置。**-r** 旗標會使 shell 忘記所有位置。如果您不指定這個旗標或任何指令，shell 將以下列格式顯示已記住指令的相關資訊： `Hits Cost Command` `Hits` 會指出 shell 處理程序已執行過某個指令的次數。`Cost` 為必要工作的計量以在搜尋路徑中尋找指令。`Command` 會顯示每個指定指令的路徑名稱。在某些狀況下，指令的儲存位置需要重新計算；例如，改變現行目錄時相對路徑名稱的位置。已完成的指令會在 `Hits` 資訊的旁邊用星號 (*) 來表示。當重新計算完成時，`Cost` 會遞增。
pwd	顯示現行目錄。請參閱 **pwd** 指令以取得指令選項的討論說明。
read [*Name* . . .]	從標準輸入讀取一行。將此行中的第一個字組指派給第一個 *Name* 參數，第二個 字組指派給第二個 *Name* 參數，其他的字組則指派給最後的 *Name* 參數。 除非指令發現一個檔案尾字元，否則其會傳回零值。
readonly [*Name* . . .]	將 *Name* 參數所指定的名稱標示成唯讀。名稱的值不能重設。如果您未指定任何 *Name*，則 **readonly** 指令會顯示所有唯讀名稱的清單。
return [*n*]	導致函數結束，並傳回 *n* 值。如果您不指定 *n* 變數，此函數會傳回在該函數中最後一個指令執行的狀態。此指令只在 shell 函數中執行有效。

項目	說明
set [*Flag* [*Argument*] . . .]	設定一個以上如下的旗標： **-a** 標示匯出所有變數至已執行指派作業的程序。如果指派作業在指令名稱之前，匯出屬性僅對指令執行環境 有效，除了其中一個特殊的內建指令之前外。在此情形下，匯出屬性在內建指令完成後會 持續作用。如果此指派不在指令名稱之前，或此指派作業為 **getopts** 或 **read** 指令 作業的結果，則匯出屬性會持續作用到變數取消設定為止。 **-e** 如果某個指令在所有下列條件下，立即結束： · 其結束時帶有一個大於零的傳回值。 · 其不是 **while**、**until** 或 **if** 指令之複合清單的一部分。 · 其不是使用 AND 或 OR 清單測試。 · 其不是前置 !（驚嘆號）保留字的管線。（驚嘆號位置）保留字。 **-f** 停用檔名替代。 **-h** 在定義函數時，尋找並且記得函數呼叫的指令。（一般來說，在執行此函數時，即會尋找這些指令；請參閱 **hash** 指令。） **-k** 將所有的關鍵字置於指令環境中，而不是只有在指令名稱之前的那此而已。 **-n** 讀取指令但不執行。若要檢查 shell Script 的語法錯誤，請使用 **-n** 旗標。 **-t** 在讀取和執行一個指令之後結束。 **-u** 執行變數替代時，將未設定之變數視為錯誤並立即結束。互動式 shell 不會結束。 **-v** 顯示讀取輸入時 shell 輸入行。 **-x** 在執行之前顯示指令及它們的引數。 — 不變更任何旗標。將 **$1** 位置參數設為以連字號 (-) 來開頭的字串會非常有用。 使用加號 (+) 而不是連字號 (-) 來解除設定旗標。您亦可在 shell 指令行指定這些旗標。 *$-* 特殊變數包含現行的旗標集。 **set** 指令的任何引數都會變成位置參數，並且會被依序指派為 $1、$2、**...**，依此類推。如果沒有指定 *flag* 或 *Argument*，**set** 指令會顯示所有現行 shell 變數的名稱及值。
shift [*n*]	將指令行引數移至左邊；亦即，藉由捨棄 **$1** 的現行值來重新指派位置參數的值，並指派 **$2** 的值給 **$1**，**$3** 的值給 **$2**，等等。如果有 9 個以上的指令行引數，則會將第 10 個指派給 **$9**，而剩下的指令行引數則仍然未指派（直到在另一個 **shift** 之後）。如果引數為 9 個或更少，**shift** 指令會取消設定最高編號之帶值位置參數。 **$0** 位置參數絕不會移位。**shift** *n* 指令為指定連續移位 *n* 數目 的速記表示法。*n* 參數的預設值為 1。
test *Expression* | [*Expression*]	評估條件表示式。請參閱 **test** 指令，以取得指令旗標及參數的討論說明。 **-h** 旗標不被 **bsh** 的內建測試指令所支援。

項目	說明
times	顯示執行 shell 之處理程序的使用者和系統累計次數。
trap [*Command*] [*n*] . . .	當 shell 收到由 *n* 參數所指定的信號時， 便會執行由 *Command* 參數所指定的指令。執行 **trap** 指令以代替信號數目。若要在進入現行 shell 時被忽略的信號上設定設陷，任何嘗試都是無效的。 **註 :** 在設定設陷時，shell 會掃描 *Command* Command 參數一次，並在採行設陷時，再掃描一次。 如果您不指定指令，則所有由 *n* 參數所指定的設陷會重設至它們的現行值。如果您 指定空字串，則 shell 及其呼叫的指令會忽略這個信號。如果 *n* 參數為零 (0)，則當您結束 shell 時會執行指定的指令。如果您不指定指令或信號，則 **trap** 指令會顯示 與每一個信號號碼相關的指令清單。
type [*Name* . . .]	指出 shell 要如何針對每個指定的 *Name*，將它解譯為指令名稱。
ulimit [**-HS**] [**-c** | **-d** | **-f** | **-m** | **-r** | **-s** | **-t** |**-u**] [*limit*]	顯示或調整配置的 shell 資源。shell 資源設定值可以個別顯示，也可以群組顯示。預設模式為以群組方式顯示資源設定為軟體設定，或下限。 shell 資源的設定是根據現行 shell 之有效的使用者 ID 而定。硬體層次僅可在現行 shell 的 使用者 ID 為 root 時設定。如果您不是 root 使用者，但嘗試設定硬體層次的資源，將會發生錯誤。就預設值而言，root 使用者設定部分資源的硬體及軟體限制。因此，root 使用者應該小心 使用 **-S**，**-H**，或限制設定的旗標之用法。 除非您是 root 使用者，否則您僅可設定資源的軟體限制。當非 root 使用者減少限制之後，即使回到原來的系統限制，亦無法增加。 欲設定資源限制，請選取適當的新資源旗標及限制值（應為整數）。您一次僅可設定一個 資源限制。如果指定一個以上的資源旗標，您將收到未定義的結果。就預設值而言，在指令行 中 **ulimit** 只有一個新值以設定 shell 的檔案大小。**-f** 旗標的使用是選用的。 您可以指定下列 **ulimit** 指令旗標： **-c** 設定或顯示 shell 之核心區段。 **-d** 設定或顯示 shell 之資料區段。 **-f** 設定或顯示 shell 的檔案大小。 **-H** 設定或顯示硬體資源限制（僅限 root 使用者）。 **-m** 設定或顯示 shell 的記憶體。 **-r** 設定或顯示每個處理程序的執行緒上限。 **-s** 設定或顯示 shell 之堆疊區段。 **-S** 設定或顯示軟體資源限制。 **-t** 設定或顯示 shell 的 CPU 時間最大值。 **-u** 設定或顯示每個使用者的處理程序上限。
umask [*nnn*]	決定檔案的許可權。在檔案建立時，此值與建立處理程序之許可權將決定檔案的許可權。預設值為 022。若未輸入任何值，**umask** 會顯示現行值。

項目	說明
unset [*Name . . .*]	移除由 *Name* 參數所指定的每個名稱之對應變數或函數。無法取消 *PATH*、*PS1*、*PS2*、 *MAILCHECK* 及 *IFS* shell 變數之設定。
wait [*n*]	等待由 *n* 參數指定程序號碼的子程序結束，然後傳回該程序的結束狀態。如果您不指定 *n* 參數，shell 會等待所有目前為作用中的子程序，且傳回值為 0。

Bourne shell 中的指令替代
指令替代可讓您擷取任何指令的輸出以作為其他指令的引數。

當您將指令行放在反引號 (` `) 之內時，shell 會先執行此指令， 然後用輸出內容來置換整個表示式（包含反引號）。此特性常用來提供 shell 變數值。例如，陳述式：

```
today=`date`
```

指派代表現行日期的字串給 *today* 變數。下列指派將儲存（在 *files* 變數中）現行目錄中的存檔數：

```
files=`ls | wc -l`
```

您可在任何寫入標準輸出的指令上執行指令替代。

若要重疊套入指令替代，請在每一個要套入的反引號之前加上反斜線 (\)，如下所示：

```
logmsg=`echo Your login directory is \`pwd\``
```

您亦可使用 **read** 特殊指令間接提供 shell 變數值。 此指令自標準輸入（通常是鍵盤）取得一行，並在指派該行之連續字組 給任何變數名稱。例如：

```
read first init last
```

取得此格式的某個輸入行：

```
J. Q. Public
```

與您鍵入下列字行具有相同的效果：

```
first=J. init=Q. last=Public
```

read 特殊指令指派任一多出來的字組至最後一個變數。

Bourne shell 中的變數替代
Bourne shell 允許您執行變數替代。

Bourne shell 具有數種建立變數的機制（將字串值指派給名稱）。一般而言，某些變數只在指令行上設定，包括位置參數和關鍵字參數。其他變數只是您或 shell 可指派字串值的名稱。

相關概念
無人式終端機
如果終端機已登入且無人操作，所有系統就很容易受到攻擊。當系統管理人員讓使用 root 授權啟用的終端機處於無人看管的狀態時，會發生最嚴重的問題。一般而言，使用者只要離開終端機，就應該要登出。

***Bourne shell* 中使用者定義的變數**
Bourne shell 會辨識可指派字串值的英數變數。

若要指派名稱的字串值，請鍵入：

```
Name=String
```

名稱是以底線或字母為開頭之字組的字母、數字和底線的順序排列。若要使用您已指派給變數的值，請在其名稱開頭加上一個錢幣符號 ($)。如此，*$Name* 變數會產生由 *String* 變數指定的值。請注意，在指派陳述式中的等號 (=) 兩邊皆不可存有空格。（位置參數不能出現在指派陳述式中。 您可在一指令行上放置一個以上的指派，但請記得 shell 由右向左來執行指派。

如果您以雙引號或單引號（" 或 '）來括住 *String* 變數， shell 就不會將字串中的空格、跳格字元、分號及換行字元視為定界字元，而是逐字地將它們嵌入字串內。

如果您以雙引號 (") 括住 *String* 變數，則 shell 仍會辨識字串中的變數名稱，並執行變數替代；亦即，以對應值來取代以錢幣符號 ($) 為開頭的位置參數參照及其他變數名稱（若有的話）。shell 亦於雙引號括住的字串內執行指令替代。

如果使用單引號 (') 括住 *String* 變數，則 shell 不會替換字串中的變數或指令。下列順序說明這種差異：

```
您：             num=875
                                number1="Add $num"
                                number2='Add $num'
                                echo $number1
系統:            Add 875
您:              echo $number2
系統：           Add $num
```

在變數替代之後，shell 不會重新解譯指派中的空白。因此，下列指派會導致 `$first` 及 `$second` 具有相同的值：

```
first='a string with embedded blanks'
second=$first
```

當您參照某個變數時，您可以使用 `{ }` 來括住變數名稱（或指定位置參數的數字），以劃定變數名稱與下列任一字串的界限。尤其，若緊跟著名稱的字元是字母、數字或底線，而且變數不是位置參數的話，則需要大括弧：

```
您：             a='This is a'
                                echo "${a}n example"
系統:            This is an example
您:              echo "$a test"
系統：           This is a test
```

相關概念

Bourne shell 中的位置參數
當執行 shell 程序時，此 shell 會悄悄地建立位置參數，這些參數會依據其在指令行的位置來參照每一個字組在指令行上的位置。

相關參考

Bourne shell 中的條件性替代
一般來說，shell 會以指派給 *Variable* 變數的字串值來取代表示式 $*Variable*（若有的話）。然而，根據變數是否已設定或非空值，或兩種情況皆存在，有一種特殊的表示法可允許條件性替代。

Bourne shell 使用的變數
shell 會使用下列變數。雖然 shell 會設定部分變數，您仍可設定或重設變數。

項目	說明
CDPATH	指定 **cd**（變更目錄）指令的搜尋路徑。
HOME	指出登入目錄的名稱， 該目錄在完成登入之後會成為現行目錄。`login` 程式會起始設定此變數。**cd** 指令會使用 *$HOME* 變數的值作為其預設值。在 shell 程序中使用此變數而非明確的路徑名稱，可不需切換目錄即可從另一個目錄執行程序。
IFS	屬於 IFS（內部欄位分隔字元）的字元，shell 會在空白解譯期間使用這些字元。shell 初始設定 *IFS* 變數來包括空白、跳格字元和換行字元。
LANG	當 *LC_ALL* 變數和對應的環境變數（以 `LC_` 為開頭）未指定語言環境時，可決定用於語言環境種類的語言環境。
LC_ALL	決定一個語言環境，用來置換由 *LANG* 環境變數或以 `LC_` 開頭之任何環境變數設定所指定的語言環境種類之任何值。
LC_COLLATE	定義當排序名稱和型樣中出現字元範圍時使用的對照順序。

項目	說明
LC_CTYPE	決定將文字資料的位元組序列解譯為字元時（亦即，引數和輸入檔中的單位元組字元對多位元組字元）所用的語言環境、 哪些字元被定義成字母（**alpha** 字母類別），以及在型樣相符中的字元類別行為。
LC_MESSAGES	決定撰寫訊息的語言。
LIBPATH	指定共用檔案庫的搜尋路徑。
LOGNAME	指定登入名稱，在 `/etc/profile` 檔案中標示為 `readonly`。
MAIL	指出郵件系統用來偵測新郵件的檔案之路徑名稱。如果有設定此變數，則 shell 會定期檢查此檔案的修改時間；如果時間變更，且檔案的長度大於 0，則會顯示 `$MAILMSG` 的值。請在 `.profile` 檔案中設定 *MAIL* 變數。一般而言，由 **mail** 指令的使用者所指派給它的值是 `/usr/spool/mail/$LOGNAME`。
MAILCHECK	shell 在 *MAILPATH* 或 *MAIL* 變數指定的檔案內檢查新郵件的間隔秒數。 預設值是 600 秒（10 分鐘）。如果您設定 *MAILCHECK* 變數為 0，則 shell 會在每一個提示之前先作檢查。
MAILMSG	郵件通知訊息。如果明確地將 *MAILMSG* 變數設定為空字串 (`MAILMSG=""`)，則不會顯示訊息。
MAILPATH	由冒號隔開的檔名清單。若設定此變數，則當清單中指定的任何檔案內有新郵件時，shell 會通知您。您可以在每一個檔名後面 加上一個 %，則有新郵件時會顯示訊息。否則，shell 會使用 *MAILMSG* 變數的值，或預設的訊息 `[YOU HAVE NEW MAIL]`。 **註：** 若有設定 *MAILPATH* 變數時，就會檢查這些檔案，而非檢查由 *MAIL* 變數設定的檔案。若要檢查由 *MAILPATH* 變數設定的檔案以及由 *MAIL* 變數設定的檔案，請在 `MAILPATH` 檔案清單中指定 `MAIL` 檔案。
PATH	指令的搜尋路徑，是以冒號隔開且依序的目錄路徑名稱清單。當 shell 尋找指令時，會依指定順序來搜尋這些目錄。清單中的空字串代表現行目錄。 *PATH* 變數通常是在 `/etc/environment` 檔案中起始設定，一般會設定為 `/usr/bin:/etc:/usr/sbin:/usr/ucb:/usr/bin/X11:/sbin`。您可重設此變數來符合您的需求。在 `.profile` 檔案中提供的 *PATH* 變數，也會包括 `$HOME/bin` 及您的現行目錄。 如果您有專案特有的指令目錄，例如 `/project/bin`，您希望在標準系統目錄之前先搜尋，請設定 *PATH* 變數，如下所示： `PATH=/project/bin:$PATH` 將 *PATH* 變數設定成非預設值的最佳位置為 `$HOME/.profile` 檔案。若正在受限制的 shell 之下執行指令，則無法重設 *PATH* 變數。
PS1	要作為主要系統提示的字串。當互動式 shell 等待輸入時會顯示此提示字串。就非 root 使用者而言，*PS1* 變數的預設值為 $ 再加上一個空格。
PS2	次要提示字串的值。當 shell 在其輸入中遇到換行字元時，如果期待更多的輸入，則會以 *PS2* 變數值來提示。*PS2* 變數的預設值為 >，其後接著一個空格。
SHACCT	您擁有的檔案之名稱。若設定此變數，則 shell 會針對每一個執行的 shell Script 在檔案中寫入一筆帳戶記錄。您可以使用統計作業程式， 例如 `acctcom` 及 `acctcms`，來分析收集的資料。
SHELL	shell 的路徑名稱，保留在環境中。此變數應由每一個限制登入的 `$HOME/.profile` 檔案來設定及匯出。

項目	說明
TIMEOUT	一個 shell 結束之前維持非作用中狀態的分鐘數。如果該變數 設定成一個大於 0（零）的值，那麼在發出 PS1 提示之後，於規定的秒數之內未輸入指令的話， shell 將會結束。（請注意，可使用不超出此值的最大界限來編譯 shell。）0 這個值表示無時間限制。

相關概念

空白解譯

shell 執行變數及指令替代之後，它會掃描結果以尋找內部欄位分隔字元（定義於 *IFS* shell 變數中）。

Bourne shell 中預先定義的特殊變數

有數個變數具有特殊的意義。下列變數只能由 Bourne shell 設定：

項目	說明
$@	展開以 **$1** 開頭的位置參數。以一個空格隔開每一個參數。 如果您在 **$@** 兩側加上雙引號 (" ")，則 shell 會將每一個位置參數視為個別的字串。若沒有位置參數，則 Bourne shell 會將陳述式展開成無引號的空字串。
$*	展開以 **$1** 開頭的位置參數。shell 會使用 *IFS* 變數值的第一個字元來隔開每一個參數。 如果您在 **$*** 兩側加上雙引號 (" ")，則 shell 會以雙引號括住位置參數值。由 *IFS* 變數的第一個字元隔開每一個值。
$#	指定傳送至 shell 的位置參數數目，不計算 shell 程序名稱本身。因此，*$#* 變數會產生已設定之最高編號位置參數的數目。此變數的其中一個主要作用是檢查是否有必要的引數數目。透過 shell 只能存取位置參數 **$0** 到 **$9**。
$?	指定上次執行之指令的結束值。它的值是一個十進位字串。大部分指令會傳回一個 0 值來表示順利完成。shell 本身會傳回 *$?* 變數的現行值作為它的結束值。

$$	定義現行處理程序的程序號碼。因為程序號碼在全部現存處理程序中是唯一的，所以此字串經常用來產生暫用檔的唯一名稱。
	下列範例說明建議您練習的動作：僅為此目的之目錄中建立暫用檔：
	`temp=/tmp/$$`
	`ls >$temp`
	`.`
	`.`
	`.`
	`rm $temp`

$!	使用 & 終止符號來指定在背景中執行的最後一個處理程序的號碼。
$-	由目前設定於 shell 的執行旗標名稱所組成的字串。

相關概念

Bourne shell 中的位置參數

當執行 shell 程序時，此 shell 會悄悄地建立位置參數，這些參數會依據其在指令行的位置來參照每一個字組在指令行上的位置。

空白解譯

shell 執行變數及指令替代之後，它會掃描結果以尋找內部欄位分隔字元（定義於 *IFS* shell 變數中）。

shell 會在它找到一或多個這些字元的每一個位置上，將此行分割成一些不同的字組，並以單一空格來隔開每一個不同的字組。shell 也會保留明確的空值 ("" 或 '')，並且會捨棄隱含的空值引數（那些空值引數是由沒有值的引數所造成的）。

相關參考

Bourne shell 使用的變數

shell 會使用下列變數。雖然 shell 會設定部分變數，您仍可設定或重設變數。

C shell

C shell 是一種互動式指令直譯器，也是一種指令程式設計語言。它使用的語法和 C 語言相似。

C shell 由 **csh** 指令來起始。

當您登入時，**csh** 指令會先搜尋全系統設定檔 /etc/csh.cshrc。如果設定檔存在，C shell 便會執行儲存在該檔案中的指令。之後，如果全系統設定檔 /etc/csh.login 是可用的，C shell 便會執行它。 接著，它會在起始目錄中搜尋 .cshrc 和 .login 檔案。如果它們存在，其中會包含與執行 C shell 相關的任何自訂使用者資訊。設定在 /etc/csh.cshrc 和 /etc/csh.login 檔案中的所有變數，可能會被您的 $HOME 目錄中的 .cshrc 和 .login 檔案置換。只有 root 使用者可以修改 /etc/csh.cshrc 及 /etc/csh.login 檔案。

/etc/csh.login 及 $HOME/.login 檔案都只在登入時執行一次。這些檔案通常是用來存放環境變數定義、您要在登入時執行一次的指令，或是設定終端機特性的指令。

/etc/csh.cshrc 及 $HOME/.cshrc 檔案會在登入時，以及每次呼叫 **csh** 指令或 C shell Script 時執行。它們通常都用來定義 C shell 的性質，如別名和 C shell 變數（例如，*history*、*noclobber* 或 *ignoreeof*）。建議您只使用 /etc/csh.cshrc 及 $HOME/.cshrc 檔案中的 C shell 內建指令，因為使用其他指令會增加 shell Script 的啟動時間。

相關參考

C shell 內建指令清單

以下是 C shell 的內建指令。

C shell 限制

下列是 C shell 的限制。

- 字組不能超出 1024 位元組。
- 引數清單的限制為 *ARG_MAX* 個位元組。*ARG_MAX* 變數的值位於 /usr/include/sys/limits.h 檔案中。
- 包含副檔名的指令所能使用的引數個數，限制為引數清單中所允許之位元組數的 1/6th。
- 指令替代所能替代的，不能超出引數列所允許的位元組數。
- 為偵測迴圈，shell 將單行的別名替代數限制為 20。
- **csh** 指令不支援以同等字元分類為基礎的檔名擴充。
- 在 **csh** 執行任何應用程式之前即已開啟的檔案描述子（不同於標準輸入、標準輸出及標準錯誤），無法用於該應用程式。

C shell 中的別名替代

別名即指派給指令或指令字串的名稱。C shell 可讓您指派別名並隨意下指令使用它們。shell 會維護您所定義的別名清單。

當 shell 掃描過指令行後，會將指令行分成不同的字組，並由左到右檢查每一指令的第一個字組，檢查是否有別名。若發現別名，shell 會使用歷程機制，將別名的文字置換成別名所參考的指令文字。結果的字組置換指令及引數清單。如果歷程清單沒有參考，則引數清單不會改變。

alias 及 **unalias** 內建指令會建立、顯示與修改別名清單。以下列格式使用別名指令：

```
alias [Name [WordList]]
```

選用性 *Name* 變數會指定已指定之名稱的別名。如果您使用 *WordList* 變數來指定字組清單，則指令會指派它為 *Name* 變數的別名。如果您沒有使用其中任一個選用性變數來執行 **alias** 指令，便會顯示所有的 C shell 別名。

如果 **ls** 指令的別名為 **ls -l**，則下列指令：

```
ls /usr
```

會被下列指令取代：

```
ls -l /usr
```

引數清單不受影響，理由是具有別名的指令中沒有參考到歷程清單。同樣地，如果 **lookup** 指令的別名如下：

```
grep \!^ /etc/passwd
```

則 shell 會以下列指令行來取代 `lookup bill`：

```
grep bill /etc/passwd
```

在此範例中，`!^` 是指歷程清單， shell 會使用輸入行的第一個引數來取代它，在此例中為 `bill`。

您可以在別名中使用特殊的型樣相符字元。下列指令：

```
alias lprint 'pr &bslash2.!* >
> print'
```

會建立一個可格式化其引數到單行印表機的指令。! 字元在別名中被 shell 使用單引號保護，所以該別名不會展開，直到執行 **pr** 指令為止。

如果 shell 尋找別名，則它執行輸入文字的字組轉換，並且開始在修改的輸入行上，重新開始別名處理程序。如果下一段文字的第一個字與前一段文字相同， 便會將別名標上旗標來終止別名處理程序，以防止循環。其他後續的循環會被偵測，然後導致錯誤。

相關概念

C shell 中的歷程替代

歷程替代可讓您修改上一個指令的個別字組以建立新的指令。歷程替代讓它輕易重複指令、重複在目前指令中上一個指令的引數，或是修訂在上一個指令中鍵入的拼字錯誤。

C shell 中的變數替代

C shell 維持一組變數，每一個變數都有自己的值，這個值可能是一列 0 或多個字組。由 shell 設定或參照一部分這些變數。例如，*argv* 變數是 shell 變數清單的影像，而構成此變數值的字組會以特殊方式來參照。

若要變更及顯示變數值，請使用 **set** 及 **unset** 指令。在 shell 參照的變數中，有一些是輪換（用來開啟及關閉的變數）。shell 查不出值的輪換， 只有查出它們是設定或取消設定。例如，*verbose* shell 變數是一個會使指令輸入產生回應的輪換。在指令行發出 **-v** 旗標而產生此變數的設定。

其他作業以數字處理變數。**@** 指令執行數值計算，而且結果是指派給變數的。然而一定以字串（零或更多）來表現變數值。就數值運算而言，空字串被視為零，並且忽略多字值的第二個字和後續字組。

發出某個指令之後，shell 會剖析輸入行並執行別名替代。接下來它會執行變數替代之後才執行指令。$ 字元鍵替代。然而，其後若接著空格、跳格字元或換行字元，則略過並維持不變。除了下列兩種情況以外，在 $ 字元之前加上 \（反斜線）可防止這種擴充：

- 指令以 " " 來含括。在此情形下，shell 一定執行替代。
- 指令以 ' ' 來含括。在此情形下，shell 不執行替代。以 '' 含括的字串會被解譯，執行指令替代

擴充變數之前，shell 會辨識輸入和輸出重新導向，然後各別地擴充每一個變數。否則，會同時擴充指令名稱和完整引數清單。因此第一個（指令）字有可能產生一個以上的字， 第一個字會變成指令名稱，其餘的字則變成參數。

除非以 " " 括住，或提供 :q 修飾元，否則變數替代的結果最後可能會附屬於指令和檔名替代。被雙引號括住時，具有一個含有多字的值的變數會擴充為單一字組或單一字組的一部分，同時由空白隔開變數值的字組。將 :q 修飾元應用於某個替代時，變數會擴充成多個字。以空格隔開每一個字，並且用雙引號括住該字，以防止被後面的指令或檔名替代。

下列表示法可讓您將變數值導入 shell 輸入。除了上述情況以外，參照一個不是使用 **set** 指令設定的變數是錯誤的。

您可以將修飾元 :gh、 :gt、:gr、:h、 :r、 :q 和 :x 應用於下列替代中。如果 { }（大括弧）出現在指令表格中，則修飾元必須放在大括弧內。每個變數擴充上只能有一個 **:** 修飾元。

項目	說明
$*Name*	
${*Name***}**	會被指派給 Name 變數的字組所取代，以空格隔開每一個字組。成對的大括弧將 *Name* 變數與下列所有字元隔開；若未隔開的話，下列字元會成為變數的一部分。shell 變數名稱以字母作為開頭，且最多可包含 20 個字母和數字，其中包括底線字元 (_)。若 Name 變數沒有指定 shell 變數，但已設定於此環境，則會傳回它的值。前面加上冒號的修飾元以及此處說明的其他格式不適用於這種情況。
$*Name***[***number***]**	
$ { *Name*[*number*]}	僅選取 *Name* 變數值中的部分字組。數字受限於變數替代，並且可能包含單一數字或由連字號 (-) 隔開的兩個數字。 變數字串值的第一個字是編號 1。若省略某範圍的第一個數字，則會預設成 1。若省略某範圍的最後一個數字，則會預設成 **$#***Name*。星號 (*) 符號會選取全部字組。若省略第二個引數或它是位於某範圍，則空的範圍不算錯誤。
$#*Name*	
${#*Name***}**	在 *Name* 變數中提供字數。這可以用在 **[***number***]** 中，如上述。例如，$*Name*[$*#Name*]。
$0	替換從它那裡讀取指令輸入的檔名。若不知道此檔名則會發生錯誤。
$*number*	
${*number***}**	相當於 $argv[*number*]。
$*	相當於 $argv[*]。

下列的替代可能不會隨著 **:** 修飾元一起改變：

項目	說明
$?*name*	
${?*name***}**	若已設定 *name* 變數，則替換字串 1；若未設定此變數，則替換字串 0（零）。
$?0	若已知現行輸入檔名稱，則會替換 1；若不知道檔案名稱，則替換 0（零）。
$$	替換母項 shell 的（十進位）處理號碼。
$<	替換標準輸入的某一行，但沒有進一步解譯。在 shell 程序中使用此替代來從鍵盤讀取。

相關概念

C shell 中的指令替代

在指令替代中，shell 將執行指定的指令，然後以其輸出置換該指令。

C shell 中的檔名替代

C shell 允許執行檔名替代。

C shell 提供數個捷徑，可節省時間和按鍵。若字組中包含 *、?、[ ] 或 { } 中的任何一個字元，或是以顎化符號 (~) 作為開頭，則該字組可作為檔名替代的候選。C shell 將此字視為一個型樣，並以符合該型樣且按字母順序排列的檔名清單來置換此字。

依 *LC_COLLATE* 或 *LANG* 環境變數的指定，使用現行對照順序。在指定檔名替代的字組清單中，若無型樣符合現存的檔名，則會發生錯誤。然而，並不需要每一個型樣都符合。只有字元相符的符號 *、? 及 [ ] 表示型樣相符或檔名擴充。顎化符號 (~) 及 { } 字元則表示檔名縮寫。

C shell 中的副檔名

星號 (*) 字元，符合任何字串，包括空字串。

例如，在一個含有下列檔案的目錄中：

```
a aa aax alice b bb c cc
```

echo a* 指令會列印所有以 a 字元開頭的檔名：

```
a aa aax alice
```

註： 當比對檔名時，字元點 (.) 及斜線 (/) 必須完全相符。

問號 (?) 字元，符合任何單一字元。下列指令：

```
ls a?x
```

清單以字母 a 為開頭，其後跟隨著一個字元，並以字母 x 為結尾的每一個檔名：

```
aax
```

若要符合單一字元或某字元範圍，請使用 [ ] 來括住字元。下列指令：

```
ls [abc]
```

清單完全符合任一個含括字元的所有檔名：

```
a b c
```

在方括弧內，由 [a-z] 指出字元的詞彙範圍。由現行對照順序定義符合此型樣的字元。

C shell 中的檔名縮寫

顎化符號 (~) 及 { 字元表示檔名縮寫。位在檔名開頭的 ~ 可用來代表起始目錄。單獨的 ~ 字元可展開成您的起始目錄，如同 *home* shell 變數值反應的結果。

例如，下列指令：

```
ls ~
```

清單位於 $HOME 目錄中的全部檔案和目錄。

當指令後面接著由字母、數字和連字號 (-) 字元所組成的名稱時， shell 會搜尋具有該名稱的使用者，並且替換該使用者的 $HOME 目錄。

註： 如果 ~ 字元後面跟著的不是字母或斜線 (/)，或出現在字組開頭以外的地方，它就不會展開。

若要符合檔名中的字元又不鍵入整個檔名，請使用 { } 來括住檔名。型樣 a{b,c,d}e 是另一種撰寫 abe ace ade 的方式。shell 保留由左向右順序，並且將符合的結果各別儲存在一個低層次以保留此順序。它可能是巢狀建構。因此，下列：

```
~source/s1/{oldls,ls}.c
```

展開成：

```
/usr/source/s1/oldls.c /usr/source/s1/ls.c
```

如果 **source** 的起始目錄為 /usr/source。同樣地，下列：

```
../{memo,*box}
```

可能會展開成：

```
../memo ../box ../mbox
```

註： memo 不是使用符合 *box 的結果來排序。這是一個特殊情況，會略過 { 、 } 及 { } 字元而維持不變。

C shell 中的字元類別

您可以使用字元類別來比對範圍指示內的檔名。

下列格式會指示系統來比對屬於指定類別的任何單一字元：

```
[:charclass:]
```

下列類別對應於 ctype 子常式：

字元類別	定義
alnum	英數字元
alpha	大寫與小寫字母
cntrl	控制字元
digit	數字
graph	圖形字元
lower	小寫字母
print	可列印字元
punct	標點字元
space	空格、水平欄標、換行字元、換行、垂直欄標或換頁字元
upper	大寫字元
xdigit	十六位數

假設您位於包含下列檔案的目錄中：

```
a aa aax Alice b bb c cc
```

請在 C shell 提示下鍵入下列指令：

```
     ls [:lower:]
```

C shell 會列出以小寫字元為開頭的所有檔名：

```
a aa aax b bb c cc
```

如需字元類別的詳細資訊，請參閱 **ed** 指令。

C shell 中的環境變數

對 C shell 而言，某些變數具有特殊的意義。當然，*argv*、*cwd*、*home*、*path*、*prompt*、*shell* 與 *status* 一定是由 shell 所設定。

除了 *cwd* 與 *status* 變數外， shell 設定的動作只會在起始設定時發生。上面的所有變數都將維持其設定，除非您明確地重設它們。

csh 指令會將 *USER*、*TERM*、*HOME* 及 *PATH* 等環境變數，分別複製至 *csh* 變數 *user*、*term*、*home* 及 *path*。不論何時一般 shell 變數重設，該數值將複製回環境中。*path* 變數不可以設定在 **.cshrc** 檔案以外的地方，因為 **csh** 次處理程序會從環境匯入路徑定義，有變更時，再重新匯出。

下列變數具有特殊的意義：

項目	說明
argv	包含傳送至 shell Script 的引數。位置參數將自此變數中替換。
cdpath	藉由 **chdir** 或 **cd** 指令，指定將搜尋的替代目錄清單，以尋找子目錄。
cwd	指定現行目錄的完整路徑名稱。
echo	在使用 **-x** 指令行旗標時設定，設定時，將導致每一指令與其引數在執行之前產生回應。對不是內建的指令而言，所有的擴充都將在回應前發生。內建指令將在指令與檔案名稱替代前回應，因為這些替代作業是選擇性完成的。
histchars	指定字串值，變更使用在歷程替代的字元。使用其值的第一個字元作為歷程替代字元， 這會置換預設字元！。其值的第二個字元會快速置換 ^ 字元。 **註 :** 將 **histchars** 值設定成指令或檔名中所使用的字元，可能會導致意外的歷程替代。
history	包含一控制歷程清單大小的數值。在所認可之事件數量中參照的任何指令都不會被捨棄。非常大的 *history* 變數值可能導致 shell 的執行發生記憶 體不足情形。不論此變數是否已設定，C shell 將一定會儲存在歷程清單中最後執行的指令。
home	表示從環境中起始設定的起始目錄。顎化符號 (~) 字元的檔名擴充會參照此變數。
ignoreeof	指定 shell 將忽略工作站輸入裝置的檔案尾字元。如此可防止 shell 在讀取檔案結尾字元 (ctrl-D) 時被意外刪除。
mail	指定 shell 在其中檢查郵件的檔案。在每一指令完成後，即可完成此作 業；如果經歷過指定的時間間隔，並將產生一提示。如果檔案存在的存取時間少於其變更時間，shell 便會顯示下列訊息：檔案中有郵件。 如果 *mail* 變數值的第一個字是數字， 就會指定不同的郵件檢查時間間隔（以秒計算），其預設值為 600（10 分鐘）。如果您指定多重郵件檔案，當指定的檔案中有郵件時，shell 便會顯示下列訊息：檔案中有新郵件。
noclobber	限制輸出重新導向，以確保檔案不致意外受損，並確保重新導向附加至現存的檔案中。
noglob	抑制檔名擴充。在不處理檔名的 shell Script 中，或是已取得檔名清單且不需進一步擴充時，這是最有用的方法。
nonomatch	如果檔名擴充不符任何現存的檔案，則指定無錯誤結果；反之，則傳回 初始值型樣。如果初始值型樣不當形成，仍是一個錯誤。
notify	指定 shell 傳送非同步工作狀態變更通知。預設值將在顯示 shell 提 示前，呈現狀態變更。
path	指定發現指令執行的目錄。空值字組指定現行目錄。如果沒有設定 *path* 變數，就只有完整的路徑名稱可以執行。預設搜尋路徑（從在登入期間使用的 `/etc/environment` 檔案開始）如下： `/usr/bin /etc /usr/sbin /usr/ucb /usr/bin/X11 /sbin` 既不提供 **-c** 也不提供 **-t** 旗標的 shell，在讀取 `.cshrc` 之後與每次重設 *path* 變數時， 通常會雜湊 *path* 變數中的目錄內容。新的指令如果在 shell 作用中時，將新的指令加入這些目錄，您就必須提供 **rehash** 指令。否則可能會找不到該指令。
prompt	指定在自互動式工作站輸入中讀取每一指令之前顯示的字串。如果！出現在字串中，它會被現行事件編號所置換。如果！字元是在引號內的字串（而此字串被單引號或雙引號所含括），則必須在！字元的前面加上 \。沒有 root 授權的使用者之預設提示字元為 %。具有 root 授權之使用者的預設提示字元為 #。
savehist	指定一個數值，以控制歷程清單的項目數（在您登出時，該歷程清單會儲存在 `~/.history` 檔案中）。任何參照此事件數目的指令都將儲存。在啟 動期間，shell 將讀取歷程清單中的 `~/.history`，讓歷程在登入時儲 存。非常大的 *savehist* 變數值將減緩 shell 的啟動速度。

項目	說明
shell	指定 C shell 所處的檔案。此作業是使用在 fork shell 中，以解譯 具有執行位元組，但無法由系統執行的檔案。此作業將起始設定於 C shell 的 起始位置。
status	指定由前一指令所傳回的狀態。如果指令異常結束，會將 0200 新增至狀態。 不成功的內建指令將傳回結束狀態 1，成功的內建指令則設定狀態為 0 值。
time	控制指令自動計時。如果設定此變數，任何使用超過指定 CPU 秒數的指令，都會在執行結束時顯示一行已使用的資源。請參閱內建 **time** 指令，以取得預設輸出的相關資訊。
verbose	此變數是由 **-v** 指令行旗標所設定，它會在歷程替代之後顯示出每個指令的字詞。

C shell 中的工作控制

shell 會與每一個處理程序連結一個工作號碼。shell 會保留現行工作的表格，並指派小整數給它們。

當您使用 & 符號 (&) 在背景啟動工作時，shell 會印出類似下列的字行：

```
[1] 1234
```

此行指出工作號碼為 1，且該工作是由處理程序 ID 為 1234 的單一處理程序所組成。請使用內建的 **jobs** 指令察看現行工作的表格。

如果嘗試從工作站讀取，則一個執行於背景的工作會與輸入競爭。背景工作也會產生與其他工作輸出交錯的工作站輸出。

您可以用好幾種方法來參照 shell 中的工作。使用百分比 (%) 字元來介紹工作名稱。這個名稱可以是啟動的工作號碼或是指令名稱（如果這個名稱是唯一的話）。例如，若 **make** 處理程序是當成工作 1 來執行，您可以將它參照為 %1。如果只有一個暫停工作，且其名稱的開頭具有 make 字串，則您也可以將它參照為 %make。您也可以使用：

```
%?String
```

以指定名稱含有 *String* 變數的工作（如果只有一項此類工作的話）。

每當有處理程序變更其狀態，shell 就會立刻偵測。如果某個工作成為暫停執行以致於下一步的進度無法繼續，則 shell 會傳送訊息至工作站。您要按 Enter 鍵之後才會顯示此訊息。然而，如果設定了 *notify* 的 shell 變數，則 shell 會立即發出背景工作狀態變更的指示訊息。使用內建的 **notify** 指令來標記單一處理程序，以致於它的狀態變更會迅速的報告出來。根據預設，**notify** 指令會標記現行的處理程序。

C shell 內建指令清單

以下是 C shell 的內建指令。

項目	說明
@	顯示指定的 shell 變數值。
alias	顯示指定的別名或所有的別名。
bg	將現行或指定的工作放入背景。
break	在最接近的含括 **foreach** 或 **while** 指令結束之後，回復執行。
breaksw	自 **switch** 指令岔斷。
case	定義 **switch** 指令中的標籤。
cd	將現行目錄變更至指定的目錄。
chdir	將現行目錄變更至指定的目錄。
continue	繼續執行最接近的含括 **foreach** 或 **while** 指令。
default	標示 **switch** 陳述式中的預設情況。

項目	說明
dirs	顯示目錄堆疊。
echo	將字串寫入 shell 的標準輸出。
else	執行 `if` （*Expression*） `then` ...`else if` （*Expression2*） `then` ... `else` ... `endif` 序列中， 第二個 `else` 後面的指令。
end	表示前置 **foreach** 指令的指令順序結束。
endif	執行 `if` （*Expression*） `then` ... `else if` (*Expression2*） `then` ... `else` ... `endif` 指令順序後面， 第二個 `then` 陳述式後面的指令。
endsw	標示 **switch** (*String*) `case` *String* `:` ... `breaksw default:` ... `breaksw endsw` 指令順序的結尾。此指令序列根據 *String* 變數的值來連續與每一個 case 標籤相比對。若執行 **breaksw** 指令，或是如果沒有任何標籤相符，而且沒有預設值，則繼續執行 **endsw** 之後的指令。
eval	讀取變數值作為 shell 的輸入，並且在現行 shell 的上下文中執行結果指令。
exec	執行指定的指令來代替現行 shell。
exit	以狀態 shell 變數的值或指定表示式的值來結束 shell。
fg	將現行或指定的工作帶至前景，繼續執行原本已停止的工作。
foreach	連續設定一個 *Name* 變數給 *List* 變數所指定的每個成員，並設定指令順序，直到遇到 **end** 指令為止。
glob	使用歷程、變數及檔名擴充來顯示清單。
goto	在指定行之後繼續執行。
hashstat	顯示統計資料，指出 hash 表如何順利完成尋找指令。
history	顯示歷程事件清單。
if	若表示式為真，則執行指定的指令。
jobs	列出作用中的工作。
kill	將 **TERM**（終止）信號或 *Signal* 變數所指定的信號傳給指定的工作或處理程序。
limit	對於現行處理程序及其建立的每一個子程序，限制其所用的指定資源。
login	結束登入 shell，並以 `/usr/sbin/login` 指令的案例來將其置換。
logout	結束登入 shell。
nice	設定 shell 中執行之指令的優先順序。
nohup	針對某個程序的其餘部分，使掛斷被忽略。
notify	當現行或指定的工作狀態改變時，使 shell 以非同步方式通知您。
onintr	控制 shell 岔斷動作。
popd	蹦現目錄堆疊，並且返回新的頂端目錄。
pushd	交換目錄堆疊的元素。
rehash	導致重新計算內部 hash 表，該表包含路徑 shell 變數中的目錄內容。
repeat	以指定次數來執行指定的指令，且條件與 **if** 指令的限制相同。
set	顯示全部 shell 變數的值。
setenv	修改指定的環境變數值。
shift	將指定的變數往左邊移位。
source	讀取由 *Name* 變數所指定的指令。

項目	說明
stop	停止背景中執行的現行或指定工作。
suspend	若收到 **STOP** 信號，則停止 shell。
switch	開始 **switch** (*String*) case *String* : ... breaksw default: ... breaksw endsw 指令順序。此指令序列根據 *String* 變數的值來連續與每一個 case 標籤相比對。如果在發現預設標籤前無符合的標籤，則依預設值標籤執行。
time	顯示 shell 及其子程序所使用的時間總結。
umask	決定檔案的許可權。
unalias	捨棄所有符合 *Pattern* 變數的 alias，
unhash	停用內部 hash 表，以尋找執行中的程式。
unlimit	移除資源限制。
unset	移除名稱與 *Pattern* 變數相符的所有變數。
unsetenv	從環境中移除名稱與所指定之 *Pattern* 變數相符的所有變數。
wait	等待所有背景 工作。
while	當 *Expression* 變數所指定的表示式評估為非零值時， 評估 while 和相符之 end 指令順序之間的指令。

以下是相關資訊：

Korn shell

ksh 及 **stty** 指令。

alias、**cd**、**export**、**fc**、**getopts**、**read**、**set** 及 **typeset** Korn shell 指令。

/etc/passwd 檔案。

Bourne Shell

bsh 或 **Rsh** 指令，**login** 指令。

Bourne shell **read** 特殊指令。

The setuid 子常式，setgid 子常式。

null 特殊檔案。

environment 檔案，profile 檔案格式。

C shell

csh 指令、**ed** 指令。

alias、**unalias**、**jobs**、**notify** 及 **set** C shell 內建指令。

相關概念

C shell
C shell 是一種互動式指令直譯器，也是一種指令程式設計語言。它使用的語法和 C 語言相似。

C shell 內建指令
內建指令是在 shell 中執行的。如果內建指令是管線中的任何元件，除了最後一個指令以外，其他指令將在 subshell 中執行。

***C shell* 中的信號處理**
C shell 通常會略過無聲信號。離線執行的工作中不會受到鍵盤所產生之信號的影響（**INTERRUPT** 、**QUIT** 及 **HANGUP**）。

其他信號的值，則是 shell 從上代繼承而來的值。您可以在 shell 程序中，以 **onintr** 來控制 shell 對 **INTERRUPT** 及 **TERMINATE** 信號的處理方式。登入 shell 會根據 **TERMINATE** 信號的設置方式，決定要抓住它，或忽略它。 登入 shell 以外的其他 shell 會將 **TERMINATE** 信號傳送給子程序。當登入 shell 讀取 `.logout` 檔案時，絕不會允許出現 **INTERRUPT** 信號。

C shell 指令

簡式指令即是由空格或標籤所隔開的一系列字組。字組為一系列字元或數值，或兩者都有，但不包含沒有引號的空格。

此外，下列字元與雙字元在當作指令分隔字元或終止器時，也將形成單一字組。

```
&       |       ;
&&      ||      <<          > >
<       >       (           )
```

這些特殊字元可以是其他字組的一部分。然而若在它們前面加上反斜線 (\)，則可防止 shell 將它們解譯為特殊字元。以 ' ' 或 " "（一組對稱的引號字元）或左引號括住的字串，也可形成字組的一部分。當空格、跳格字元與特殊字元括住在這些標記中時，將無法形成個別字組。此外，您可以在換行字元之前加上一個反斜線 (\)，將換行字元含括在這些標記中。

簡式指令序列中的第一個字組（編號 0），通常是指定指令 Name。任何剩餘 字組，除了少數例外，都將傳送至該指令。如果指令指定一可執行檔，而該檔案為 一經過編譯的程式，則 shell 將立即執行該程式。如果該檔案標記為可執行檔 ，但未經編譯，shell 將假設它是一 shell Script。在此情形下，shell 將開始本身的另一案例 (subshell)，以讀取檔案，並執行內含的指令。

C shell 內建指令

內建指令是在 shell 中執行的。如果內建指令是管線中的任何元件，除了最後一個指令以外，其他指令將在 subshell 中執行。

註 : 如果您是自 C shell 提示輸入指令，則系統會先搜尋內建指令。如果內建指令並不存在，系統將搜尋由 *path* shell 變數所指定的目錄，尋找系統層次的指令。某些 C shell 內建指令與作業系統指令名 稱相同。然而，這些指令並不一定具有相同功能。欲進一步瞭解指令運作的方式，請查閱適當的指令說明。

如果您自 shell 執行 shell Script，且 shell Script 的第一行開頭為 `#!/shellPathname`，則 C shell 將執行註解中所指定的 shell，以處理 Script。否則，將執行預設的 shell（鏈結至 `/usr/bin/sh` 的 shell）。如果由預設的 shell 執行，則 C shell 內建 指令可能無法辨識。若要執行 C shell 指令，請將 Script 的第一行設定成 `#!/usr/bin/csh`。

相關參考

C shell 內建指令清單

以下是 C shell 的內建指令。

C shell 指令說明

C shell 提供下列內建指令。

項目	說明
alias [*Name* [*WordList*]]	若未指定任何參數，則顯示所有 alias。否則，指令將顯示 alias 為指定的 *Name*。如果指定 *WordList*，則指令將指派 *WordList* 的數值至 alias *Name*。指定的別名 *Name* 不能是 `alias` 或 `unalias`。
bg [%*Job* ...]	將現行工作或由工作 所指定的工作置於背景執行，並繼續執行已停止的工作。
break	在最接近含括的 **foreach** 或 **while** 指令的 end 之後，回復執行。
breaksw	岔斷自 **switch** 指令；在 **endsw** 指令後回復。
case *Label*:	定義 **switch** 指令中的 *Label*。
cd[*Name*]	相當於 **chdir** 指令（請參閱下列說明）。

項目	說明
chdir [*Name*]	變更現行目錄為由 *Name* 變數指定的目錄。如果您不指定 *Name*，指令將變更為您的起始目錄。如果 *Name* 變數值不是現行目錄的子目錄，且不是以 /、./ 或 ../ 開始，shell 會檢查 ***cdpath*** 變數的每一個元件，看看其是否有子目錄符合 *Name* 變數。如果 *Name* 變數是 shell 變數，且其值是以斜線 (/) 當作開頭，則 shell 會試用此變數，以查看它是否為目錄。**chdir** 指令相當於 **cd** 指令。
continue	在最接近含括的 **while** 或 **foreach** 指令的 end 之後，繼續執行。
default:	將 switch 陳述式標示為 default。**default** 應該出現在所有其他 case 標籤之後。
dirs	顯示目錄堆疊。
echo	將字串寫入 shell 的標準輸出。
else	執行 if（*Expression*）then ...else if（*Expression2*）then ... else ... endif 指令順序中， 第二個 else 後面的指令。 **註 :** 使用 **if(***expr***) then** ..**else** ...**endif** 時，**else** 陳述式為 **csh** 內建指令。如果 **(***expr***)** 為真，則會執行指令直到 **else** 陳述式。如果 **(***expr***)** 為假，則會執行 **else** 與 **endif** 陳述式之間的指令。以單引號括住的任何字元都採取文字方式，並且不會解譯。
end	連續將 *Name* 變數設定為由 *List* 變數所指定的每一成員，然後執行 **foreach** 與符合的 **end** 陳述式之間的一連串 *Commands* 指令。**foreach** 與 **end** 陳述式必須在分隔行上單獨出現。 使用 continue 陳述式，繼續執行迴圈；與 break 陳述式，提前結束迴圈。自終端機讀取 **foreach** 指令時，C shell 將以 ?（問號）提示，以輸入指令。迴圈中的指令將以 ? 提示，且不會列入歷程清單之中。
endif	如果 *expression* 變數為真，將會執行第一個 then 陳述式後面的 *Commands*。如果 else if *Expression2* 為真， 將會執行第二個 then 陳述式後面的 *Commands*。如果 else if *Expression2* 為假， 將會執行 then 後面的 *Commands*。可能會有任何數目的 else if 配對。 但僅需要一個 endif 陳述式。else 區段為選用性。 字組 else 及 endif 只可以使用在輸入行開頭。 if 區 段必須單獨出現在其輸入行，或在 else 指令之後。
endsw	持續將每一 **case** 標籤和 *string* 變數值比對。字串為指令與第一個展開的檔名。請在 **case** 標籤中使用型樣相符字元 *、? 及 [. . .]，它們是由變數展開來的。如果在發現**預設**標籤前找不到相符的標籤，則依**預設**標籤執行。**case** 標籤與 **default** 標籤必須出現在指令行開頭。**breaksw** 指令將在 **endsw** 指令之後執行。否則，控制元將可能落入 **case** 及 **default** 標籤，如同 C 程式設計語言。如果沒有標籤符合且沒有**預設值**，將會在 **endsw** 指令之後繼續執行。
eval *Parameter . . .*	讀取輸入至 shell 時的 *parameter* 變數值，並執行結果指令，或現行 shell 中的指令。使用此指令來執行指令或變數替代所產生的指令，因為會在進行這些替代作業之前先進行剖析。
exec *Command*	執行現行 shell 中指定的指令。
exit (*Expression***)**	以 ***status*** shell 變數值（如果未指定 *Expression*），或以指定的 *Expression* 數值結束 shell。
fg [%*Job* ...]	將現行工作或由工作所指定的工作置於前端執行，並繼續執行已停止的工作。

項目	說明
foreach *Name* (*List*) *Command*. . .	連續設定一個 *Name* 變數給 *List* 變數所指定的每個成員，並設定指令順序，直到遇到 **end** 指令為止。
glob *List*	使用歷程、變數與檔名擴充顯示 *list*。將空字元置於字組之間，且在結束時不包括換行字元。
goto *Word*	繼續執行由 *word* 變數所指定的指令行。指定的 *word* 為檔名與展開以產生由 *Label:* 變數所指定的字串格式的指令。 shell 將盡可能重新整理輸入資料，並搜尋格式為 *label***:** 的指令行，該行可能在空格或標籤之後。
hashstat	顯示統計資料，指出 hash 表如何順利完成尋找指令。
history [**-r** | **-h**] [*n*]	顯示歷程事件清單。最舊的事件最先顯示。如果您指定一數字 *n*，則只會顯示指定數目的最新事件。 **-r** 旗標將反轉顯示事件的順序，因此變成先顯示最新的事件。**-h** 旗標會顯示沒有編號的歷程清單。使用此旗標可產生適合與 **source** 指令的 **-h** 旗標一起使用的檔案。
if (*Expression*) *Command*	如果指定的表示式 為真，則執行指定的指令（包括其引數）。 *Command* 變數上的變數替代較早發生，與 `if` 陳述式的其他指令同時發生。指定的指令必須為一簡式指令（非管線，或括弧內的指令清單）。 **註 :** 即使 *Expression* 變數為假，且未執行 *Command* 指令，也會發生輸入和輸出重新導向。
jobs [**-l**]	列出作用中的工作。使用 **-l**（小寫 *L*）旗標時， **jobs** 指令會列出處理程序 ID 與工作號碼及名稱。
kill -l | [[-*Signal*] % *Job*...| *PID*...]	將 **TERM**（終止）信號或由 *signal* 所指定的信號傳送至指定的 *Job* 或 *PID*（處理程序）。藉由編號或名稱指定信號（如在 `/usr/include/sys/signal.h` 檔案中所提供的，並去掉 **SIG** 字首）。**-l** (lowercase *L*) 旗標會列出信號名稱。

項目	說明
limit [**-h**] [*Resource* [*Max-Use*]]	由現行處理程序與其所建立的處理程序，限制指定資源的用法。處理資源限制將定義在 /etc/security/limits 檔案中。可控制的資源為中央處理單位 (CPU) 時間、檔案大小、資料大小、核心傾出大小與記憶體使用。當使用者新增至系統中時，這些資源的最大可接受值將設定於 **mkuser** 指令中。它們將以 **chuser** 指令變更。 限制分為軟性和硬性。使用者可能增加它們的軟性限制，至由強迫限制實施的上限。您必須有 root 使用者權限，才可增加軟性限制超越強迫限制，或變更強迫限制。**-h** 旗標顯示強迫限制，而不顯示軟性限制。 如果未指定 *Max-Use* 參數，**limit** 指令會顯示指定資源的現行限制。如果未指定 *Resource* 參數，**limit** 指令會顯示所有資源的現行限制。如需由 **limit** 次指令所控制之資源的相關資訊，請參閱 *Technical Reference: Base Operating System and Extensions, Volume 1* 中的 getrlimit、setrlimit 或 vlimit 次指令。 CPU 時間的 *Max-Use* 參數是以 *hh*:*mm*:ss 格式指定。其他資源的 *Max-Use* 參數則指定為浮點數字或整數，其後有時會跟著比例係數。比例係數是 k 或 KB（1024 個位元組）、m 或 MB，或者是 b 或區塊（ulimit 子常式所使用的單位，說明於 *Technical Reference: Base Operating System and Extensions, Volume 2*）。如果您不指定比例係數，所有資源將使用 k 作為比例係數。就資源 Name 與比例係數二者而言，明確的 Name 字首即已足夠。 **註：**只有在其他作用中的處理程序爭相使用系統記憶體時，此指令才會限制可用在處理程序的實體記憶體。
login	結束登入 shell，並置換為 /usr/bin/login 指令的案例。 這是登出的方法之一（包含 **ksh** 和 **bsh** 指令的相容性）。
logout	結束登入 shell。如果設定 ignoreeof 選項，則需使用此指令。
nice [**+***n*] [*Command*]	如果未指定數值，則將設定在此 shell 中執行的指令優先順序為 24。如果指定 ***+n*** 旗標，則將設定優先順序加上指定的數目。如果指定 ***+n*** 旗標與 *Command*，則將以優先順序 24 加上指定數字來執行 *Command*。如果您有 root 使用者權限，您可以負數執行 **nice** 陳述式。*Command* 一定是在 subshell 中執行，且適用對簡式 **if** 陳述式中的指令所設的限制。
nohup [*Command*]	如果未指定 *Command*，將導致 Script 的其餘部分忽略 **hangups**。如果有指定 *Command*，則會導致指定的 *Command* 執行時，忽略 **hangups**。如需執行管線或指令清單，請將管線或清單放置在 shell Script 中，提供 Script 執行許可權，並使用 shell Script 作為指令變數值。在背景中以 & 符號 (&) 來執行的所有處理程序都會受到有效的保護，在您登出時， 不會傳送 **hangup** 信號。不過，這些處理程序仍將明確地傳送 **hangups**，除非使用 **nohup** 陳述式。
notify [**%***Job*...]	將導致 shell 通知您何時變更現行工作狀態或指定的工作。一般而言，shell 將在顯示 shell 提示前提供通知。如果設定 ***notify*** shell 變數，此特性將成為自動。
onintr [**-** | *Label*]	控制 shell 岔斷動作。若未指定引數，將還原 shell 的岔斷預設動作，此舉將結束 shell Script，或回到指令輸入層次。如果指定 **-** 旗標，將導致所有岔斷被忽略。如果有指定 *Label*，當 shell 收到岔斷或當子處理程序由於岔斷而結束時， 將導致 shell 執行 goto *Label* 陳述式。不論如何，如果 shell 正在分開執行，且岔斷被忽略，則所有格式的 **onintr** 陳述式將無意義。shell 與所有被呼叫的指令將繼續忽略岔斷。

項目	說明
popd [**+***n*]	存取目錄堆疊，並變更至新的頂端目錄。如果有指定 ***+n*** 變數，指令將會捨棄堆疊中的第 *n* 個項目。目錄堆疊的元素自頂端從 0 開始編號。
pushd [**+***n*|*Name*]	如果沒有引數，將交換目錄堆疊最頂端的 2 個元素。利用 *Name* 變數，指令將會變更為新的目錄，並將舊的現行目錄（如在 ***cwd*** shell 變數中所提供的）推向目錄堆疊。如果有指定 ***+n*** 變數，指令會輪換目錄堆疊中的第 *n* 個元件直到項端元素，然後變成它。目錄堆疊的成員自頂端，從 0 開始編號。
rehash	將導致 ***path*** shell 變數中的目錄內容的內部 hash 表重新計算。 如果新的指令在您登入時，新增至 ***path*** shell 變數中的目錄，則需要此動作。只要新增至其中一個使用者自己的目錄，或如果有人變更其中一個系統目錄，則需要執行 **rehash** 指令。
repeat *Count Command*	遵循在指令簡式 **if** 陳述式中相同的限制，與由 *Count* 所指定的次數，執行指定的 *Command*。 **註：** I/O 重新導向只會發生一次，即使 *Count* 變數等於 0 也是如此。
set [[*Name*[*n*]] [**=** *Word*]] | [*Name* = (*List*)]	當不使用引數時，顯示所有 shell 變數的值。具有一個以上單一字組的變數，將顯示為一使用括弧的字組清單。如果只指定 *Name* ，C shell 將設定 *Name* 變數為空字串。否則將設定 *Name* 為 *word* 變數值，或設定 *Name* 變數為由 *List* 變數所指定的字詞清單。有指定 *n* 時， *Name* 變數的第 *n* 個元件會設定為 *Word* 變數的值；第 *n* 個元件必須已存在。不論如何，數值為指令與檔名的擴充。這些引數可能在單一 **set** 指令中，重複設定多重數值。然而，變數擴充在任何設定前，將發生在所有引數身 上。
setenv*Name Value*	將由 *Name* 變數所指定的環境變數值設定為 *value*，一單一字串。最常使用的環境變數 ***USER***、***TERM***、***HOME*** 及 ***PATH*** 會自動匯入 C shell 變數 ***user***、***term***、***home*** 及 ***path***，並且會自動從這些 C shell 變數中匯出。這些變數使用 **setenv** 陳述式。
shift [*Variable*]	將 ***argv*** shell 變數的成員，或指定的 *Variable* 移位至左邊。如果未設定 **argv** shell 變數或指定的 *Variable*，或其值少於一個字詞，將會發生錯誤。
source[**-h**] *Name*	讀取在 *Name* 檔案中撰寫的指令。您可將 **source** 指令作巢狀處理。然而，如果巢狀太深，shell 將可能會用盡檔案描述子。任何層次所發生的 **source** 指令錯誤，將結束所有巢狀 **source** 指令。一般而言，在 **source** 指令中的輸入， 將不會列入歷程。**-h** 旗標將導致指令置於歷程清單中，而不予執行。
stop [**%***Job* ...]	停止現行工作，或在背景 執行的指定工作。
suspend	停止 shell，如同已接收 **STOP** 信號。
switch (*string*)	開始 **switch**（*String*）**case** *String*：... **breaksw** **default:** ... **breaksw** **endsw** 指令順序。此指令序列根據 *String* 變數的值來連續與每一個 case 標籤相比對。如果在發現預設標籤前無符合的標籤，則依預設值標籤執行。

項目	說明

time [*Command*]	**time** 指令控制指令自動計時。如果未指定 *Command* 變數，則 **time** 指令會顯示此 shell 與其子項使用的時間摘要。如果使用 *Command* 變數指定指令，將會進行計時。然後 shell 將顯示時間總結，如 ***time*** shell 變數說明。如有必要將建立額外 shell，在指令完成時，將顯示時間統計值。
	
	下列範例使用 **time** 及 **sleep** 指令：
	
	`time sleep`
	
	此指令的輸出值看起來像下面這個樣子：
	
	`0.0u 0.0s 0:00 100% 44+4k 0+0io 0pf+0w`
	
	輸出欄位如下：
	
	第一
	投注在使用者處理程序的 CPU 時間秒數。
	第二
	由核心代表使用者處理程序使用的 CPU 時間秒數。
	第三
	指令經歷過的（壁鐘）時間
	第四
	總計使用者 CPU 時間加上系統時間，作為使用時間的百分比。
	第五
	已使用的共用記憶體平均數量，加上已使用的非共用資料空間，以千位元組計算
	第六
	輸入和輸出作業區塊數目
	第七
	尋頁錯失加上交換數目

umask [*Value*]	決定檔案的許可權。此 *value* 與建立處理程序許可權，在檔案建立時，決定檔案的許可權。預設值為 022。若未指定 *value* ，則會顯示現行設定。
unalias *|*Pattern	捨棄所有符合 *Pattern* 變數的 alias， 所有 alias 將由 **unalias *** 指令移除。沒有 alias 並不會導致錯誤。
unhash	停用內部 hash 表，以尋找執行中的程式。
unlimit [**-h**][*Resource*]	移除 *Resource* 變數的限制。如果沒有指定 *Resource* 變數，會移除所有的資源限制。請參閱 **limit** 指令的說明，以取得 *Resource* 名稱的清單。 **-h** 旗標將移除相對應的強迫限制。只有具有 root 使用者權限的使用者才可變更強迫限制。
unset *|*Pattern	移除全部符合型樣變數的 Name 變數。使用 **unset ***，移除全部變數。若未設定變數並不會導致錯誤發生。
unsetenv*Pattern*	從其名稱符合指定 *Pattern* 的環境中移除所有的變數 (請參閱 **setenv** 內建指令）。
wait	等待所有背景 工作。如果 shell 為互動式，**INTERRUPT**（通常是 Ctrl-C 鍵順序）會中斷等待。然後 shell 將顯示未執行的所有工作的 Name 與工作號碼。

項目	說明
while (*Expression*) *Command*. . . end	當由 *Expression* 變數指定的表示式評估為非零值時，會評估 **while** 與配對的 **end** 陳述式之間的 *Commands*。您可使用 **break** 陳述式提前結束，並使用 **continue** 陳述式繼續迴圈。**while** 與 **end** 陳述式必須單獨出現在它們的輸入行中。如果輸入是來自終端機，則會在類似 **foreach** 陳述式的 **while** (*Expression*) 之後出現提示。
@ [*Name*[*n*] = *Expression*]	當不使用引數時，顯示所有 shell 變數的值。否則將設定由 *Name* 變數指定的 Name 為 *Expression* 變數的值。如果表示式包含 <、>、& 或 | 字元，這部分的表示式必須放置在括弧內。有指定 *n* 時， *Name* 變數的第 *n* 個元件會設定為 *Expression* 變數。*Name* 變數及其第 *n* 個元件都必須存在。 可使用 C 語言運算子（例如 *= 及 +=）。隔開 *Name* 變數與指派運算子的空格為選用性的。無論如何，然而，隔開表示式變數元件的空格是必要的，否則該元件將會視為一單一字組。特殊字尾運算子、雙加號 (++) 及雙連字號 (- -) 將分別增加及減少 *Name* 變數的值。

C shell 表示式及運算子

@ 內建指令與 **exit**、**if** 及 **while** 等陳述式，都可以接受包括和 C 語言類似的運算子表示式，且其優先順序相同。

下列運算子可供使用：

運算子	意義為
()	變更優先順序
~	補數
!	否定
*/ %	乘，除，模數
+ -	加，減
<< > >	左移，右移
<= >= < >	關係運算子
== != =~ !~	字串比較／型樣相符
&	執行位元 AND 運算
^	執行位元互斥 OR 運算
|	執行位元併入 OR 運算
&&	邏輯 AND
||	邏輯 OR

在上述清單中，運算子的優先順序是由左至右、由上至下降低。

註： 運算子 + 與 - 為右向組合。例如，執行運算式 a + b - c，如下所示：

```
a + (b - c)
```

不是像下面這樣：

```
(a + b) - c
```

==、!=、=~ 及 !~ 等運算子都會比較它們的引數與字串；其他的都當作數字來運算。=~ 與 !~ 運算子類似 == 與 !=，除了最右側的是 *pattern*，最左側的為其配對的運算元。這樣會減少在 shell 程序中使用 **switch** 陳述式的需要。

也可以使用邏輯運算子 **or** (||) 及 **and** (&&)。您可以用它們來檢查某一範圍的號碼，範例如下：

```
if ($#argv > 2 && $#argv < 7) then
```

前述範例中，引數數字必須大於 2，且小於 7。

開頭為零 (0) 的字串被視為是八進位數字。空值或遺漏的引數則視為 0，所有表示式的結果都是十進位數的字串。請注意表示式的兩個元件可以出現在相同的字組內。除非鄰近的表示式元件的語法對剖析器 (& | < > ( )) 具有意義，否則表示式元件前後應該以空格隔開。

也可以在原始的運算元為含括在括弧 ( ) 內的指令執行，且檔案的查詢格式為 (**-operator** *Filename*) 的表示式中使用，其中的 **operator** 為下列其中一項：

項目	說明
r	讀取權
w	寫入權
x	執行權
e	存在
o	所有權
z	零大小
f	一般檔案
d	目錄

指定的 *Filename* 是擴充的指令及檔名，然後會測試它，以察看它與實際使用者之間是否有指定的關係。如果 *Filename* 不存在或無法存取，所有的查詢都會傳回 `false(0)`。如果指令順利執行，則查詢將傳回 `true(1)` 值。否則如果指令失效，則查詢將傳回 `false(0)` 值。如果需要更多詳細狀態資訊，請在表示式外執行該指令，然後檢查 *status* shell 變數。

C shell 中的指令替代
在指令替代中，shell 將執行指定的指令，然後以其輸出置換該指令。

若要在 C shell 中執行指令替代，請將指令或指令字串含括在左引號 (` `) 中。一般而言，shell 會在空格、跳格字元與換行字元的地方，將指令的輸出值分開成個別的字組。然後以其輸出置換原始指令。

在下列範例中，含括 **date** 指令的左引號 (` `) 指出將會替代指令的輸出：

```
echo The current date and time is: `date`
```

此指令的輸出看起來類似下列內容：

```
The current date and time is: Wed Apr 8 13:52:14 CDT 1992
```

C shell 有條件的執行內建 shell 指令的引數指令替代。此舉表示並未擴充未經求值的表示式的那些部分。對不是內建的指令而言，shell 將個別替換引數清單中的指令名稱。替代會發生在主要 shell 的子項中， 就在 shell 執行輸入或輸出重新導向之後。

如果指令字串由 " " 括住，則 shell 只將換行字元視為字組分隔字元，如此即可將空格與跳格字元 (Tab) 保留在字組內。在所有個案中，單一最終換行字元並不一定會強制形成新的字組。

相關概念
C shell 中的變數替代
C shell 維持一組變數，每一個變數都有自己的值，這個值可能是一列 0 或多個字組。由 shell 設定或參照一部分這些變數。例如，*argv* 變數是 shell 變數清單的影像，而構成此變數值的字組會以特殊方式來參照。

非內建的 *C shell* 指令執行
當 C shell 判斷指令並非內建 shell 指令時，會嘗試以 execv 子常式來執行指令。

path shell 變數中的每一字組，將指定在其中執行指令的目錄。如果沒有**-c** 或 **-t** 旗標，shell 將這些目錄中的 Name 置於內部表格內。只有在指令可能位於目錄時，shell 才會嘗試呼叫目錄上的 execv 子常式。如果您以 **unhash** 指令關閉此機制，或給與 shell **-c** 或 **-t** 旗標，則 shell 將連結此給定的指令名稱，來形成檔案路徑名稱。若有任何目錄元件的 *path* 變數不是以斜線 (/) 開頭，shell 也會執行此作業。然後 shell 將嘗試執行該指令。

括弧中的指令一定是在 subshell 中執行。例如：

```
(cd ; pwd) ; pwd
```

顯示起始目錄，而不變更現行目錄位置。然而，指令：

```
cd ; pwd
```

將變更現行目錄位置為起始目錄。括弧中的指令大部分通常是用以防止 **chdir** 指令影響現行 shell。

如果檔案具有執行許可，但非系統可執行的二元運算子，shell 將假設其為包含 shell 執行新 shell 讀取的檔案。

如果有 shell alias，則 alias 的字組字首將使用引數 list，以形成 shell 指令。alias 的第一個字組應該是 shell 的完整路徑名稱。

C shell 中的歷程替代
歷程替代可讓您修改上一個指令的個別字組以建立新的指令。歷程替代讓它輕易重複指令、重複在目前指令中上一個指令的引數，或是修訂在上一個指令中鍵入的拼字錯誤。

歷程替代以驚嘆號 (!) 字元為開頭，如果它們不是巢狀的（也就是說，歷程替代無法包含另一個歷程替代），則可以出現在指令行的任何地方。您可以在 ! 之前加上 \，以取消驚嘆號的特殊意義。此外，如果您在空格、跳格字元、 換行字元、= 或 (之前加上 !，就不會發生歷程替代。

歷程替代也會發生在您以 ^ 為開頭輸入行時。這個 shell 在執行該行之前會回應任何在工作站包含歷程替代的輸入行。

相關概念
C shell 中的別名替代
別名即指派給指令或指令字串的名稱。C shell 可讓您指派別名並隨意下指令使用它們。shell 會維護您所定義的別名清單。

C shell 中的歷程清單
歷程清單會儲存 shell 從由一個或是一個以上字詞所組成的指令行中讀取的指令。歷程替代會從這些儲存至輸入串流的指令中再介紹字組的序列。

history shell 變數控制歷程清單的大小。您必須在 .cshrc 檔案中，或利用內建 **set** 指令在指令行中設定 *history* shell 變數。上一個指令不管是不是 *history* 變數的值一定會保留。歷程清單中的指令從 1 開始循序編號。內建的 **history** 指令會產生類似下列的輸出：

```
9 write michael
10 ed write.c
11 cat oldwrite.c
12 diff *write.c
```

此 shell 會顯示指令字串，以及它們的事件編號。事件號碼會出現在指令的左邊，並表示指令輸入的時間與歷程中其他指令之間的關係。通常並不需要使用事件編號來參照事件，但是您可以在指派給 *PROMPT* 環境變數的提示字串中放置一個驚嘆號 (!)，將目前的事件編號顯示成系統提示的一部分。

一個完整的歷程參照包含某個特定事件、字詞指派元以及在下列一般格式中一個或一個以上的修飾元：

```
Event[.]Word:Modifier[:Modifier] . . .
```

註 : 只能修改一個字組。系統不接受包含空白的字串。

在上一個 **history** 指令輸出中的範例中，現行事件編號是 13。使用本例，下列項目請參閱上一個事件：

項目	說明
!10	事件編號 10。
!-2	事件編號 11（現行事件減號 2）。
!d	以 d 開頭的指令字組（事件編號 12）。
!?mic?	包含字串 mic 的指令字組（事件編號 9）。

這些格式，沒有做進一步的修改，簡單地再介紹指定的事件字組，每一個事件字組都由一個空白所隔開。在特殊情況下 !! 代表前一指令；單獨出現在輸入行的 **!!** 指令，表示重新執行前一指令。

C shell 的事件規格
若要選取某個事件中的字詞，請在事件規格之後加上一個冒號 (:)， 其後接著下列其中一個字組指派元（輸入行的字組是從 0 開始循序編號）

項目	說明
0	第一個字組（指令名稱）
n	第 *n* 個引數
^	第一個引數
$	最後一個引數
%	以前置 ?*string*? 搜尋相符的字組
x-y	從第 *x* 個字到第 *y* 個字的字組範圍。
-y	從第一個字 (0) 到第 *y* 個字的字組範圍
*****	第一個到最後一個引數，或是如果事件中只有一個字組（指令名稱），則沒有引數
*x******	第 *x* 個引數到最後一個引數
*x***-**	與 *x** 相同，但省略了最後一個引數

如果字組指定元是以 ^、$、*、- 或 % 開頭，您可以省略用來將事件規格及字組指定元隔開的冒號。您也可以在選用字組指派元之後，放置下列修改的序列，每一個序列之前都加上冒號：

項目	說明
h	移除一個跟隨在路徑名稱之後的副檔名，離開標頭。
r	移除結尾的 *.xxx* 元件，留下根名稱。
e	移除全部，但 *.xxx* 尾端擴充除外。
s/ *OldWord***/***NewWord* **/**	以 *NewWord* 變數的值來替代 *OldWord* 變數的值。

某個替代的左邊不是一個由編輯器可辨識之字串的型樣；然而，它是一個字組、一個沒有空白的單一單元。一般來說， 斜線 (/) 會區隔原始字詞 (*OldWord*) 以及它的替代字組 (*NewWord*)。然而，您可使用任何字元為定界字元。在下列範例中，使用 % 作為定界字元，可讓 / 併入字詞中：

```
s%/home/myfile%/home/yourfile%
```

此 shell 會以 *NewWord* 變數中的 *OldWord* 文字來置換 & 符號 (&)。在下列範例中，/home/myfile 會變成 /temp/home/myfile。

```
s%/home/myfile%/temp&%
```

shell 會以最後的替代，或是在上下文掃描 `!?String?` 中使用的最後字串，來置換空值字詞。如果後面要立刻接換行字元的話，您可以省略尾端的定界字元 (/)。請使用下列修飾元來區隔歷程清單：

項目	說明
t	移除所有前導路徑名稱元件，只留最後的部分
&	重複之前的替代作業
g	全面引用變更；也就是所出現的每一行。
p	顯示新的指令，但不執行
q	將替換字組放在引號中，以防止進一步的替代
x	動作類似 **q** 修飾元，但它是在空格、跳格字元以及換行字元的地方中斷字組

使用前置的修飾元時，變更只會套用在第一個可更改的字組上，除非 **g** 修飾元位於所選取的修飾元之前。

如果您提供的歷程參照沒有事件規格（例如，`!$`），則此 shell 會使用上一個指令作為事件。如果上一個歷程參照發生在相同一行，則此 shell 會重複上一個參照。如此，下列序列會提供符合 `?foo?`指令的第一以及最後一個引數。

```
!?foo?^ !$
```

特殊的歷程參照縮寫會發生在第一個輸入行是 ^ 非空白字元時。這相當於 `!:s^`，這樣可提供在上一行文字中便於速記的替代。指令 `^lb^` lib 會更正指令中的 `lib` 拼字。

如果需要的話，您可以將歷程替代含括在 `{ }` 中， 讓它和隨後的字元隔離。例如，如果您想要使用某個參照的指令：

```
ls -ld ~paul
```

執行此指令：

```
ls -ld ~paula
```

使用下列建構：

```
!{l}a
```

在此範例中，`!{l}a` 會尋找以 `l` 開頭，並在尾端附加 a 的指令。

以單引號及雙引號來含括
若要防止進一步解譯所有或部分的替代，請用單引號或雙引號來含括字串。

將字串含括在 ' ' 中可以避免進一步的解譯，不過，如果字串是含括在 " " 中，則允許進一步的擴充。在這兩種清況下，最後的文字都會變成單一字組的全部或部分。

C shell 中的輸入及輸出重新導向
在 C shell 執行某個指令之前，它會掃描指令行中的重新導向字元。這些特殊的表示法會將 shell 導入重新導向輸入及輸出。

您可以用下列的語法陳述指令來重新導向標準輸入及輸出：

項目	說明
< *File*	開啟指定的 *File*（它是所展開的第一個變數、指令及檔名）來作為標準輸入。

項目	說明
<<*Word*	讀取 shell 輸入，直到符合 *Word* 變數值的那一行為止。*Word* 變數並不受限於變數、檔案名稱或是指令替代。每一個輸入行會在任何行中完成替代之前與 *Word* 變數相比較。除非在 *Word* 變數中出現引用字元 （\、"、' 或 `），否則 shell 會在其間的行上執行變數及指令替代，讓 \ 字元能引用 $、\ 及 ` 字元。被替換的指令擁有所有保留的空格、跳格字元及換行字元，但最後被捨棄的換行字元除外。所產生的文字會置於提供給指令作為標準輸入的匿名暫用檔中。
> *File* >!*File* >& *File* >&! *File*	使用指定的 *File* 來作為標準輸出。如果 *File* 不存在，則它會被建立。如果 *File* 存在，它會被截斷，而且會遺失之前的內容。如果有設定 *noclobber* shell 變數，*File* 就一定不能存在，或是為特殊字元的檔案，否則將產生錯誤。這樣有助於防止意外破壞檔案。在此例中，是使用包含！的格式 以抑制此檢查。*File* 會以和 < 輸入檔案名稱相同的方式來展開。>& 格式會將標準輸出及標準錯誤重新導向至指定的 *File*。下列範例會顯示如何個別將標準輸出重新導向至 /dev/tty，並將標準錯誤重新導向至 /dev/null。此括弧允許隔開標準輸出及標準錯誤。 `% (find / -name vi -print > /dev/tty) >& /dev/null`
> >*File* > >!*File* > >& *File* > >&! *File*	使用指定的 *File* 作為標準輸出（如 >），但將輸出附加至 *File* 的尾端。如果有設定 *noclobber* shell 變數，當 *File* 不存在時，將會產生錯誤，除非提供其中一個包括！的格式。否則，它會類似 >。

某個指令會接收呼叫 shell 中的環境，作為由輸入/ 輸出參數所做的變更，且指令是以 pipline 的型態存在。因此，不像先前某些 shell 一樣，從 shell Script 中執行的指令不會由預設值來存取指令的文字。它們會改為接收原始標準輸入的 shell。請使用 << 機制來表示行內資料， 這樣可以讓 shell 指令檔當成管線的元件來使用，並可讓 shell 區塊讀取其輸入。請注意，分離執行的指令預設標準輸入是不會變更為空的 /dev/null 檔。標準輸入會改會保留原始標準輸入的 shell。

若要透過管線和標準輸出來重新導向標準錯誤，請使用 |& 格式，而不只使用 |。

C shell 中的流程控制
shell 含有用來調節指令檔中的控制串流 (S hell Script) 以及（有限的但是非常有用的）從 shell 指令行輸入的指令。這些指令都由強制 shell 重複，或是在它的輸入中略過來作業。

foreach、**switch** 和 **while** 陳述式，以及 **if** 陳述式的 **if-then-else** 格式，都需要主要關鍵字是以單一簡式指令出現在輸入行上。

如果 shell 輸入不可搜尋，則每逢讀取一個循環時， shell 就會將輸入置入緩衝區，並且會搜尋內部緩衝區來執行循環指示的重新讀取動作。此動作的允許範圍是往回的 **goto** 必須順利到達您無法搜尋的輸入。

作業系統安全

電腦安全的目標是要保護儲存在電腦系統中的資訊。

資訊安全保護的目標如下：

項目	說明
完整性	所有資訊的價值都在於它的精確度。如果對資料進行未獲授權的變更，這些資料就會損失部分或所有的價值。
私密性	許多資訊的價值在於它的機密性。
可用性	資訊必須是隨時可用的。

在您開始使用系統之前，規劃及實施您的安全策略是非常有用的。日後若要改變安全策略是非常耗時的，所以現在的規劃可節省日後很多時間。

識別與鑑別
識別與鑑別用來建立您的身分。

您需要登入系統。如果帳戶有密碼，請提供您的使用者名稱和密碼（在安全保護系統中，所有帳戶都應有密碼或使其無效）。如果密碼正確，您就可以登入該帳戶；您可以獲得該帳戶的存取權與專用權。

由於密碼是對您帳戶的唯一保護，所以應小心地選擇及保護您的密碼。許多侵入系統的嘗試都是從猜測密碼開始。作業系統藉由將使用者密碼與其他使用者資訊分開保存，提供有意義的密碼保護。使用者的加密密碼及其他安全性相關資料都儲存在 /etc/security/passwd 檔案中。僅 root 使用者可存取此檔案。透過對加密密碼的限制存取，侵入者無法經由所有可能與類似的密碼簡單循環來對密碼解密。

它仍有可能猜測密碼來重複登入帳戶的嘗試。如果密碼很簡單，或是沒有經常變更，則很容易發生這類的侵入事件。

登入使用者 ID
作業系統可藉由使用者的登入使用者 *ID* 來識別使用者。

登入使用者 ID 容許系統追蹤所有使用者動作的來源。在使用者登入系統之後，與執行初始使用者程式之前，系統會將處理程序的登入 ID 設成在使用者資料庫中找到的使用者 ID。登入階段作業期間所有後續的處理程序都以這個 ID 標示。這些標籤提供登入使用者 ID 執行的所有活動追蹤。

在階段作業期間，您可以重設有效使用者 *ID*、實際使用者 *ID*、有效群組 *ID*、實際群組 *ID* 及輔助群組 *ID*，但您無法變更登入使用者 ID。

無人式終端機
如果終端機已登入且無人操作，所有系統就很容易受到攻擊。當系統管理人員讓使用 root 授權啟用的終端機處於無人看管的狀態時，會發生最嚴重的問題。一般而言，使用者只要離開終端機，就應該要登出。

在 /etc/profile 檔案中設定 TMOUT 及 TIMEOUT 參數，可以強迫終端機在不作用一段時間後，即登出。此 TMOUT 參數適用於 **ksh** (Korn) shell 中，而 TIMEOUT 參數適用於 **bsh** (Bourne) shell。

下列範例來自 .profile 檔，強迫終端機在不活動一個小時後登出：

```
TO=3600
echo "Setting Autologout to $TO"
TIMEOUT=$TO
TMOUT=$TO
export TIMEOUT TMOUT
```

註：您可以在您的起始目錄的 .profile 檔案中指定不同的值，來置換 /etc/profile 檔案中的 TMOUT 及 TIMEOUT 值。

相關概念
Bourne shell 中的變數替代
Bourne shell 允許您執行變數替代。

相關參考
Korn shell 或 POSIX shell 中的參數替代
Korn shell 或 POSIX shell 可讓您執行參數替代。

檔案所有權及使用者群組
起初，檔案擁有者是藉由該檔建立者的使用者 ID 來識別。

檔案擁有者可決定誰可以讀取、寫入（修改）或執行此檔。您可以使用 **chown** 指令來變更所有權。

每一個使用者 ID 是以唯一的群組 ID 指派至群組。 系統管理者在設置系統時便建立了使用者群組。在建立新檔案時，作業系統會指派許可權給建立的使用者 ID、 包含檔案擁有者的群組 ID，以及一個叫做 others 的群組，其包含所有其他使用者。**id** 指令可顯示您的使用者 ID (UID)、群組 ID (GID) 以及您所屬的所有群組名稱。

在檔案清單中（例如：**ls** 指令所顯示的清單），使用者的群組一定都是以下列次序呈現：使用者、群組及其他。如果您需要找出您的群組名稱， **groups** 指令會顯示使用者 ID 的全部群組。

變更檔案或目錄所有權

使用 **chown** 指令來變更您檔案的擁有者。

當您指定 **-R** 選項時，**chown** 指令會從指定目錄的目錄結構遞迴地下降。在發現符號鏈結時，鏈結指向的檔案或目錄所有權會變更；但符號鏈結的所有權不變。

註： 只有 root 使用者才能變更另一個檔案的擁有者。若指定了 **-f** 選項，則系統不會顯示錯誤。

例如，若要變更 `program.c` 檔案的擁有者，請鍵入：

```
chown jim program.c
```

`program.c` 檔案使用者存取權現在適用於 `jim`。作為擁有者，`jim` 可以使用 **chmod** 指令來允許或拒絕其他使用者存取 `program.c` 檔案。

請參閱 **chown** 指令，以取得完整語法。

檔案和目錄存取模式

每一個檔案都有一個擁有者。對新檔案來說，建立此檔的使用者便是該檔的擁有者。擁有者會指派存取模式給此檔案。 存取模式會授與其他系統使用者許可權來讀取、修改或執行該檔。僅有檔案擁有者或具 root 權限的使用者可以變更此檔的存取模式。

使用者有三個類別：使用者/擁有者、 群組以及其他所有使用者。存取權是以下列三種模式的組合來授與這些使用者類別：讀取、寫入或執行。在建立新檔案時，建立該檔的使用者之預設許可權為讀取、寫入及執行。其他兩個群組則具有讀取以及執行許可權。下表說明這三類使用者群組的預設檔案存取模式：

項目	**說明**		
類別	**讀取**	**寫入**	**執行**
擁有者	是	是	是
群組	是	否	是
其他	是	否	是

系統決定誰具有許可權及其對每一個活動所具有的許可權層次。在作業系統中，存取模式會以符號及號碼二種方式呈現。

相關概念

檔案類型

系統可以辨識的檔案類型為**一般**、**目錄**或**特殊**。然而，作業系統使用這些基本類型的變化類型。

存取模式的符號表示法

存取模式可以使用符號表示。

項目	**說明**
r	表示讀取許可權，可讓使用者檢視檔案內容。
w	表示寫入許可權，可讓使用者修改檔案內容。
x	表示執行許可權。對可執行檔而言（包含程式的一般檔案），執行許可權意即可以執行程式。對目錄而言，執行許可權意即可以搜尋檔案內容。

檔案或目錄的存取模式是以九個字元來表示。第一組的三個字元代表現行的**擁有者**許可權， 第二組的三個字元代表現行的**群組**許可權， 第三組的三個字元則代表**其他**許可權的現行設定。在這個九個字元的組合中，連字號 (**-**) 表示沒有給予許可權。例如，存取模式設定為 `rwxr-xr-x` 的檔案提供讀取及執行許可權給所有的三個群組，但寫入許可權僅提供給檔案擁有者。 這是預設值的符號式表示法。

當 **ls** 指令與 **-l**（小寫 L）旗標一起使用時，將提供現行目錄之詳細清單。**ls -l** 清單的前 10 個字元表示這三個群組各自的檔案類型及許可權。**ls -l** 指令也會列出與每一個檔案及目錄相關聯的擁有者和群組。

第一個字元指出檔案類型。 剩餘的九個字元包含這三種使用者類別中每一種的檔案許可權資訊。下列符號是用來代表檔案類型：

項目	說明
-	一般檔案
d	目錄
b	區塊特殊的檔案
c	字元特殊的檔案
p	管線特殊的檔案
l	符號鏈結
s	Socket

例如，下面是 **ls -l** 範例清單：

```
-rwxrwxr-x   2   janet   acct   512 Mar 01 13:33 january
```

在此，第一個連字號 (-) 表示一般檔案。接下來的九個字元 (rwxrwxr-x) 代表「使用者」、「群組」及「其他」存取模式，如之前所討論。janet 是檔案擁有者，acct 則是 Janet 的群組名稱。512 是檔案大小（位元組）， Mar 01 13:33 是最近修改的日期及時間， january 是檔名。2 是指與該檔案鏈結的數目。

存取模式的數字表示法
在數值上，讀取權是以數值 4 表示，寫入許可權是以數值 2 表示，而執行許可權是以數值 1 表示。介於 1 到 7 的總數值是表示每一群組（使用者、群組及其他）的存取模式。

下表說明各層次存取權的數值：

總值	讀取	寫入	執行
0	-	-	-
1	-	-	1
2	-	2	-
3	-	2	1
4	4	-	-
5	4	-	1
6	4	2	-
7	4	2	1

在建立檔案時， 預設檔案存取模式為 755。這表示使用者具有讀取、寫入及執行許可權 (4+2+1=7)， 群組具有讀取及執行許可權 (4+1=5)，而所有其他使用者則具有讀取及執行許可權 (4+1=5)。若要變更您所擁有之檔案的存取權模式，請執行 **chmod**（變更模式）指令。

顯示群組資訊
使用 **lsgroup** 指令來顯示系統上所有群組（或指定的群組）的屬性。如果無法讀取一個以上屬性，**lsgroup** 指令將盡可能列出所有能讀取的資訊。

屬性資訊顯示成 *Attribute=Value* 定義，並由空格隔開。

1. 若要列出所有在系統上的群組，請鍵入：

   ```
   lsgroup ALL
   ```

系統將顯示每一個群組、群組 ID 和群組中所有使用者，清單如下所示：

```
system  0        arne,pubs,ctw,geo,root,chucka,noer,su,dea,backup,build,janice,denise
staff   1        john,ryan,flynn,daveb,jzitt,glover,maple,ken,gordon,mbrady
bin     2        root,bin
sys     3        root,su,bin,sys
```

2. 若要顯示所有群組的特定屬性，請執行下列其中一項：

   - 您可以用 *Attribute=Value* 格式，來列出屬性，並以空格隔開。此為預設樣式。例如，若要列出系統上所有群組的 ID 和使用者，請鍵入：

     ```
     lsgroup -a id users ALL | pg
     ```

     會顯示類似以下的清單：

     ```
     system id=0 users=arne,pubs,ctw,geo,root,chucka,noer,su,dea,backup,build
     staff id=1 users=john,ryan,flynn,daveb,jzitt,glover,maple,ken
     ```

   - 您也可以使用段落格式來列出資訊。例如，若要列出以段落格式列出系統上所有群組的 ID 和使用者，請鍵入：

     ```
     lsgroup -a -f id users ALL | pg
     ```

     會顯示類似以下的清單：

     ```
     system:
        id=0
        users=pubs,ctw,geo,root,chucka,noer,su,dea,backup,build

     staff:
        id=1
        users=john,ryan,flynn,daveb,jzitt,glover,maple,ken

     bin:
        id=2
        users=root,bin

     sys:
        id=3
        users=root,su,bin,sys
     ```

3. 若要顯示特定群組的所有屬性，您可以使用下列其中一種樣式來列出所有群組的特定屬性：

   - 您可以用 *Attribute=Value* 格式，來列出每一個屬性，並以空格隔開。此為預設樣式。例如，若要列出群組系統的所有屬性，請鍵入：

     ```
     lsgroup system
     ```

     會顯示類似以下的清單：

     ```
     system id=0 users=arne,pubs,ctw,geo,root,chucka,noer,su,dea,backup,build,janice,denise
     ```

   - 您也可以使用段落格式來列出資訊。例如，若要以段落格式列出群組 bin 的所有屬性，請鍵入：

     ```
     lsgroup -f system
     ```

     會顯示類似以下的清單：

     ```
     system:
        id=0   users=arne,pubs,ctw,geo,root,chucka,noer,su,dea,backup,build,janice,denise
     ```

4. 若要列出特定群組的特定屬性，請鍵入：

   ```
   lsgroup -a Attributes Group
   ```

例如，若要列出群組 bin 的 ID 和使用者，請鍵入：

```
lsgroup -a id users bin
```

會顯示類似以下的清單：

```
bin id=2 users=root,bin
```

請參閱 **lsgroup** 指令，以取得完整語法。

變更檔案或目錄許可權
使用 **chmod** 指令可以變更檔案的許可權。

1. 若要新增許可權類型到 chap1 及 chap2，請鍵入：

   ```
   chmod g+w chap1 chap2
   ```

   這會將新增群組成員之寫入許可權至檔案 chap1 及 chap2。
2. 若要一次變更 mydir 目錄的幾種許可權，請鍵入：

   ```
   chmod go-w+x mydir
   ```

   如此會拒絕 (**-**) 群組成員 (**g**) 及其他人 (**o**) 在 mydir 目錄中建立或刪除檔案 (**w**) 的許可權，並可讓 (**+**) 群組成員及其他人搜尋 mydir 目錄，或是將 (**x**) 用在路徑名稱中。這與下列指令序列相等：

   ```
   chmod g-w mydir
   chmod o-w mydir
   chmod g+x mydir
   chmod o+x mydir
   ```
3. 若只要許可擁有者使用 shell 程序 **cmd** 作為指令，請鍵入：

   ```
   chmod u=rwx,go= cmd
   ```

   這將提供讀取、寫入及執行許可權給擁有檔案的使用者 (**u=rwx**)。這也會拒絕群組和其他使用者以任何方法存取 cmd (**go=**) 的許可權。
4. 若要使用 **chmod** 指令的數值模式格式來變更 text 的許可權，請鍵入：

   ```
   chmod 644 text
   ```

   這將設定擁有者的讀取及寫入許可權，並設定群組及其他使用者的唯讀模式。

請參閱 **chmod** 指令，以取得完整語法。

存取控制清單
存取控制由受保護的資訊資源所組成，這些資源會指定誰有權存取這樣的資源。

作業系統允許「需要知道」或「絕對」的安全保護。資訊資源的擁有者可授與其他使用者讀取及寫入該資源的存取權。被授與資源存取權的使用者可轉送那些權利給其他使用者。此安全保護允許在系統中使用者控制的資訊流通；資訊資源的擁有者定義物件的存取許可權。

使用者僅對其所擁有的物件具有以使用者為基礎的存取權。一般而言，使用者接收資源的群組許可權或預設許可權。管理存取控制的主要工作是定義使用者的群組成員資格，因為這些成員資格決定了使用者對其所未擁有之檔案的存取權。

檔案系統物件的存取控制清單
一般來說，檔案系統物件會與「存取控制清單 (ACL)」相關聯，此清單通常是由一系列的「存取控制項目 (ACE)」所組成的。每一個 ACE 都會定義識別及其相關的存取權。

若要維護存取控制清單，請使用 **aclget**、**acledit**、**aclput** 及 **aclconvert** 指令。

請注意，一般來說，會利用實體檔案系統 (PFS) 在媒體上儲存及管理 ACL。AIX 作業系統會為實體檔案系統提供基礎架構，來支援及管理多種 ACL 類型。 AIX 隨附的 JFS2 檔案系統支援兩種 ACL 類型：

· AIXC

· NFS4

舊版的檔案系統僅支援如舊版 AIX 中的 AIXC ACL。這些 ACL 類型會在安全中有詳細的討論。

AIXC 存取控制清單類型
針對 ACL 行為提供 AIXC (AIX Classic) ACL 類型，如舊版 AIX 所定義。此 ACL 類型是由正規基本模式位元與延伸許可權 (ACE) 所組成的。

使用延伸許可權，您可以在不變更基本許可權的情況下，允許或拒絕特定個人或群組對檔案的存取。

註 : 檔案的 AIXC ACL 大小不能超過一個記憶體頁面（大約是 4096 個位元組）。

數值模式（八進位表示法）的 **chmod** 指令可以設定基本許可權及屬性。該指令所呼叫的 chmod 子常式可停用延伸許可權。如果您在具有 ACL 的檔案上使用數值模式的 **chmod** 指令，則會停用延伸許可權。當相關聯的 ACL 是屬於 AIXC 類型時， 符號模式的 **chmod** 指令便不會停用延伸許可權。如需數值及符號模式的詳細資訊，請參閱 **chmod** 指令。如需 chmod 指令的相關資訊，請參閱 **chmod**。

基本許可權

AIXC ACL 特定的基本許可權是指派給檔案擁有者、 檔案群組及其他使用者的傳統檔案存取模式。存取模式為讀取 (r)、寫入 (w) 及執行/搜尋 (x)。

註 : AIXC ACL 類型的基本許可權將會與儲存在檔案系統物件的 inode 標頭中的檔案模式位元相同。也就是說，在檔案系統物件上執行 **stat** 時，位在基本模式位元中的資訊會與檔案系統所傳回的值相同。

在存取控制清單中，基本許可權的格式如，使用 **Mode** 參數表示為 rwx（以連字號 (-) 代替每一個未指定的許可權）：

```
基本許可權：
      owner(name): Mode
      group(group): Mode
      others: Mode
```

屬性

有三個可新增到存取控制清單中的屬性：

setuid (SUID)
設定使用者 ID 模式位元。此屬性會將處理程序的有效且已儲存的使用者 ID 設定為執行中之檔案的擁有者 ID。

setgid (SGID)
設定群組 ID 模式位元。此屬性會將處理程序的有效且已儲存的群組 ID 設定為執行中之檔案的群組 ID。

savetext (SVTX)
以文字檔格式來儲存文字。

以上的屬性會以下列格式來新增：

```
屬性：SUID, SGID, SVTX
```

延伸許可權

AIXC ACL 延伸許可權可讓檔案的擁有者更精確地定義對該檔案的存取權。延伸許可權會修改基本檔案許可權（owner、group、others），方法為允許、拒絕或指定特定之個人、群組或使用者與群組組合的存取模式。您可使用關鍵字來修改許可權。

permit、deny 及 specify 關鍵字的定義如下：

permit
授與使用者或群組指定的檔案存取權

deny
限制使用者或群組使用所指定的檔案存取權

specify
精確定義使用者或群組的檔案存取權

如果 deny 或 specify 關鍵字拒絕使用者的某個存取權，則沒有其他項目可以置換該拒絕存取權。

必須在 ACL 中指定 enabled 關鍵字，才能使延伸許可權生效。預設值為 disabled 關鍵字。

在 AIXC ACL 中，延伸許可權為下列格式：

```
延伸許可權：
    enabled | disabled
        permit   Mode  UserInfo...:
        deny     Mode  UserInfo...:
        specify  Mode  UserInfo...:
```

針對每一個 permit、deny 或 specify 項目單獨使用一行。**Mode** 參數會以 `rwx`（以連字號 (-) 代替每一個未指定的許可權）來表示。**UserInfo** 參數會以 `u:UserName`、`g:GroupName`，或是以逗點隔開的 `u:UserName` 及 `g:GroupName` 組合來表示。

註： 如果在項目中指定了一個以上的使用者名稱，則該項目不能在存取控制決策中使用，因為一個處理程序僅能有一個使用者 ID。

NFS4 存取控制清單類型
AIX 中的 JFS2 檔案系統也支援 NFS4 ACL 類型。此 ACL 施行遵循 ACL 定義，如 NFS4 第 4 版通訊協定相關的 RFC 所指定。

此 ACL 在存取權方面提供更好的縝密度控制， 同時也提供其他特性（例如，繼承）。NFS4 ACL 是由 ACE 的陣列所組成的。每一個 ACE 都會為識別定義存取權。如 RFC 中所定義， NFS4 ACE 的主要元件如下：

```
struct nfsace4 {

                                acetype4          type;
                                aceflag4           flag;
                                acemask4        access_mask;
                                utf8str_mixed   who;
};
```

其中：

type
定義 ACE 類型的位元遮罩。在此詳細定義像是這個 ACE 允許存取還是拒絕存取...等的明細資訊。

flag
說明 ACE 在繼承方面的位元遮罩。定義此 ACE 適用於檔案系統物件、或是其子項、或是這兩者。

access_mask
定義各種可能的存取權之位元遮罩。定義的存取權包括讀取、寫入、執行、建立、刪除、建立子項、刪除子項...等。

who
這個以空字元結尾的字串定義此 ACE 將要套用的使用者身分。請注意，每一個 RFC（此字串大小並無限制）及寬鬆的定義，允許在 NFS 第 4 版網路中定義網域。本質上（大部分時間）AIX 不會解譯此字串，每一個 ACE 都會與 AIX 知道的識別身分（如 `uid` 或 `gid`）產生關聯。NFS 第 4 版檔案系統應該會在必要時，解譯這些字串並轉換成 OS 知道的使用者或群組 ID。AIX 只了解定義在 RFC 中的部分特殊 who 字串。

在 AIX 中，請使用 **aclget**、**acledit**、**aclput** 及 **aclconvert** 指令來管理 NFS4 ACL。

註： 任何類型的 **chmod** 指令，都會消除檔案的 ACL。

***AIXC* 的存取控制清單範例**
下列是 AIXC 存取控制清單 (ACL) 的範例。

下列是 AIXC ACL 的範例：

```
屬性：SUID
基本許可權：
            owner(frank):  rw-
            group(system): r-x
```

```
            others: ---
延伸許可權：
            enabled
                permit  rw-  u:dhs
                deny    r--  u:chas, g:system
                specify r--  u:john, g:gateway, g:mail
                permit  rw-  g:account, g:finance
```

ACL 的各部分及其意義如下所述：

- 第一行表示 setuid 位元已開啟。
- 介紹基本許可權的下一行是可選用的。
- 下面三行指定基本許可權。括弧內的擁有者及群組名稱僅為參考之用。變更這些名稱不會改變檔案擁有者或檔案群組。只有 **chown** 指令及 **chgrp** 指令可變更這些檔案屬性。如需這些指令的相關資訊，請參閱 **chown** 及 **chgrp**。
- 介紹延伸許可權的下一行是可選用的。
- 下一行指出後面的延伸許可權已啟用。
- 最後四行為延伸項目。
- 第一個延伸項目授與使用者 dhs 在檔案上的讀取 (r) 及 寫入 (w) 許可權。
- 第二個延伸項目只有在使用者 chas 是 system 群組的成員時，才會拒絕對該使用者的讀取權 (r)。
- 第三個延伸項目指定只要使用者 john 是 gateway 群組及 mail 群組兩者的成員，此使用者即有讀取權 (r)。如果使用者 john 不是上述兩個群組的成員，則不能引用這個延伸許可權。
- 最後一個延伸項目會將讀取 (r) 及寫入 (w) 許可權， 授予給**同時存在** account 群組及 finance 群組中的任何使用者。

註： 一個處理程序中可應用一個以上的延伸項目，但限制模式優先於許可模式。

如需相關資訊及完整語法，請參閱 *Commands Reference, Volume 1* 中的 **acledit** 指令。

存取控制清單存取授權

資訊資源的擁有者負責管理存取權。資源是以許可位元加以保護，而這些資源則包含在物件模式中。

在 AIXC ACL 中， 許可位元定義了授予物件、物件群組及其他預設類別的存取許可權。AIXC ACL 類型支援三種存取（讀取、寫入及執行）模式，這三種模式可個別授予。

當使用者登入帳戶時（使用 **login** 或 **su** 指令），指派給該帳戶的使用者 ID 及群組 ID 就會與使用者的處理程序產生關聯。這些 ID 決定此處理程序的存取權。

檔案、目錄、具名管道及裝置（特殊檔案）具有相關的 AIX ACL 時，存取的授權方式如下：

- 對於存取控制清單 (ACL) 中的每個存取控制項目 (ACE)，會比較 ID 清單與處理程序的 ID。如果相符，處理程序會收到定義該項目的許可權和限制。ACL 中每一個相符的項目，會運算出許可權和限制兩者的邏輯聯集。如果要求處理程序與 ACL 中的任何項目不符，則會收到預設項目的許可權和限制。
- 如果所要求的存取模式被允許（包含在許可權的聯集中）且不受限制（包含在限制的聯集中），則會授與存取權。否則，會拒絕存取。

此外，若是 AIXC ACL 類型，只要清單中所有的 ID 都符合要求處理程序的有效 ID 相對應類型，ACL 的 ID 清單便符合處理程序。符合的 USER 類型 ID，就是處理程序的有效使用者 ID，如果 Group 類型 ID，等於處理程序的有效群組 ID 或其中一個補充群組 ID，就是符合的 ID。例如，具下列 ID 清單的 ACE：

```
USER:fred, GROUP:philosophers, GROUP:software_programmer
```

會符合以 fred 為有效使用者 ID 且具有下列群組的處理程序：

```
philosophers, philanthropists, software_programmer, doc_design
```

但不符合以 fred 為有效使用者 ID 且具有下列群組的處理程序：

```
philosophers, iconoclasts, hardware_developer, graphic_design
```

請注意具有下列 ID 清單的 ACE 可符合這兩個處理程序：

```
USER:fred, GROUP:philosophers
```

換句話說，ACE 函數中的 ID 清單，是一組須保留給指定之授與存取權的條件。

任意存取控制機制可以有效控制對資訊資源的存取，並另外對資訊的機密性與完整性提供保護。使用者控制的存取控制機制的效能，會隨使用者的製作而定。所有使用者都必須瞭解如何授與和拒絕存取權，以及其設定方式。

請注意，與 NFS4 ACL 類型相關聯的檔案系統物件之存取權檢查是以各種 ACE 為基礎， 這些 ACE 會依在 NFS 第 4 版通訊協定相關的 RFC 中設定的每個規則，組成 ACL。根據 ACE 中所定義的使用者 ID、群組 ID 或特殊 who 字串，對處理程序的認證完成識別符合。如果有相符的項目，會對定義在 ACE 中的存取權檢查所要求的存取權。當任一存取權獲得允許，就會去除，並繼續對下一個 ACE 進行比較作業。此處理程序會繼續進行， 直到到達 ACL 的結尾或所有存取權都符合，或是所要求的任何一個存取權遭到拒絕。下列步驟會在檔案系統與 NFS4 ACL 相關聯的情況下，擷取存取檢查：

1. 對於存取控制清單 (ACL) 中的每個存取控制項目 (ACE)，會比較 ID 清單與處理程序的 ID。識別檢查包括定義在 ACE 中的使用者 ID 或群組 ID。同時，如果將識別定義為含有像是 OWNER@ 字串的 special， 若呼叫處理程序是依據檔案的擁有者，將會發生符合的狀況。如果有一相符者，處理程序會收到為該項目定義的許可權。否則會繼續進行下一個 ACE。
2. 所要求的存取權會與從 ACE 項目中擷取的存取權相比較。如果 ACE 明確拒絕任何要求的存取權， 那麼存取檢查程序會停止，且要求處理程序就會是被拒絕存取。
3. 如果 ACE 符合部分要求的存取權，那麼這些存取權便會從要求的存取權清單中清除，且會繼續對下一個 ACE 進行比對作業。
4. 如果 ACE 符合所有所要求的存取權，便允許所要求的存取權。
5. 如果在解析所有所要求的存取權之前到達 ACL 結尾，則會拒絕存取。

請注意，除了 ACL 類型為主的存取檢查，個別實體檔案系統可能也會選擇提供根據特殊權限的存取權給檔案系統物件。例如，某個擁有者可能一直都有權修改 ACL，不論目前的 ACL 存取權是什麼。使用者 ID 為 0 的處理程序即為 root 使用者處理程序。一般而言，允許這些處理程序擁有所有的存取許可權。不過，如果 root 使用者處理程序要求程式的執行權限，則只有在執行權限授與一個以上的使用者時，才會授與存取權。

第一次存取物件時，會在系統呼叫層次檢查這些物件的所有存取權。因為存取 System V 跨處理程序通訊 (SVIPC) 物件並沒有區域限制，所以每一次存取都會加以檢查。但是，實體檔案系統可能會在檔案系統物件開啟的時候，而不是在讀取或寫入作業時進行檢查。針對具有檔案系統名稱的物件，須能分辨實際物件的名稱。名稱的解析會是相對（到處理程序的工作目錄），或是絕對（到處理程序的根目錄）。所有的名稱分辨開始於搜尋這些目錄中的一個。

顯示存取控制資訊的指令（*aclget* 指令）

aclget 指令會顯示檔案的存取控制資訊。您所檢視的資訊包括屬性、基本許可權及延伸許可權。

例如，若要顯示 status 檔案的存取控制資訊，請鍵入：

```
aclget 狀態
```

所顯示的存取控制資訊包括屬性、基本許可權及延伸許可權的清單。

請參閱 *Commands Reference, Volume 1* 中的 **aclget** 指令，以取得完整語法。

相關概念

存取控制清單範例及說明

下列是存取控制清單 (ACL) 的範例及說明。

設定存取控制資訊（*aclput* 指令）

若要設定檔案的存取控制資訊，請使用 **aclput** 指令。

註： 檔案之存取控制清單的大小，不能超過一個記憶體頁面（大約 4096 位元組）。

請參閱下列範例：

例如，若要使用儲存在 acldefs 檔案內的存取控制資料，來設定狀態檔案的存取控制資訊，請鍵入：

```
aclput -i acldefs status
```

若要使用 plans 檔案所用之相同資訊來設定狀態檔案的存取控制資訊，請鍵入：

```
aclget plans | aclput status
```

如需相關資訊及完整語法，請參閱 *Commands Reference, Volume 1* 中的 **aclput** 指令。

存取控制清單範例及說明

下列是存取控制清單 (ACL) 的範例及說明。

下列為一個 ACL 的範例：

```
屬性：SUID
基本許可權：
             owner(frank):  rw-
             group(system): r-x
             others: ---
延伸許可權：
             enabled
                 permit  rw-  u:dhs
                 deny    r--  u:chas, g:system
                 specify r--  u:john, g:gateway, g:mail
                 permit  rw-  g:account, g:finance
```

ACL 的各部分及其意義分述如下：

- 第一行表示 **setuid** 位元已開啟。
- 介紹基本許可權的下一行是可選用的。
- 下面三行指定基本許可權。括弧內的擁有者及群組名稱僅為參考之用。變更這些名稱不會改變檔案擁有者或檔案群組。只有 **chown** 指令及 **chgrp** 指令可變更這些檔案屬性。如需這些指令的相關資訊，請參閱 **chown** 及 **chgrp**。
- 介紹延伸許可權的下一行是可選用的。
- 下一行指出後面的延伸許可權已啟用。
- 最後四行為延伸項目。第一個延伸項目授與使用者 dhs 在檔案上的讀取 (r) 及 寫入 (w) 許可權。
- 第二個延伸項目只有在使用者 chas 是 syetem 群組的成員時，才會拒絕對該使用者的讀取權 (r)。
- 第三個延伸項目指定只要使用者 john 是 gateway 群組及 mail 群組兩者的成員，即有讀取權 (r)。如果使用者 john 不是上述兩個群組的成員，則不能引用這個延伸許可權。
- 最後一個延伸項目會將讀取 (r) 及寫入 (w) 許可權， 授予給**同時存在** account 群組及 finance 群組中的任何使用者。

  註：一個處理程序中可應用一個以上的延伸項目，但限制模式優先於許可模式。

  請參閱 *Commands Reference, Volume 1* 中的 **acledit** 指令，以取得完整語法。

相關概念

顯示存取控制資訊的指令（aclget 指令）

aclget 指令會顯示檔案的存取控制資訊。您所檢視的資訊包括屬性、基本許可權及延伸許可權。

相關工作

編輯存取控制資訊（acledit 指令）

使用 **acledit** 指令來變更檔案的存取控制資訊。此指令顯示現行存取控制資訊，並可讓檔案擁有者來變更它。

編輯存取控制資訊（*acledit* 指令）

使用 **acledit** 指令來變更檔案的存取控制資訊。此指令顯示現行存取控制資訊，並可讓檔案擁有者來變更它。

在做任何永久變更之前，此指令會詢問您是否要執行。如需 **acledit** 指令的相關資訊，請參閱 **acledit**。

註 : 必須以完整的路徑名稱來指定 *EDITOR* 環境變數；否則 **acledit** 指令將會失敗。

所顯示的存取控制資訊是 ACL 類型特有的，並且會包括屬性、基本許可權及延伸許可權的清單。

例如，若要編輯 plans 檔案的存取控制資訊，請鍵入：

```
acledit plans
```

請參閱 *Commands Reference, Volume 1* 中的 **acledit** 指令，以取得完整語法。

相關概念
存取控制清單範例及說明
下列是存取控制清單 (ACL) 的範例及說明。

鎖定您的終端機（lock 或 xlock 指令）
使用 **lock** 指令來鎖定您的終端機。**lock** 指令會要求您密碼，讀取它，並再要求一次密碼，以求驗證。

在此期間，指令鎖定終端機，直到第二次接到密碼時才解除。逾時預設值是 15 分鐘，但可使用 **-***Number* 旗標來改變。

註 : 如果您的介面是 AIXwindows，請以相同的方式來使用 **xlock** 指令。

例如，若要以密碼控制鎖定終端機，請鍵入：

```
lock
```

系統會要求您輸入兩次密碼，以便進行驗證。若未於 15 分鐘內重複輸入密碼，則指令會逾時。

若要保留終端機受密碼控制，且逾時間隔為 10 分鐘，請鍵入：

```
lock -10
```

請參閱 位於 *Commands Reference* 中的 **lock** 或 **xlock** 指令，以取得完整語法。

>|

鑑別

xlock 指令是具有「可插入鑑別模組 (PAM)」功能的 X 伺服器指令，其會鎖定 X 伺服器，直到使用者輸入密碼為止。它同時支援本端 UNIX 鑑別及 PAM 鑑別，用於解除鎖定 X 伺服器。

您可以設定系統層面的配置來將 PAM 用於鑑別，方法是透過提供 root 使用者存取權，以及對 /etc/security/login.cfg 檔案的 **usw** 段落中的 *PAM_AUTH* 修改 *auth_type* 屬性值。

PAM 啟用時使用的鑑別機制取決於 /etc/pam.conf 檔案中登入服務的配置。**xlock** 指令需要 **auth**、**account**、**password** 及 **session** 模組類型的 /etc/pam.conf 檔案項目。下列是針對 **xlock** 指令中 /etc/pam.conf 檔案項目建議的配置：

```
xlock           auth            required        pam_aix
xlock           account         required        pam_aix
xlock           password        required        pam_aix
xlock           session         required        pam_aix
```

|<

檔案及系統安全的指令摘要
下列是檔案系統及安全的指令。

項目	說明
acledit	編輯檔案的存取控制資訊
aclget	顯示檔案的存取控制資訊

項目	說明
aclput	設定檔案的存取控制資訊
chmod	變更許可權模式
chown	變更與檔案相關聯的使用者
lock	保留終端機
lsgroup	顯示群組的屬性
xlock	鎖定本端 X 顯示器，直到輸入密碼為止

使用者環境

每一個登入名稱都有它自己的系統環境。

系統環境是一個區域，在這個區域中儲存了在階段作業中執行之所有程序的持有率資訊。您可以使用數種指令來顯示您系統的相關資訊。

使用者環境檔案及自訂程序

這些檔案及程序可協助使用者自訂系統環境。

系統啟動檔案

項目	說明
/etc/profile	含有指令的系統檔，這些指令是您登入時系統所執行的指令。
/etc/environment	含有變數的系統檔，這些變數指定所有處理程序的基本環境。
$HOME/.profile	位於起始目錄中的檔案，含有您登入時用來置換系統 **/etc/profile** 的指令。如需相關資訊，請參閱 .profile file。
$HOME/.env	位於您的起始目錄中的檔案，會置換系統 **/etc/environment**，並且其中會包含一些變數，用以指定全部處理程序的基本環境。如需相關資訊，請參閱 .env 檔案。

AIXwindows 啟動檔案

項目	說明
$HOME/.xinitrc	位於您的起始目錄中的檔案，會負責控制您啟動 AIXwindows 時，所啟動的視窗和應用程式。如需相關資訊，請參閱 .xinitrc 檔案。
$HOME/.Xdefaults	位於您的起始目錄中的檔案，會負責控制 AIXwindows 資源的視覺或行為方面。如需相關資訊，請參閱 第 280 頁的『.Xdefaults 檔』。
$HOME/.mwmrc	位於您的起始目錄中的檔案，會定義視窗管理程式的按鍵連結、滑鼠按鈕連結和功能表定義。如需相關資訊，請參閱 第 280 頁的『.mwmrc 檔』。

自訂程序

項目	說明
PS1	標準系統提示
PS2	其他輸入系統提示
PS3	Root 系統提示
chfont	改變系統重新啟動時，顯示器所使用的字型
stty	設定、重設和報告工作站作業參數

系統裝置清單（lscfg 指令）

若要顯示在現行配置中找到的每個裝置的名稱、位置及說明，請使用 **lscfg** 指令。該清單以裝置位置來排序。

例如，若要列出系統內配置的裝置，請在提示時鍵入：

```
lscfg
```

系統會顯示與下列相似的輸出：

```
INSTALLED RESOURCE LIST

The following resources are installed on your machine.

+/- = Added/Deleted from Diagnostic Test List.
*   = NOT Supported by Diagnostics.


  Model Architecture: chrp
  Model Implementation: Multiple Processor, PCI bus

+  sysplanar0   00-00            CPU Planar
+  fpa0         00-00            Floating Point Processor
+  mem0         00-0A            Memory Card
+  mem1         00-0B            Memory Card
+  ioplanar0    00-00            I/O Planar
+  rs2320       00-01            RS232 Card
+  tty0         00-01-0-01       RS232 Card Port
-  tty1         00-01-0-02       RS232 Card Port
   ..
   ..
   ..
```

裝置清單不是單純以裝置位置來排序的。它也以上下代階層排序。如果母項具有多重子項，則會以裝置位置來將子項排序。如果子項具有相同的裝置位置，就會以軟體取得子項的順序來顯示。若要顯示特定裝置的相關資訊，您可使用 **-l** 旗標。例如，若要列出裝置 **sysplanar0** 的資訊，請鍵入：

```
lscfg -l sysplanar0
```

系統會顯示與下列相似的輸出：

```
DEVICE          LOCATION     DESCRIPTION

sysplanar0      00-00        CPU Planar
```

您亦可使用 **lscfg** 指令來顯示重要產品資料 (VPD)，例如產品編號、序號以及工程變更層次。對於某些裝置而言，會自動收集 VPD，並且新增至系統配置。對於其他裝置而言，則要自行鍵入 VPD。資料前面的 ME 表示資料是手動鍵入的。

例如，若要列出系統中配置之裝置的 VPD，請在提示時鍵入：

```
lscfg -v
```

系統會顯示與下列相似的輸出：

```
INSTALLED RESOURCE LIST WITH VPD
```

```
下列資源會安裝在您的機器中。
```

```
  Model Architecture: chrp
  Model Implementation: Multiple Processor, PCI bus
```

```
sysplanar0      00-00        CPU Planar
```

```
   Part Number.........342522
   EC Level............254921
   Serial Number.......353535
```

```
fpa0   00-00   Floating Point Processor
mem0   00-0A   Memory Card
```

```
   EC Level............990221
.
.
.
```

請參閱 *Commands Reference, Volume 3* 中的 **lscfg** 指令，以取得完整語法。

顯示主控台名稱
若要將現行主控台裝置的名稱寫入標準輸出（通常是螢幕畫面），請使用 **lscons** 指令。

例如，在提示時鍵入：

```
lscons
```

系統會顯示與下列相似的輸出：

```
/dev/lft0
```

請參閱 **lscons** 指令，以取得完整語法。

顯示終端機名稱（tty 指令）
若要顯示終端機的名稱，請使用 **tty** 指令。

例如，在提示時鍵入：

```
tty
```

系統會顯示如下的資訊：

```
/dev/tty06
```

在此範例中，tty06 為終端機名稱，而 **/dev/tty06** 為包含此終端機介面的裝置檔案。

請參閱 *Commands Reference, Volume 5* 中的 **tty** 指令，以取得完整語法。

列出可用的顯示畫面（lsdisp 指令）
若要列出您系統上目前可用的顯示器，以提供每個顯示器的識別名稱、介面槽號碼、 顯示器名稱和每個顯示器的說明，請使用 **lsdisp** 指令。

例如，若要列出所有可用的變數，請鍵入：

```
lsdisp
```

下列是輸出的範例。清單次序以介面槽號碼的升序順序為根據。

```
Name   Slot    Name        Description
ppr0   00-01   POWER_G4    Midrange Graphics Adapter
gda0   00-03   colorgda    Color Graphics Display Adapter
ppr1   00-04   POWER_Gt3   Midrange Entry Graphics Adapter
```

請參閱 *Commands Reference, Volume 3* 中的 **lsdisp** 指令，以取得完整語法。

列出可用的字型組（lsfont 指令）
若要顯示可用於您的顯示器的字型組清單，請使用 **lsfont** 指令。

例如，若要列出所有顯示畫面可用的字型，請鍵入：

```
lsfont
```

下列是輸出範例，顯示出字型 ID、檔名、字符大小及字型編碼：

```
FONT  FILE            GLYPH  FONT
ID    NAME            SIZE   ENCODING
====  ==============  =====  =========
0     Erg22.iso1.snf  12x30  ISO8859-1
1     Erg11.iso1.snf   8x15  ISO8859-1
```

請參閱 *Commands Reference, Volume 3* 中的 **lsfont** 指令，以取得完整語法。

列出現行軟體鍵盤對映（lskbd 指令）

若要顯示目前載入系統之軟體鍵盤對映的絕對路徑名稱，請使用 **lskbd** 指令。

例如，若要列出您現行的鍵盤對映，請鍵入：

```
lskbd
```

下面是由 **lskbd** 指令所顯示的清單範例：

```
The current software keyboard map = /usr/lib/nls/loc/C.lftkeymap
```

列出可用的軟體產品（lslpp 指令）

若要顯示您系統中可用的軟體產品之相關資訊，請使用 **lslpp** 指令。

例如，若要列出系統中所有的軟體產品，請在系統提示時鍵入：

```
lslpp -l -a
```

下列是輸出的範例：

```
Fileset                 Level  State     Description
------------------------------------------------------------------
Path: /usr/lib/objrepos
   X11_3d.gl.dev.obj           APPLIED   AIXwindows/3D GL
                                         Development Utilities
Fonts
  X11fnt.oldX.fnt              APPLIED   AIXwindows Miscellaneous
                                         X Fonts
X11mEn_US.msg                  APPLIED   AIXwindows NL Message
                                         files
```

```
.
```

```
.
```

```
.
```

如果報表很長，頂端部分可能會捲出畫面之外。若要一次顯示一頁（畫面）的報表，請使用 **lslpp** 指令加上 **pg** 指令。在提示時鍵入：

```
lslpp -l -a | pg
```

請參閱 *Commands Reference, Volume 3* 中的 **lslpp** 指令，以取得完整語法。

列出您的終端機所用的控制鍵指定（stty 指令）

若要顯示您的終端機設定，請使用 **stty** 指令。請特別注意您的終端機是用哪些按鍵來作為控制鍵。

例如，在提示時鍵入：

```
stty -a
```

系統會顯示如下的資訊：

```
.
```

```
.
```

```
.
```

```
intr = ^C; quit = ^\; erase = ^H; kill = ^U; eof = ^D;
eol = ^@ start = ^Q; stop = ^S; susp = ^Z; dsusp = ^Y;
reprint = ^R discard = ^O; werase = ^W; lnext = ^V
```

```
.
```

```
.
```

```
.
```

在本例中，像 `intr = ^C; quit = ^\; erase = ^H;` 這些行會顯示您的控制鍵設定。^H 鍵是倒退鍵，它的設定是執行「消除」功能。

如果報表很長，頂端部分可能會捲出畫面之外。若要一次顯示一頁（畫面）的報表，請使用 **stty** 指令加上 **pg** 指令。在提示時鍵入：

```
stty -a | pg
```

請參閱 *Commands Reference, Volume 5* 中的 **stty** 指令，以取得完整語法。

相關概念

前景處理程序取消

如果您啟動了前景處理程序，然後決定不要完成它，您可按 INTERRUPT 來取消。這通常是 Ctrl-C 或 Ctrl-Backspace。

列出環境變數（env 指令）

若要顯示您現行的環境變數，請使用 **env** 指令。您的所有處理程序都能存取的環境變數稱為廣域變數。

例如，若要列出所有環境變數及其相關值，請鍵入：

```
env
```

下列是輸出的範例：

```
TMPDIR=/usr/tmp
myid=denise
LANG=En_US
UNAME=barnard
PAGER=/bin/pg
VISUAL=vi
PATH=/usr/ucb:/usr/lpp/X11/bin:/bin:/usr/bin:/etc:/u/denise:/u/denise/bin:/u/bin1
MAILPATH=/usr/mail/denise?denise has mail !!!
MAILRECORD=/u/denise/.Outmail
EXINIT=set beautify noflash nomesg report=1 showmode showmatch
        EDITOR=vi
PSCH=>
HISTFILE=/u/denise/.history
LOGNAME=denise
MAIL=/usr/mail/denise
PS1=denise@barnard:${PWD}>
PS3=#
PS2=>
0epath=/usr/bin
USER=denise
SHELL=/bin/ksh
HISTSIZE=500
HOME=/u/denise
FCEDIT=vi
TERM=lft
MAILMSG=**YOU HAVE NEW MAIL. USE THE mail COMMAND TO SEE YOUR PWD=/u/denise
ENV=/u/denise/.env
```

如果報表很長，則頂端部分會捲出畫面之外。若要一次顯示一頁（畫面）的報表，請使用 **env** 指令加入 **pg** 指令。在提示時鍵入：

```
env | pg
```

請參閱 *Commands Reference, Volume 2* 中的 **env** 指令，以取得完整語法。

顯示環境變數值（printenv 指令）

若要顯示環境變數的值，請使用 **printenv** 指令。

如果您指定 **Name** 參數，則系統只會列印與您所要求的變數相關的值。如果沒有指定 **Name** 參數，**printenv** 指令就會顯示所有現行的環境變數， 每行顯示一組 **Name** =*Value* 序列。

例如，若要找出 *MAILMSG* 環境變數的現行設定，請鍵入：

```
printenv MAILMSG
```

該指令會傳回 *MAILMSG* 環境變數的值。例如：

```
YOU HAVE NEW MAIL
```

請參閱 *Commands Reference, Volume 4* 中的 **printenv** 指令，以取得完整語法。

雙向語言（aixterm 指令）

aixterm 指令支援 阿拉伯文及希伯來文，它們是雙向語言。

雙向語言可以從兩個方向（從左到右以及從右到左）來閱讀與撰寫。您開啟指定為阿拉伯文或希伯來文語言環境的視窗，便可使用阿拉伯文和希伯來文應用程式。

請參閱 *Commands Reference, Volume 1* 中的 **aixterm** 指令，以取得完整語法。

使用者環境及系統資訊的指令摘要

下列是使用者環境及系統資訊的指令。

項目	說明
aixterm	可讓您使用雙向語言。
env	顯示目前的環境，或設定指令的執行環境
lscfg	顯示裝置的診斷資訊
lscons	顯示現行主控台的名稱
lsdisp	列出系統上目前可使用的顯示器
lsfont	列出顯示器所能使用的字型
lskbd	列出系統目前所載入的鍵盤對映
lslpp	列出軟體產品
printenv	顯示環境變數值
stty	顯示系統設定值
tty	顯示終端機的完整路徑名稱

使用者環境自訂

作業系統提供各種指令及起始設定檔案，可讓您自訂使用者環境的行為及外觀。

您也可以自訂某些用在系統上之應用程式的預設資源。預設值是由程式在啟動時起始設定。當您變更預設值時，必須結束程式並重新啟動，讓新的預設值生效。

有關自訂「一般桌上管理系統環境」之行為及外觀的相關資訊，請參閱 *Common Desktop Environment 1.0: Advanced User's and System Administrator's Guide*。

系統啟動檔案

登入時，shell 讀取您已設定的起始設定檔之後會定義您的使用者環境。根據提供給環境變數的值來定義使用者環境的性質。您要維護此環境，直到登出系統為止。

當您登入作業系統時，shell 會使用兩種類型的設定檔。它評估這些檔案所含的指令，然後執行指令來設定系統環境。這些檔案皆具有類似的功能，除了 /etc/profile 檔案會控制系統上全部使用者的 profile 變數，以及 .profile 檔案可讓您自訂自己的環境，這兩種功能不同以外。

shell 會先執行指令以在 /etc/environment 檔案中設定您的系統環境， 然後評估包含在 /etc/profile 檔案中的指令。執行這些檔案之後，系統會接著檢查在您的起始目錄中是否具有 .profile 檔。若有 .profile 檔案，系統便會執行此檔案。.profile 檔案將會指定是否也存在一個環境檔。若有一個環境檔（通常稱為 .env），則系統會執行此檔案並設定您的環境變數。

登入時會執行一次 /etc/environment、/etc/profile 及 .profile 檔。另一方面，每次開啟新 shell 或視窗時都會執行一次 .env 檔。

/etc/environment 檔案

登入時作業系統使用的第一個檔案是 /etc/environment 檔案。 /etc/environment 檔案包含一些變數，這些變數指定全部處理程序的基本環境。

開始執行新的處理程序時，exec 子常式會製作一個可用的字串陣列， 其格式為 *Name=Value*。這個字串陣列就叫做環境。其中一個字串所定義的每個名稱就叫做環境變數或 *shell* 變數。exec 子常式可以一次設定整個環境。

當您登入時，系統會先從 /etc/environment 檔來設定環境變數， 然後才讀取您的登入設定檔（名稱為 .profile）。 下列變數組成基本環境：

項目	說明
HOME	使用者登入的完整路徑名稱或 HOME 目錄。login 程式將此變數設定為 /etc/passwd 檔案中指定的名稱。
LANG	目前作用中的語言環境名稱。*LANG* 變數是在安裝時，起始設定在 /etc/profile 檔中。
NLSPATH	訊息型錄的完整路徑名稱。
LOCPATH	「國際語言支援 (NLS)」表格的位置之完整路徑名稱。
PATH	在尋找路徑名稱不完整的指令時，指令（如：**sh**、**time**、**nice** 及 **nohup**）用來搜尋的目錄順序。
TZ	時區資訊。*TZ* 環境變數是由系統登入設定檔 /etc/profile 起始設定。

如需 /etc/environment 檔的詳細資訊，請參閱 *Files Reference*。

/etc/profile 檔案

登入時作業系統使用的第二個檔案是 /etc/profile 檔案。

/etc/profile 檔案控制全系統的預設變數，例如：

- 匯出變數
- 檔案建立遮罩 (umask)
- 終端機類型
- 郵件訊息，指出新郵件到達的時間

系統管理者會為系統上的全部使用者配置 /etc/profile 檔。僅系統管理者能夠變更此檔案。

下列範例是典型的 /etc/profile 檔案：

```
#設定檔案建立遮罩
umask 022
#當新郵件到達時告知我
```

```
MAIL=/usr/mail/$LOGNAME
#將我的 /bin 目錄新增至 shell 搜尋順序
PATH=/usr/bin:/usr/sbin:/etc::
#設定終端機類型
TERM=lft
#讓部分環境變數成為廣域變數
export MAIL PATH TERM
```

如需 /etc/profile 檔案的詳細資訊，請參閱 *Files Reference*。

.profile 檔案
.profile 檔位於您的起始 ($HOME) 目錄中，並且可讓您自訂個人的工作環境。

因為 .profile 檔是隱藏檔，所以請使用 **ls -a** 指令來列出它。

當 **login** 程式將 *LOGNAME*（登入名稱）及 *HOME*（登入目錄）變數新增至環境中後，即會執行 $HOME/.profile 檔案中的指令（如果這個檔案存在的話）。.profile 檔包含您的個人設定檔，該檔案會置換 /etc/profile 檔案中設定的變數。 .profile 檔通常是用來設定匯出環境變數和終端機模式。您可以修改 .profile 檔案來自訂您的環境。使用 .profile 檔案來控制下列預設值：

· 要開啟的 shell

· 提示外觀

· 鍵盤聲音

下列範例是典型的 .profile 檔案：

```
PATH=/usr/bin:/etc:/home/bin1:/usr/lpp/tps4.0/user::
epath=/home/gsc/e3:
export PATH epath
csh
```

此範例已定義兩個路徑變數（*PATH* 及 *epath*）、匯出它們，以及開啟 C shell (csh)。

您也可以使用 .profile 檔案（如果該檔案不存在，則使用 /etc/profile 檔案）來決定登入 shell 變數。您亦可自訂其他 shell 環境。例如，在啟動每一種 shell 類型之後，使用 .cshrc 檔案及 .kshrc 檔案，分別自訂 C shell 及 Korn shell。

.env 檔
如果 .profile 含有下列指令行：export ENV=$HOME/.env，則作業系統在登入時所使用的第四個檔案為 .env 檔

.env 檔可讓您自訂個人工作環境變數。因為 .env 檔是隱藏檔，因此請使用 **ls -a** 指令來列出它。如需 **ls** 指令的相關資訊，請參閱 **ls**。.env 檔包含個別使用者環境變數，這些環境變數會置換那些設定於 /etc/environment 檔中的變數。 您可以視需要修改 .env 檔案，以自訂您的環境變數。

下例範例是典型的 .env 檔：

```
export  myid=`id | sed -n -e 's/).*$//' -e 's/^.*(//p'`
#set prompt: login & system name & path
if [ $myid = root ]
        then    typeset -x PSCH='#:\${PWD}> '
                PS1="#:\${PWD}> "
        else    typeset -x PSCH='>'
                PS1="$LOGNAME@$UNAME:\${PWD}> "
                PS2=">"
                PS3="#?"
fi
export PS1 PS2 PS3
#setup my command aliases
alias   ls="/bin/ls -CF" \
        d="/bin/ls -Fal | pg" \
        rm="/bin/rm -i" \
        up="cd .."
```

註： 修改 .env 檔案時，請確定新建立的環境變數不會與標準變數（例如 *MAIL*、*PS1*、*PS2* 及 *IFS* 等）發生衝突。

AIXwindows 啟動檔案

不同的電腦系統有不同的方法來啟動「X 伺服器」及 AIXwindows。

因為不同的電腦系統有不同的方法來啟動 X 伺服器和 AIXwindows，請向您的系統管理者查詢啟動的方式。通常，X 伺服器和 AIXwindows 是從您登入時自動執行的 shell Script 來啟動的。然而，您可能發現您需要啟動 X 伺服器或 AIXwindows，或同時啟動這兩者。

如果您登入後發現您的顯示器是當作單一終端機使用，而沒有顯示任何視窗，您可以鍵入下列指令來啟動 X 伺服器：

```
xinit
```

註： 在輸入此指令之前，請確定指標停在有系統提示的視窗內。

如果此指令沒有啟動 X 伺服器，請詢問您的系統管理者，以確定您的搜尋路徑包含具有可執行檔程式的 X11 目錄。自一系統到另一系統的適當路徑可能不同。

如果您登入後發現一或多個沒有頁框的視窗，您可以鍵入以下指令來啟動「AIXwindows 視窗管理程式」：

```
mwm &
```

因為 AIXwindows 允許程式設計師和使用者撰寫 AIXwindows 應用程式以及自訂，所以您可能會發現滑鼠按鈕或其他功能 不同於您閱讀這份文件後所預期的操作。您可以將您的 AIXwindows 環境重設為預設行為，方法是按住下列四個按鍵：

Alt-Ctrl-Shift-!

您重新按下此鍵順序，便會返回自訂行為。如果您的系統不允許此種按鍵組合，您也可自預設的根功能表還原預設值行為。

.xinitrc 檔

xinit 指令使用可自訂的 shell Script 檔案，它會列出要啟動的 X 用戶端程式。當您啟動 AIXwindows 時，起始目錄中的 `.xinitrc` 檔案會負責控制所啟動的視窗及應用程式。

xinit 指令會依照下列次序，與 shell Script 搭配使用：

1. **xinit** 指令首先會尋找 *$XINITRC* 環境變數來啟動 AIXwindows。
2. 如果找不到 *$XINITRC* 環境變數，**xinit** 指令會尋找 $HOME/.xinitrc shell Script。
3. 如果找不到 $HOME/.xinitrc shell Script，則 **xinit** 指令會啟動 /usr/lib/X11/$LANG/xinitrc shell Script。
4. 如果找不到 /usr/lib/X11/$LANG/xinitrc，它就找 /usr/lpp/X11/defaults /$LANG/xinitrc shell Script。如果找不到該 Script，它就搜尋 /usr/lpp/X11/defaults/xinitrc shell Script。
5. xinitrc shell Script 會啟動指令，如 **mwm**（AIXwindows 視窗管理程式）、**aixterm** 及 **xclock** 等指令。

xinit 指令執行下列作業：

- 在現行顯示器上啟動 X 伺服器
- 設定 *$DISPLAY* 環境變數
- 執行 xinitrc 檔案以啟動 X 用戶端程式

下列範例顯現您可自訂的 xinitrc 檔案部分：

```
# This script is invoked by /usr/lpp/X11/bin/xinit

.
.
.
#***************************************************************
#  Start the X clients. Change the following lines to         *
#  whatever command(s) you desire!                           *
#  The default clients are an analog clock (xclock), an lft  *
#  terminal emulator (aixterm), and the Motif Window  Manager  *
```

```
#  (mwm).  *
#*****************************************************************
exec mwm
```

.Xdefaults 檔
如果您使用 AIXwindows 介面執行工作，您可以使用 .Xdefaults 檔案來自訂這個介面。AIXwindows 可讓您指定您在視覺化方面的喜好設定，如顏色及字型。

Windows 作業系統型應用程式的外觀及行為，在許多方面被稱為資源 的變數集所控制。資源的視覺化或行為方面，是由它的指派值所決定。有許多不同類型的資源值。例如，您可指派預定值給用來控制顏色的資源，如：*DarkSlateBlue* 或 *Black*。指派指定維度的資源為數值。某些資源採取布林值（真或 假）。

如果您的起始目錄中沒有 .Xdefaults 檔案，您可用任何文字編輯器來建立一個。起始目錄中有了此檔案之後，您就可以在其中隨意設定資源值。名為 Xdefaults.tmpl 的範例預設值位於 /usr/lpp/X11/defaults 目錄中。

下列範例顯現一般 .Xdefaults 檔案的部分：

```
*AutoRaise: on
*DeIconifyWarp: on
*warp:on
*TitleFont:andysans12
*scrollBar: true
*font: Rom10.500
Mwm*menu*foreground: black
Mwm*menu*background: CornflowerBlue
Mwm*menu*RootMenu*foreground: black
Mwm*menu*RootMenu*background: CornflowerBlue
Mwm*icon*foreground: grey25
Mwm*icon*background: LightGray
Mwm*foreground: black
Mwm*background: LightSkyBlue
Mwm*bottomShadowColor: Blue1
Mwm*topShadowColor: CornflowerBlue
Mwm*activeForeground: white
Mwm*activeBackground: Blue1
Mwm*activeBottomShadowColor: black
Mwm*activeTopShadowColor: LightSkyBlue
Mwm*border: black
Mwm*highlight:white
```

```
aixterm.foreground: green
aixterm.background: black
aixterm.fullcursor: true
aixterm.ScrollKey: on
aixterm.autoRaise: true
aixterm.autoRaiseDelay: 2
aixterm.boldFont:Rom10.500
aixterm.geometry: 80x25
aixterm.iconFont: Rom8.500
aixterm.iconStartup: false
aixterm.jumpScroll: true
aixterm.reverseWrap: true
aixterm.saveLines: 500
aixterm.scrollInput: true
aixterm.scrollKey: false
aixterm.title: AIX
```

.mwmrc 檔
您可使用在您 .Xdefaults 檔案中的資源，來設定大部分您要自訂的特性。然而，您視窗管理程式的鍵連結、滑鼠按鈕及功能表定義被指定在補充的 .mwmrc 檔案中，該檔案被 .Xdefaults 檔案中的資源所參考。

如果您的起始目錄中沒有 .mwmrc 檔案，您可用以下方法來複製它：

```
cp /usr/lib/X11/system.mwmrc .mwmrc
```

由於 .mwmrc 檔案會置換 system.mwmrc 檔案的全面性的系統效果，您的指定不會干擾其他使用者的指定。

下列範例顯示一般 system.mwmrc 檔案的一部分：

```
#  DEFAULT mwm RESOURCE DESCRIPTION FILE (system.mwmrc)
#
# menu pane descriptions
#
# Root Menu Description
```

```
Menu RootMenu
{ "Root Menu"        f.title
  no-label           f.separator
    "New Window"       f.exec "aixterm &"
    "Shuffle Up"       f.circle_up
    "Shuffle Down"     f.circle_down
    "Refresh"          f.refresh
  no-label           f.separator
    "Restart"          f.restart
    "Quit"             f.quit_mwm
}
```

```
# Default Window Menu Description

Menu DefaultWindowMenu MwmWindowMenu
{ "Restore"   _R   Alt<Key>F5              f.normalize
  "Move"      _M   Alt<Key>F7              f.move
  "Size"      _S   Alt<Key>F8              f.resize
  "Minimize"  _n   Alt<Key>F9              f.minimize
  "Maximize"  _x   Alt<Key>F10             f.maximize
  "Lower"     _L   Alt<Key>F3              f.lower
  no-label                                 f.separator
  "Close"     _C   Alt<Key>F4              f.kill
}
```

```
# no acclerator window menu
Menu NoAccWindowMenu
{
  "Restore"   _R     f.normalize
  "Move"      _M     f.move
  "Size"      _S     f.resize
  "Minimize"  _n     f.minimize
  "Maximize"  _x     f.maximize
  "Lower"     _L     f.lower
  no-label           f.separator
  "Close"     _C     f.kill
}
```

```
Keys DefaultKeyBindings
{
  Shift<Key>Escape             icon|window        f.post_wmenu
  Meta<Key>space               icon|window        f.post_wmenu
  Meta<Key>Tab                 root|icon|window   f.next_key
  Meta Shift<Key>Tab           root|icon|window   f.prev_key
  Meta<Key>Escape              root|icon|window   f.next_key
  Meta Shift<Key>Escape        root|icon|window   f.prev_key
  Meta Ctrl Shift<Key>exclam   root|icon|window   f.set_behavior
}
```

```
#
# button binding descriptions
#
```

```
Buttons DefaultButtonBindings
{
  <Btn1Down>         frame|icon             f.raise
  <Btn3Down>         frame|icon             f.post_wmenu
  <Btn1Down>         root                   f.menu  RootMenu
  <Btn3Down>         root                   f.menu  RootMenu
  Meta<Btn1Down>     icon|window            f.lower
  Meta<Btn2Down>     window|icon            f.resize
  Meta<Btn3Down>     window                 f.move
}
```

```
Buttons PointerButtonBindings
{
```

```
  <Btn1Down>        frame|icon             f.raise
  <Btn2Down>        frame|icon             f.post_wmenu
  <Btn3Down>        frame|icon             f.lower
  <Btn1Down>        root                   f.menu  RootMenu
  Meta<Btn2Down>    window|icon            f.resize
  Meta<Btn3Down>    window|icon            f.move
}
```

```
#
#  END OF mwm RESOURCE DESCRIPTION FILE
#
```

匯出 *shell* 變數（*export shell* 指令）

區域 shell 變數為僅能由建立它的 shell 所辨識的變數。如果您啟動一新的 shell，舊有的 shell 變數將無法辨識它。如果您要讓您所開啟的新 shell 使用舊 shell 的變數，請將變數匯出，使其變成廣域。

您可使用 **export** 指令，使區域變數成為廣域變數。如需使您的區域 shell 變數自動成為廣域，請將該變數匯出至 `.profile` 檔案。

註： 變數可向下匯出至子 shell，但無法向上匯出至母項 shell。

請參閱下列範例：

· 若要將本端 shell 變數 *PATH* 設定為整體變數，請鍵入：

```
export PATH
```

· 若要列出所有已匯出的變數，請鍵入：

```

export
```

系統會顯示如下的資訊：

```
DISPLAY=unix:0
        EDITOR=vi
ENV=$HOME/.env
HISTFILE=/u/denise/.history
HISTSIZE=500
HOME=/u/denise
LANG=En_US
LOGNAME=denise
MAIL=/usr/mail/denise
MAILCHECK=0
MAILMSG=**YOU HAVE NEW MAIL.
USE THE mail COMMAND TO SEE YOUR MAILPATH=/usr/mail/denise?denise has mail !!!
MAILRECORD=/u/denise/.Outmail
PATH=/usr/ucb:/usr/lpp/X11/bin:/bin:/usr/bin:/etc:/u/denise:/u/denise/bin:/u/bin1
PWD=/u/denise
SHELL=/bin/ksh
```

變更預設字型（*chfont* 指令）

若要變更系統啟動時的預設字型，請使用 **chfont** 或 **smit** 指令。字體選用區為一系統用來定義與識別可以使用之字型的檔案。

註： 若要執行 **chfont** 指令，您必須要有 root 授權。

chfont 指令

請參閱下列有關如何使用 **chfont** 指令的範例：

· 若要將現用字型變更為字型選用區中的第五個字型，請鍵入：

```
chfont -a5
```

· 若要將字型變更為斜體、羅馬與粗體字型，請鍵入：

```
chfont -n /usr/lpp/fonts/It114.snf /usr/lpp/fonts/Bld14.snf  /usr/lpp/fonts/Rom14.snf
```

請參閱 *Commands Reference, Volume 1* 中的 **chfont** 指令，以取得完整語法。

smit 指令

chfont 指令也可以使用 **smit** 來執行。

若要選取縣現用字型，請鍵入：

```
smit chfont
```

若要刪除字型選取區，請鍵入：

```
smit chfontpl
```

變更控制鍵（*stty* 指令）

若要變更終端機用於控制鍵的按鍵，請使用 **stty** 指令。

您對控制鍵所做的變更會持續有效，直到您登出為止。若要使您的變更永久有效，請將它們放置在您的 `.profile` 檔案中。

請參閱下列範例：

- 若要指派 Ctrl-Z 作為岔斷鍵，請鍵入：

```
stty intr ^Z
```

務必要在 `intr` 與 `^Z` 之間留有一個空格字元。

- 若要將所有控制鍵重設為預設值，請鍵入：

```
stty sane
```

- 若要顯示您現行的設定，請鍵入：

```
stty -a
```

請參閱 *Commands Reference, Volume 5* 中的 **stty** 指令，以取得完整語法。

變更您的系統提示

您可以變更系統提示。

您的 shell 使用下列提示變數：

項目	說明
PS1	用來作為一般系統提示的提示
PS2	當 shell 預期有其他輸入時，所使用的提示
PS3	當您具有 root 授權時所使用的提示

您可藉由變更其 shell 變數值，變更任何您的提示字元。您的提示變更會持續生效到您登出為止。如需使您的變更永久有效，請將它們放置在您的 `.env` 檔案中。

請參閱下列範例：

- 若要顯示 *PS1* 變數的現行設定，請鍵入：

```
echo "prompt is $PS1"
```

系統會顯示如下的資訊：

```
prompt is $
```

· 若要將提示變更為 Ready>，請鍵入：

```
PS1="Ready> "
```

· 若要將接續提示變更為 Enter more->，請鍵入：

```
PS2="Enter more->"
```

· 若要將 root 提示變更為 Root->，請鍵入：

```
PS3="Root-> "
```

BSD 系統參照

本附錄主要提供給熟悉 4.3 BSD UNIX 或 System V 作業系統的系統管理者使用。這些資訊將說明那些系統和 AIX 之間的異同。

本附錄所討論的主題如下：

BSD 概念

在您開始使用 Berkeley Software Distribution (BSD) 之前，您需要瞭解 BSD 與 AIX 之間的部分差異。

AIX for BSD 系統管理程式簡介

下列是協助 Berkeley Software Distribution (BSD) 系統管理者開始管理 AIX 的要訣。

· 開始時，在圖形主控台以 root 身分登入。

· 從系統主控台執行系統管理，直到您已熟練使用該系統。從系統主控台工作要比從遠端終端機工作來得容易。一旦您已熟練使用該系統，便可以從 xterm 或 ASCII 終端機進行遠端工作。

· 利用系統管理作業的數個 AIX 機能。它們包括：

  – 系統管理介面工具 (SMIT)。SMIT 會在系統管理者與配置及管理指令之間提供一個介面。SMIT 可幫助系統管理員執行大部分系統管理作業。
  – 物件資料管理程式 (ODM)。ODM 會提供從 ODM 資料庫存取物件的常式。ODM 資料庫包含裝置配置資訊
  – 系統資源控制器 (SRC)。SRC 會透過單一介面，提供對常駐程式及其他系統資源的存取及控制。

相關概念

系統資源控制器

「系統資源控制器 (SRC)」可提供指令及子常式集，以便系統管理員及程式設計師能夠更容易地建立及控制子系統。

相關資訊

配置大量裝置

4.3 BSD 與 *AIX* 之間的主要差異

下列是 AIX 與 4.3 BSD 系統之間的主要差異摘要。

在 AIX 上，網路常駐程式會從 /etc/rc.tcpip 檔案（而不是 /etc/rc.local 檔案）啟動。/etc/rc.tcpip shell Script 會從 /etc/inittab 檔案（而不是 /etc/rc 檔案）呼叫。

如果「系統資源控制器 (SRC)」在執行中，則 TCP/IP 常駐程式會在 SRC 控制下執行。如果您不想要 TCP/IP 常駐程式在 SRC 控制下執行，請使用 **smit setbootup_option** 捷徑來將系統變更為 BSD 樣式 **rc** 配置。

AIX 支援下列可在 4.3 BSD 上使用的網路管理功能：

· 核心層次 SYSLOG 記錄機能

· UNIX 網域 Socket 的存取權。

配置資料儲存體

4.3 BSD 通常會在 ASCII 檔案中儲存配置資料。相關資訊會保存在同一行上，且記錄處理程序（排序及搜尋）可在 ASCII 檔案本身進行。記錄的長度可變，且由換行終止。4.3 BSD 會提供若干工具來將部分潛在的大型 ASCII 檔案轉換成資料庫 (dbm) 格式。相關程式庫功能會搜尋 dbm 檔案配對（如果它們存在），但如果找不到 dbm 檔案，則會搜尋原始 ASCII 檔案。

AIX 的部分配置資料會儲存在 ASCII 檔案中，但通常以段落格式儲存。段落是儲存在多行群組中的相關資訊集。每則資訊均有一個標籤，以便讓檔案的內容讓人更容易了解。

AIX 還支援 dbm 版本的密碼及使用者資訊。此外，/etc/passwd、/etc/group 及 /etc/inittab 檔案是 AIX 的檔案範例，其中的資訊以傳統格式（而不是段落格式）儲存。

AIX 的其他配置資料是儲存在「物件資料管理程式 (ODM)」所維護的檔案中。「系統管理介面工具 (SMIT)」可以操作及顯示 ODM 檔案中的資訊。另外，您可以直接使用 ODM 指令來檢視這些檔案。若要查詢 ODM 檔案，請使用下列指令：

- **odmget**
- **odmshow**。

下列 ODM 指令會改變 ODM 檔案：

- **odmadd**
- **odmcreate**
- **odmdrop**
- **odmchange**
- **odmdelete**。

小心：錯誤地改變 ODM 檔案會導致系統失敗，且可能使您無法順利重新啟動系統。只有在作業特定指令（例如 SMIT 產生的那些指令）未順利完成時，才會在 ODM 檔案上直接使用 ODM 指令。

配置管理

當執行 AIX 的系統啟動時，「配置管理程式」會呼叫一組配置專用的指令。這些配置專用的指令稱為方法。方法會識別系統上的裝置並更新 /etc/objrepos 目錄中的適當 ODM 檔案。

/dev 目錄中的裝置特殊檔案不是預先安裝的。部分特殊檔案（如硬碟的檔案）會在啟動配置處理程序期間自動建立。其他特殊檔案（例如 ASCII 終端機的那些檔案）必須由系統管理者使用 SMIT **裝置**功能表來建立。 此資訊會保留在 ODM 中，供系統以後使用。

磁碟管理

在 AIX 中，將磁碟機稱為實體磁區。將分割區稱為邏輯磁區。如在 4.3 BSD 中一樣，單一實體磁區可有多個邏輯磁區。不過，與 4.3 BSD 不同的是，AIX 中的單一邏輯磁區可以跨越多個實體磁區。若要這樣做，您必須將數個實體磁區組成磁區群組，並在該磁區群組上建立邏輯磁區。

AIX 中用於檔案系統及磁區管理的指令包括：

- **crfs**
- **varyonvg**
- **varyoffvg**
- **lsvg**
- **importvg**
- **exportvg**。

下列 4.3 BSD 指令亦可用：

- **mkfs**
- **fsck**
- **fsdb**

- **mount**
- **umount**。

4.3 BSD 的這些指令與 AIX 的這些指令之差異會在 第 303 頁的『BSD 4.3 系統管理程式的檔案系統』中討論。

4.3 BSD 會維護 /etc/fstab 檔案中檔案系統的清單。AIX 會在 /etc/filesystems 檔案中針對每個檔案系統維護一個段落。

tn3270 指令

tn3270 指令是到 **telnet** 指令的鏈結，但它使用 /etc/map3270 檔案及現行 *TERM* 環境變數值，來提供 3270 鍵盤對映。因此，**tn3270** 指令的操作方式與 BSD 版本完全相同。

如果您想要變更 **tn3270**、**telnet** 或 **tn** 指令所用之預設值的 ESC 序列，請在啟動這些指令之前先設定 *TNESC* 環境變數。

新指令

為了處理新配置及磁碟管理系統，AIX 具有約 150 個指令，這些指令對 4.3 BSD 管理者而言是新指令。

啟動

AIX 支援裝置的自動識別及配置。因此，該啟動處理程序與 4.3 BSD 系統大不相同。除核心外，還會將開機檔案系統的映像檔及先前的基本裝置配置資訊載入 RAM 磁碟。在啟動的第一個階段，會載入足夠的配置資訊，並會檢查該配置資訊以允許存取邏輯磁區。會向核心識別分頁空間裝置，且會檢查硬碟的根檔案系統。此時，作業系統會將根檔案系統從 RAM 磁碟變更為硬碟，並完成啟動程序（包括配置其他裝置）。

使用者授權

4.3 BSD 及 SVR4 以前版本的 AT&T UNIX 作業系統，會將所有使用者鑑別資訊（包括已加密密碼）儲存到 /etc/passwd 檔案中。傳統上，/etc/passwd 檔案可由所有使用者讀取。

在 SVR4 系統上，加密的密碼會從 /etc/passwd 檔案移除，並儲存在 /etc/shadow 檔案中。只有擁有 root 權限的使用者及授信程式（如 /bin/login 程式），才可以讀取 /etc/shadow 檔案。

AIX 會將已加密的密碼儲存在 /etc/security/passwd 檔案中。/etc/security 目錄中的其他檔案為 user 及 limits 檔案。這三個檔案會定義允許使用者存取系統的方式（如使用 **rlogin** 或 **telnet** 指令），以及使用者的資源限制（如檔案大小及位址空間）。

列印

可支援大部分 4.3 BSD 列印指令，但會有一些小差異。其中的一個差異是 /etc/qconfig 檔案為 AIX 中的配置檔。

AIX 的行式列印系統可與 4.3 BSD 行式列印系統相互作業（既可將列印工作提交給 4.3 BSD 系統，亦可列印從 4.3 BSD 系統提交的工作）。

shell

AIX 支援 Bourne shell、C shell 及 Korn shell。Bourne shell 程式的完整路徑名稱為 /bin/bsh。/bin/sh 檔案是到 /bin/ksh 檔案的固定鏈結。此檔案可由管理者進行變更。

AIX 不在任何 shell 中支援 shell Script 的 **setuid** 或 **setgid**。

註：

1. AIX 沒有依賴 /bin/sh 的 shell Script。不過，其他系統的許多 shell Script 都依賴於 /bin/sh 是 Bourne shell。
2. 雖然 Bourne shell 與 Korn shell 類似，但 Korn shell 並不是 Bourne shell 的完美超集。

相關參考

BSD 4.3 系統管理程式的系統管理指令
此清單包含專門用來管理 AIX 環境的指令。

***4.3 BSD*、*SVR4* 及 *AIX* 的檔案比較表格**
下列表格比較 4.3 BSD、SVR4 及 AIX 之間的檔名及功能。

表 *61.* 檔案比較表格

4.3 BSD 檔案	**SVR4 檔案**	**AIX 檔案**	**資料庫**	**類型 (odm/ dbm)**
L-Devices	Devices	Devices	否	
L-dialcodes	Dialcodes	Dialcodes	否	
L.cmds	Permissions	Permissions	否	
L.sys	Systems	Systems	否	
USERFILE	Permissions	Permissions	否	
aliases	mail/ namefiles	aliases	aliasesDB/DB	dbm
fstab	vfstab	filesystems	否	
ftpusers	ftpusers	ftpusers	否	
gettytab		N/A		
group	group	group	否	
hosts	hosts	hosts	否	
hosts.equiv	hosts.equiv	hosts.equiv	否	
inetd.conf	inetd.conf	inetd.conf	否	
map3270	N/A	map3270	否	
motd	motd	motd	否	
mtab	mnttab	N/A	否	
named.boot	named.boot	named.boot	否	
named.ca		named.ca	否	
named.hosts		named.data（請參閱附註）	否	
named.local		named.local	否	
named.pid	named.pid	named.pid	否	
named.rev		named.rev	否	
networks	networks	networks	否	
passwd	passwd	passwd	否	
printcap	qconfig	qconfig		
protocols		protocols	否	
remote	remote	remote	否	
resolv.conf	resolv.conf	resolv.conf	否	
sendmail.cf	sendmail.cf	sendmail.cf	sendmail.cfDB	兩者皆非
services		services	否	
shells	shells	N/A		
stab		N/A		

表 61. 檔案比較表格 (繼續)				
4.3 BSD 檔案	**SVR4 檔案**	**AIX 檔案**	**資料庫**	**類型 (odm/ dbm)**
syslog.conf		syslog.conf	否	
syslog.pid		syslog.pid	否	
termcap	terminfo	terminfo		
ttys	ttys	N/A	是	odm
types		N/A		
utmp	utmp	utmp		
vfont		N/A		
vgrindefs		vgrindefs		
wtmp	wtmp	wtmp		

註： 檔名為 named.ca、named.hosts、named.local 及 named.rev 的檔案可在 named.boot 檔案中由使用者定義。不過，在 AIX 的文件中會使用這些名稱作為這些檔案的名稱。

名稱及位址解析

libc 程式庫中的 gethostbyname 及 gethostbyaddr 子常式，會提供對「網域名稱服務」、「網路資訊服務」（NIS，以前稱為「黃頁」），以及 /etc/hosts 資料庫的支援。

如果 /etc/resolv.conf 檔案存在，則一律會先檢查名稱伺服器。如果未解析名稱，且 NIS 在執行中，則會檢查 NIS。如果 NIS 未在執行中，則會檢查 /etc/hosts 檔案。

***BSD 4.3* 系統管理程式的線上文件及 *man* 指令**

AIX 支援 **man-k**、**apropos** 及 **whatis** 指令，但必須先使用 **catman-w** 指令建立這些指令使用的資料庫。

man 指令會在 /usr/man/cat? 檔案中先搜尋一般文字頁。接著，它會在 /usr/man/man? 檔案中搜尋 **nroff** 格式化的頁面。新的 man page 可以以一般文字或 **nroff** 格式來新增。

註：

- 系統不會提供 **man** 指令文字頁。**catman** 指令會從這些文字頁建立資料庫。這些頁面可以是 /usr/man/cat? 檔案中儲存的純文字頁，或 /usr/man/man? 檔案中儲存之 **nroff** 格式化的頁面。
- 您必須安裝「文字格式化」授權程式，**man** 指令才可以使用 **nroff** 指令來讀取 **nroff** 格式化的線上協助頁。

如需這些指令的相關資訊，請參閱 **man**、**apropos**、**whatis** 及 **catman**。

***BSD 4.3* 系統管理程式的 *NFS* 及 *NIS*（以前稱為黃頁）**

下列說明 BSD 4.3 系統管理程式的 NFS 及 NIS。

「網路檔案系統 (NFS)」及「網路資訊服務 (NIS)」常駐程式會從 /etc/rc.nfs 檔案啟動。不過，必須先在 /etc/rc.tcpip 檔案中啟動 **portmap** 常駐程式，才能啟動 NFS 及 NIS 常駐程式。根據預設值，/etc/rc.nfs 檔案不是由 /etc/inittab 檔案呼叫。如果您在 /etc/inittab 檔案中新增一行來呼叫 /etc/rc.nfs Script，則應在 /etc/rc.tcpip Script 之後呼叫之。

如果 NIS 在作用中，請在/etc/passwd 檔案中 +::（加號、冒號、冒號）項目之前加上一個 root 項目，並在 /etc/group 檔案中 +:: 項目之前加上一個系統項目。這可讓系統管理者以 root 身分登入並進行變更（如果系統無法與 NIS 伺服器通訊）。

NFS 可以使用 SMIT 捷徑 **smit nfs** 來進行配置。SMIT 功能表將 NIS（以前稱為「黃頁」）稱為 NIS。在 /etc 及 /usr/etc 目錄中會找到許多 NFS 及 NIS 指令。

部分 NFS 環境使用 **arch** 指令來識別機器系列及機器類型。例如，如果您是使用 IBM RS/6000，請針對系列 (CPU) 指定 power ID，並針對類型（機器）指定 ibm6000 ID。

BSD 4.3 系統管理程式的使用者密碼

當您以 root 使用者的身分使用 AIX 的 **/bin/passwd** 指令時，系統會提示您輸入現行 root 使用者密碼。

使用 **/bin/passwd** 指令的範例如下：

```
# passwd cslater
變更 "cslater" 的密碼
輸入 root 的密碼或
cslater 的舊密碼：
cslater 的新密碼：
重新輸入 cslater 的
新密碼：
#
```

4.3 BSD 版本不會提示您輸入現行 root 使用者密碼。4.3 BSD 版本的範例如下：

```
# passwd cslater
新密碼：
重新鍵入新密碼：
#
```

管理 BSD

有多個 BSD 的指令，您可以用來測量效能、列印及管理您的系統。

BSD 4.3 系統管理程式的統計作業

新增了 4.3 BSD 統計公用程式之後，AIX 之 /usr/lib/sa 目錄中的系統活動報告工具及 /usr/lib/acct 目錄中的統計檔與 AT&T System V 版次 4 (SVR4) 所提供的相同。

許多統計指令都在 /usr/lib/acct 目錄中。若要開始系統統計作業，請使用 /usr/lib/acct/startup 指令。如果統計作業未啟動，諸如 **lastcomm(1)** 的指令將無法傳回資訊。

AIX 會提供下列 4.3 BSD 統計機能：

項目	說明
last(1)	指示使用者及終端機的上次登入
lastcomm(1)	以反向次序顯示最後執行的指令
acct(3)	啟用及停用處理程序統計作業
ac(8)	登入統計作業
accton(8)	啟用或停用系統統計作業
sa(8)	一般維護系統統計檔。

AIX 還會提供下列 System V Interface Definition (SVID) Issue II 統計指令及程式庫功能：

項目	說明
acctcms(1)	從統計記錄產生指令的用法摘要
acctcom(1)	顯示所選處理程序統計記錄摘要
acctcon1(1)	將登入/登出記錄轉換成階段作業記錄
acctcon2(1)	將登入/登出記錄轉換成總統計記錄
acctdisk(1)	從 **diskusg(1)** 指令輸出產生總統計記錄
acctmerg(1)	將總統計檔合併到中間檔案
accton(1)	啟用統計作業
acctprc1(1)	處理 **acct(3)** 指令中的統計資訊

項目	說明
acctprc2(1)	將 **acctprc1(1)** 指令的輸出處理成總統計記錄
acctwtmp(1)	操作連線時間統計記錄
chargefee(1)	向登入名稱收費
ckpacct(1)	檢查 /usr/adm/pacct 檔案的大小
diskusg(1)	產生磁碟統計資訊
dodisk(1)	執行磁碟統計作業
fwtmp(1)	將二進位記錄（wtmp 檔案）轉換成格式化的 ASCII。 **註：** wtmp 檔案位於 /var/adm 目錄中。
lastlogin(1)	更新每個使用者登入的最後日期
monacct(1)	建立每月摘要檔案
prctmp(1)	列印 **acctcon1(1)** 指令產生的階段作業記錄檔案
prdaily(1)	將昨天統計資訊的報告格式化
prtacct(1)	格式化及列印任何總統計檔
runacct(1)	執行每日統計作業
shutacct(1)	由系統關機呼叫來停止統計作業並記載理由
startup(1)	由系統起始設定呼叫來啟動統計作業
turnacct(1)	啟用或停用處理程序統計作業
wtmpfix(1)	使用 wtmp 格式更正檔案中的時間/日期戳記

BSD 4.3 系統管理程式的備份

BSD 4.3 系統管理程式可以備份資料。

tar 及 **cpio** 指令可在系統之間移動資料。AIX 的 **tar** 指令不完全與 4.3 BSD **tar** 指令相容。如果 AIX 的 **tar** 指令是從管線讀取，則它需要 **-B** 旗標（區塊傳輸輸入）。AT&T **cpio** 指令與此版本相容。

AIX 可以 **dump** 及 **restore** 指令格式進行讀取及寫入。例如，具有下面語法的 AIX **backup** 指令：

```
backup -0uf Device Filesystemname
```

與具有下面語法的 4.3 BSD **dump** 指令相同：

```
dump 0uf Device Filesystemname
```

同樣地，具有下面語法的 AIX **restore** 指令：

```
restore -mivf Device
```

與具有下面語法的 4.3 BSD **restore** 指令相同：

```
restore ivf Device
```

AIX 還具有 4.3 BSD **rdump** 及 **rrestore** 指令。兩個版本的唯一差異在於：AIX 的每個引數之前均必須有 **-**（連字號）。例如，下列指令：

```
rdump -0 -f orca:/dev/rmt0 /dev/hd2
```

相當於 4.3 BSD 指令：

```
rdump 0f orca:/dev/rmt0 /dev/hd2
```

具有下列語法的 AIX **backup** 指令：

```
backup -0f /dev/rmt0 /dev/hd2
```

相當於具有此語法的 4.3 BSD **dump** 指令：

```
dump 0f /dev/rmt0 /dev/hd2
```

非 IBM SCSI 磁帶支援

AIX 不直接支援非 IBM SCSI 磁帶機。不過，您可以自行新增使用 IBM SCSI 驅動程式的標頭和介面。

相關概念

系統備份

一旦您的系統在使用中，您的下一考量應是備份檔案系統、目錄及檔案。如果有備份檔案系統，就可以在萬一硬碟損毀時，還原檔案或檔案系統。有不同的方法可以備份資訊。

相關資訊

新增不受支援的裝置至系統

啟動 *BSD 4.3* 系統管理程式

下列討論啟動 BSD 4.3 系統管理程式的 AIX 系統。

在 4.3 BSD 系統上，**init** 程式是啟動程序中的最後一個步驟。 **init** 程式的主要作用是，建立每一個可用終端機埠的處理程序。閱讀 /etc/ttys 檔案可以找到可用的終端機埠。

在 System V 上，**init** 程式會在系統起始設定時啟動。**init** 處理程序會根據 /etc/inittab 檔案中的項目來啟動處理程序。

AIX 會遵循 System V 起始設定程序。您可以使用 **telinit** 指令直接編輯 /etc/inittab 檔案，或藉由使用下列指令來編輯該檔案：

項目	說明
chitab(1)	變更 /etc/inittab 檔案中的記錄
lsitab(1)	列出 /etc/inittab 檔案中的記錄
mkitab(1)	在 /etc/inittab 檔案中製作記錄
rmitab(1)	移除 /etc/inittab 檔案中的記錄

對 /etc/inittab 檔案所作的變更，會在下次系統重新開機或執行 **telinit q** 指令時生效。

尋找並檢查 *BSD 4.3* 系統管理程式的檔案

下列是 AIX 支援的 BSD 檔案指令的清單。

AIX 支援下列 4.3 BSD 檔案指令：

- **which**
- **whereis**
- **what**
- **file**。

AIX 不支援 **find** 指令的 4.3 BSD **fast find** 語法。目前尚沒有置換功能。下面的 **ffind** shell Script 可用來模擬功能：

```
#!/bin/bsh
PATH=/bin
for dir in /bin /etc /lib /usr
```

```
            do
find $dir -print | egrep $1
done
```

ffind Script 的語法為：

```
ffind Filename
```

BSD 4.3 系統管理程式的分頁空間

下列指令會協助您管理分頁空間（亦稱交換空間）。

項目	說明
chps(1)	變更分頁空間屬性
lsps(1)	列出分頁空間的屬性
mkps(1)	將其他分頁空間新增到系統
rmps(1)	從系統移除分頁空間
swapoff(1)	取消啟動一或多個分頁空間
swapon(1)	指定用於分頁及交換的其他裝置

如果需要大的分頁空間，請針對每個硬碟放置一個分頁邏輯磁區。如此可允許跨多個磁碟機排程分頁。

變更預設啟動以允許 *4.3 BSD ASCII* 配置

您可以透過 SMIT 及 ODM 檔案，或透過 4.3 BSD ASCII 配置檔，來管理 AIX 的網路介面。

若要透過 4.3 BSD ASCII 配置檔管理網路介面，請取消標題下 /etc/rc.net 檔案中指令的註解：

```
# Part II - Traditional
Configuration
```

然後，如果您希望有純文字檔配置及 SRC 支援，請編輯 /etc/rc.net 檔案，並利用適當的參數解除註銷 **hostname**、**ifconfig** 及 **route** 指令。

如果您不希望有 SRC 支援的純文字檔配置，請使用 **smit setbootup_option** 捷徑，將系統變更為 BSD 樣式的 **rc** 配置。此選項會配置系統在啟動時使用 /etc/rc.bsdnet 檔案。您還必須編輯 /etc/rc.bsdnet 檔案，並使用適當的參數取消註銷 **hostname**、**ifconfig** 及 **route** 指令。

ifconfig 及 *netstat* 指令的其他選項

下列是 **ifconfig** 及 **netstat** 指令的其他選項清單。

AIX 的 **ifconfig** 指令具有下列其他選項：

mtu
mtu 變數指定在本端網路（及本端子網路）上使用的最大傳輸單位 (MTU)， 及用於遠端網路的 MTU。若要將與乙太網路及其他網路的相容性增至最大，請將記號環及乙太網路預設 *mtu* 值均設為 1500。

allcast
allcast 旗標會設定記號環播送策略。設定 **allcast** 旗標會將透過記號環橋接器的連通性增至最大。清除 **allcast** 旗標（藉由指定 -allcast）會將環上的多餘資料流量減至最小。

AIX 的 **netstat** 指令具有 **-v** 旗標。**netstat -v** 指令會列印驅動程式統計資料，如傳輸位元組計數、傳輸錯誤計數、接收位元組計數及接收錯誤計數。如需 **ifconfig** 及 **netstat** 指令的相關資訊，請參閱 **ifconfig** 及 **netstat**。

其他的網路管理指令

AIX 支援下列其他指令。

項目	說明
securetcpip	**securetcpip** shell Script 會啟用控制存取模式（提供增強的網路安全）。它不允許執行數個不安全的 TCP/IP 程式，如 `tftp`、`rcp`、`rlogin` 及 `rsh` 程式。它還會限制 `.netrc` 檔案的使用。
gated	**gated** 指令提供對 SNMP 的 MIB 支援。
no	**no** 指令設定網路選項，包括： **dogticks** 針對 `ifwatchdog` 常式設定計時器粗細度 **subnetsarelocal** 判斷封包位址是否在本端網路上 **ipsendredirects** 指定核心是否應傳送重新導向信號 **ipforwarding** 指定核心是否應轉遞封包 **tcp_ttl** 指定「傳輸控制通訊協定 (TCP)」封包的存活時間 **udp_ttl** 指定「使用者資料封包通訊協定 (UDP)」封包的存活時間 **maxttl** 指定「路由資訊通訊協定 (RIP)」封包的存活時間 **ipfragttl** 指定「網際網路通訊協定 (IP)」片段的存活時間 **lowclust** 針對叢集 **mbuf** 儲存區指定低臨界值 **lowmbuf** 對 **mbuf** 儲存區指定低臨界值 **thewall** 指定配置給 `mbuf` 及叢集 `mbuf` 儲存區之記憶體最大數量 **arpt_killc** 指定在刪除非作用中完整「位址解析通訊協定 (ARP)」項目之前的時間（以分鐘為單位）
iptrace	**iptrace** 指令提供對網際網路通訊協定之介面層次封包追蹤。

ipreport	**ipreport** 指令將追蹤格式化為人們可讀的格式。使用此指令的範例如下：
	`iptrace -i en0 /tmp/iptrace.log`
	`# kill iptrace daemon`
	`kill `ps ax | grep iptrace | awk '{ print $1 }'``
	`ipreport /tmp/iptrace.log | more`

匯入 *BSD 4.3* 密碼檔

您可以將 BSD 4.3 密碼檔匯入 AIX。

若要匯入 BSD 4.3 密碼檔，請執行下列步驟：

1. 將 BSD 4.3 密碼檔複製至 `/etc/passwd` 檔案，然後輸入：

   ```
   pwdck -y ALL
   ```

2. 針對任何新使用者，使用一個空值段落來更新 `/etc/security/limits` 檔案。

   usrck 指令會執行此作業，但使用 **usrck** 指令可能會導致問題，除非將 `/etc/group` 檔案與 `/etc/passwd` 檔案一起匯入。如需 **usrck** 指令的相關資訊，請參閱 **usrck**。

小心：如果修改了 /etc/security/limits 檔案，則堆疊不得超過 65,536 個位元組。如果超過，則執行 **usrck** 指令可能會導致問題。將堆疊大小變更為 65,536 並重新執行 **usrck** 指令。

3. 執行 **grpck** 及 **usrck** 指令，來驗證群組及使用者屬性。

編輯 *BSD 4.3* 系統管理程式的密碼檔

以下說明如何變更密碼檔中的項目，以及如何以 BSD 4.3 方式來管理 AIX 上的密碼。

在 AIX 中，有提供 **lsuser**、**mkuser**、**chuser** 及 **rmuser** 等指令來管理密碼。所有這些指令都可以藉由執行 SMIT 來使用。不過，所有這些指令均一次僅處理一個使用者。

如需這些指令的相關資訊，請參閱 **lsuser**、**mkuser**、**chuser** 及 **rmuser**。

註：使用編輯器一次變更多個使用者名稱項目需要同時編輯多個檔案，原因是密碼是儲存在 /etc/security/passwd 檔案中、授權資訊儲存在 /etc/security/user 檔案中，而其餘的使用者資料則儲存在 /etc/passwd 檔案中。

AIX 不支援 **vipw** 指令，但支援 **mkpasswd** 指令。不過，您仍可以利用 4.3 BSD 方式來管理 AIX 中的密碼。請使用下列程序：

1. 將 BSD 4.3 密碼檔放入 /etc/shadow 檔案。
2. 變更對該檔案的許可權，方法是輸入下列指令：

```
chmod 000 /etc/shadow
```

3. 將下列 **vipw** shell Script 放入 /etc 目錄：

```
------------------------------------------------------
----
#!/bin/bsh
#
# vipw. Uses pwdck for now. May use usrck someday
#
PATH=/bin:/usr/bin:/etc:/usr/ucb # Add to this if your editor is
                                                                  # some place else
if [ -f /etc/ptmp ] ; then
                                echo "/etc/ptmp exists. Is someone else using vipw?"
                        exit 1
fi
if [ ! -f /`which "$EDITOR" | awk '{ print $1 }'` ] ; then
                        EDITOR=vi
fi
cp /etc/shadow /etc/ptmp
if (cmp /etc/shadow /etc/ptmp) ; then
                $EDITOR /etc/ptmp
                else
                echo cannot copy shadow to ptmp
                exit 1
fi
if (egrep "^root:" /etc/ptmp >/dev/null) ; then
                cp /etc/ptmp /etc/shadow ; cp /etc/ptmp /etc/passwd
                chmod 000 /etc/passwd /etc/shadow
                pwdck -y ALL 2>1 >/dev/null # return code 114 may change
                rc=$?
                if [ $rc -eq 114 ]; then
                                chmod 644 /etc/passwd
                                                rm -f /etc/passwd.dir /etc/passwd.pag
                mkpasswd /etc/passwd
                                                # update /etc/security/limits, or ftp
                                                # will fail
                                else
                                                  pwdck -y ALL
                fi
                else
                echo bad entry for root in ptmp
fi
rm /etc/ptmp
------------------------------------------------------------
```

4. 如果您使用 **vipw** Shell Script 或 **mkpasswd** 指令，請注意 SMIT 和 **mkuser**、**chuser** 及 **rmuser** 指令不使用 **mkpasswd** 指令。您必須執行：

```
mkpasswd /etc/passwd
```

   來更新 /etc/passwd.dir 及 /etc/passwd.pag 檔案。

   小心：起始設定 *IFS* 變數及 trap 陳述式，會預防使用部分常用方法來利用 **setuid** 特性中固有的安全漏洞。不過，**vipw** 及 **passwd** shell Script 是要針對相對開放的環境，在這樣的環境中相容性是重要的考量事項。如果您想要更為安全的環境，請僅使用 AIX 的標準指令。

5. 將下列 **passwd** shell Script 放入 /usr/ucb 目錄：

```
-------------------------------------------------------
#!/bin/ksh
#
# matches changes to /etc/security/passwd file with changes to
#/etc/shadow
#
IFS=" "
PATH=/bin
trap "exit 2" 1 2 3 4 5 6 7 8 10 12 13 14 15 16 17 18 21 22 \
                23 24 25 27 28 29 30 31 32 33 34 35 36 60 61 62
if [ -n "$1" ]; then
                USERNAME=$1
                else
                USERNAME=$LOGNAME
fi
if [ -f /etc/ptmp ]; then
                echo password file busy
                exit 1
fi
                trap "rm /etc/ptmp; exit 3" 1 2 3 4 5 6 7 8 10 12 13 \
                                14 15 16 17 18 21 22 23 24 25 27 28 29 30 31 \
                                32 33 34 35 36 60 61 62
if (cp /etc/security/passwd /etc/ptmp) ; then
                chmod 000 /etc/ptmp else
                rm -f /etc/ptmp exit 1
fi
if ( /bin/passwd $USERNAME ) ; then
                PW=` awk ' BEGIN { RS = "" }
                                $1 == user { print $4 } ' user="$USERNAME:" \
/etc/security/passwd `
                else
        rm -f /etc/ptmp
                exit 1
fi
rm -f /etc/ptmp
awk -F: '$1 == user { print $1":"pw":"$3 ":"$4":"$5":"$6":"$7 }
                $1 != user { print $0 }' user="$USERNAME" pw="$PW" \
                                /etc/shadow > /etc/ptmp
chmod 000 /etc/ptmp
mv -f /etc/ptmp /etc/shadow
-------------------------------------------------------
```

6. 變更對 **passwd** Script 的許可權，方法是輸入下列指令：

```
chmod 4711 /usr/ucb/passwd
```

7. 請確定每個使用者 *PATH* 環境變數均指定在 /bin 目錄之前先搜尋 /usr/ucb 目錄。

BSD 4.3 系統管理程式的效能測量與調整

下列討論 AIX 裝置屬性以及效能測量與調整。

AIX 上的所有裝置均具有與其相關聯的屬性。若要檢視裝置屬性，請輸入：

```
lsattr -E -l devicename
```

所有具有 True 值的屬性，均可以使用下面的指令進行修改：

```
chdev -l devicename -a attr=value
```

小心：若錯誤地變更裝置參數可能會損壞您的系統。

根據預設值，每個使用者的最大處理程序數目為 40。對於同時開啟許多視窗的使用者而言，該預設值可能太低。可以使用下列指令在整個系統變更該值：

```
hdev -l sys0 -a maxuproc=100
```

此範例將最大數目變更為 100。一旦系統重新啟動，即會設定新值。

若要檢視此屬性及其他系統屬性的現行設定，請鍵入：

```
lsattr -E -l sys0
```

mbuf 服務目前不支援 maxmbuf 屬性。

AIX 支援 **vmstat** 及 **iostat** 指令，但不支援 **systat** 指令或平均資料流量。如需這些指令的相關資訊，請參閱 **vmstat** 及 **iostat**。

BSD 4.3 系統管理程式的印表機

AIX 作業系統可支援兩個列印子系統：4.3 BSD 及 System V。

System V 樣式的列印子系統會使用 System V 版次 4 的指令、佇列及檔案，且以相同的方式進行管理。下列段落會說明，若要管理 4.3 BSD 樣式的列印子系統，您需要瞭解什麼。您可透過 SMIT 控制讓哪個子系統處於作用中。一次僅可有一個子系統處於作用中。

列印是由 /usr/lpd 目錄中的程式及配置進行管理。AIX 之印表機子系統與 4.3 BSD 各有不同的設計、配置、佇列機制及常駐程式處理程序。不過，它們都使用 lpd 通訊協定進行遠端列印。兩個系統均使用 /etc/hosts.lpd（如果它存在），否則會使用 /etc/host.equiv。AIX 的印表機子系統會提供到 4.3 BSD 印表機子系統的閘道，好讓使用 AIX 的系統可以將列印工作提交給 4.3 BSD 系統，並接受 4.3 BSD 系統所提交的列印工作。

4.3 BSD 的 /etc/printcap 檔案不存在 AIX 中。此檔案是排存器配置及印表機功能資料庫的組合。 使用者需要瞭解 printcap 檔案的格式及關鍵字，才能正確地設定印表機。

AIX 的 /etc/qconfig 檔案僅包含排存器配置資訊。印表機功能定義於 ODM 預先定義/自訂的資料庫中。您可以使用 **mkvirprt**（製作虛擬印表機）指令，來向系統定義特定印表機的功能。

若要讓印表機 lp0 可用來在遠端主機 viking 上進行列印，請將下列內容放入 4.3 BSD 系統 /etc/printcap 檔案：

```
lp0|Print on remote printer attached to
viking:Z
:lp=:rm=viking:rp=lp:st=/usr/spool/lp0d
```

若要在 AIX 中執行同樣的動作，請在 /etc/qconfig 檔案中加入下列內容：

```
lp0:
        device = dlp0
        host = viking
        rq = lp
dlp0:
        backend = /usr/lib/lpd/rembak
```

AIX 支援下列印表機指令及程式庫功能：

項目	說明
cancel(1)	取消對列表機的要求
chquedev(1)	變更印表機或繪圖機佇列裝置名稱

項目	說明
chvirprt(1)	變更虛擬印表機的屬性值
disable(1)	停用印表機佇列
enable(1)	啟用印表機佇列
hplj(1)	針對帶有 K 卡匣的 HP LaserJetII，後續處理 **troff** 輸出
ibm3812(1)	後續處理 IBM 3812 Mod 2 頁式印表機的 **troff** 輸出
ibm3816(1)	後續處理 IBM 3816 頁式印表機的 **troff** 輸出
ibm5587G(1)	後續處理 IBM 5587G（含 32x32/24x24 卡匣）的 **troff** 輸出
lp(1)	將要求傳送到列表機
lpr(1)	將列印工作移入佇列
lprm(1)	將工作從列表機排存佇列移除
lpstat(1)	顯示列表機狀態資訊
lptest(1)	產生列表機波紋型樣
lsallqdev(1)	列出佇列中的所有已配置印表機佇列裝置名稱
lsvirprt(1)	顯示虛擬印表機的屬性值
mkque(1)	將印表機佇列新增到系統
mkquedev(1)	將印表機佇列裝置新增到系統
mkvirprt(1)	製作虛擬印表機
pac(1)	準備印表機/繪圖機統計記錄
piobe(1)	印表機後端的「列印工作管理程式」
pioburst(1)	產生印表機輸出的分割頁面（標頭及標尾頁面）
piocmdout(3)	針對印表機格式製作器輸出屬性字串的子常式
piodigest(1)	摘要虛擬印表機定義及儲存處的屬性值
pioexit(3)	從印表機格式製作器跳出的子常式
pioformat(1)	驅動印表機格式製作器
piofquote(1)	轉換針對 PostScript 印表機指定的特定控制字元
piogetstr(3)	針對印表機格式製作器擷取屬性字串的子常式
piogetvals(3)	針對印表機格式製作器起始設定「印表機屬性」資料庫變數的子常式
piomsgout(3)	從印表機格式製作器傳送訊息的子常式
pioout(1)	印表機後端的裝置驅動程式介面程式
piopredef(1)	建立預先定義的印表機資料串流定義
proff(1)	使用個人印表機資料串流，格式化印表機的文字
qadm(1)	執行印表機排存系統的系統管理
qconfig(4)	配置印表機佇列系統
qstatus(1)	提供印表機排存系統的印表機狀態
restore(3)	將印表機還原到其預設狀態
rmque(1)	將印表機佇列從系統移除
rmquedev(1)	將印表機或繪圖機佇列裝置從系統移除

項目	說明
rmvirprt(1)	移除虛擬印表機
splp(1)	變更或顯示印表機驅動程式設定
xpr(1)	格式化視窗傾出檔，以輸出到印表機

相關資訊
系統管理的印表機概觀

BSD 4.3 系統管理程式的系統管理指令
此清單包含專門用來管理 AIX 環境的指令。

項目	說明
bosboot(1)	起始設定開機裝置。
bootlist(1)	改變系統可用之開機裝置的清單（或清單中這些裝置的排序）。
cfgmgr(1)	藉由執行 /etc/methods 目錄中的程式來配置裝置。
chcons(1)	將系統主控台重新導向到裝置或檔案（下次啟動時有效）
chdev(1)	變更裝置性質
chdisp(1)	變更低功率終端機 (LFT) 子系統使用的顯示器。
checkcw(1)	為 **troff** 指令準備固定寬度文字。
checkeq(1)	檢查使用備忘錄巨集格式化的文件。
checkmm(1)	檢查使用備忘錄巨集格式化的文件。
checknr(1)	檢查 nroff 及 troff 檔案。
chfont(1)	在開機時變更選取的預設字型。
chfs(1)	變更檔案系統的屬性。
chgroup(1)	變更群組的屬性。
chgrpmem(1)	變更群組的管理者或成員。
chhwkbd(1)	變更在「物件資料管理程式 (ODM)」資料庫中儲存的低功率終端機 (LFT) 鍵盤屬性。
chitab(1)	變更 /etc/inittab 檔案中的記錄。
chkbd(1)	在系統啟動時變更低功率終端機 (LFT) 使用的預設鍵盤對映。
chkey(1)	變更加密金鑰。
chlang	在 /etc/environment 檔案中設定適用於下次登入的 *LANG* 環境變數。
chlicense(1)	使用者授權有兩種類型：固定的及浮動的。固定的授權一律是啟用的，且授權數可透過 **-u** 旗標進行變更。浮動授權可透過 **-f** 旗標來啟用或停用（開啟或關閉）
chlv(1)	變更邏輯磁區的性質
chnamsv(1)	變更主機上的 TCP/IP 型名稱服務配置
chprtsv(1)	變更用戶端或伺服器機器上的列印服務配置
chps(1)	變更分頁空間的屬性。
chpv(1)	變更磁區群組中實體磁區的性質。
chque(1)	變更佇列名稱。
chquedev(1)	變更印表機或繪圖機佇列裝置名稱。

項目	說明
chssys(1)	變更子系統物件類別中的子系統定義。
chtcb(1)	變更或查詢檔案的授信計算庫屬性。
chtz	變更系統時區資訊。
chuser(1)	變更指定使用者的屬性。
chvfs(1)	變更 /etc/vfs 檔案中的項目。
chvg(1)	設定磁區群組的性質。
chvirprt(1)	變更虛擬印表機的屬性值。
crfs(1)	新增檔案系統。
crvfs(1)	於 /etc/vfs 檔案中建立項目。
exportvg(1)	從實體磁區集匯出磁區群組的定義。
extendvg(1)	將實體磁區新增到磁區群組。
grpck(1)	驗證群組定義的正確性。
importvg(1)	從實體磁區集匯入新的磁區群組定義。
lsallq(1)	列出所有已配置佇列的名稱。
lsallqdev(1)	列出指定佇列中的所有已配置印表機及繪圖機佇列裝置的名稱。
lsdisp(1)	列出系統上目前可用的顯示器。
lsfont(1)	列出顯示器可用的字型。
lsfs(1)	顯示檔案系統的性質。
lsgroup(1)	顯示群組的屬性。
lsitab(1)	列出 /etc/inittab 檔案中的記錄。
lskbd(1)	列出低功率終端機 (LFT) 子系統目前可用的鍵盤對映。
lslicense(1)	顯示固定授權的數目及浮動授權的狀態。
lslpp(1)	列出選用程式產品。
lsnamsv(1)	顯示資料庫中儲存的名稱服務資訊。
lsprtsv(1)	顯示資料庫中儲存的列印服務資訊。
lsps	列出分頁空間及屬性。
lsque(1)	顯示佇列段落名稱。
lsquedev(1)	顯示裝置段落名稱。
lssrc(1)	取得子系統、子系統群組或子伺服器的狀態。
lsuser(1)	顯示使用者帳戶的屬性。
lsvfs(1)	列出 /etc/vfs 檔案中的項目。
mkcatdefs(1)	前置處理訊息原始檔。
runcat(1)	將 **mkcatdefs** 指令的輸出資料以管線輸送到 gencat 指令。
mkdev(1)	將裝置新增到系統。
mkfont(1)	將與顯示器相關的字型碼新增到系統。
mkfontdir(1)	從字型檔的目錄建立 fonts.dir 檔案。
mkgroup(1)	建立新群組。

項目	說明
mkitab(1)	在 /etc/inittab 檔案中建立記錄。
mklv(1)	建立邏輯磁區。
mklvcopy(1)	將副本新增到邏輯磁區。
mknamsv(1)	在主機上為用戶端配置 TCP/IP 型名稱服務。
mknotify(1)	將通知方法定義新增到通知物件類別。
mkprtsv(1)	在主機上配置 TCP/IP 型列印服務。
mkps(1)	將其他分頁空間新增到系統。
mkque(1)	將印表機佇列新增到系統。
mkquedev(1)	將印表機佇列裝置新增到系統。
mkserver(1)	將子伺服器定義新增到子伺服器物件類別。
mkssys(1)	將子系統定義新增到子系統物件類別。
mksysb	為後續的重新安裝備份 rootvg 磁區群組中的裝載檔案系統。
mkszfile	為重新安裝記錄 rootvg 磁區群組中的裝載檔案系統大小。
mktcpip(1)	設定在主機上啟動 TCP/IP 的必要值。
mkuser(1)	建立新使用者帳戶。
mkuser.sys(1)	自訂新使用者帳戶。
mkvg(1)	建立磁區群組。
mkvirprt(1)	建立虛擬印表機。
odmadd(1)	將物件新增至已建立的物件類別中。
odmchange(1)	變更指定物件類別中已選取物件的內容。
odmcreate(1)	產生 ODM 應用程式開發必要的 .c（來源）及 .h（併入）檔案並建立空的物件類別。
odmdelete(1)	從指定的物件類別中刪除已選取的物件。
odmdrop(1)	移除物件類別。
odmget(1)	將所指定物件類別中的物件擷取出來，並置於 odmadd 輸入檔中。
odmshow(1)	在畫面上顯示物件類別定義。
pwdck(1)	確認本端鑑別資訊的正確性。
redefinevg	重新定義裝置配置資料庫中給定磁區群組的實體磁區設定。
reducevg(1)	從磁區群組移除實體磁區。如果從磁區群組移除所有實體磁區，則亦會刪除該磁區群組。
reorgvg(1)	重組磁區群組的實體分割區配置。
restbase(1)	從開機映像檔還原自訂的資訊。
rmdel(1)	將差異處從「原始程式控制系統 (SCCS)」檔案中移除。
rmdev(1)	從系統移除裝置。
rmf(1)	移除資料夾及其所包含的訊息。
rmfs(1)	移除檔案系統。
rmgroup(1)	移除群組。
rmitab(1)	移除 /etc/inittab 檔案中的記錄。

項目	說明
rmlv(1)	從磁區群組移除邏輯磁區。
rmlvcopy(1)	從邏輯磁區移除副本。
rmm(1)	移除訊息。
rmnamsv(1)	取消配置主機上的 TCP/IP 型名稱服務。
rmnotify(1)	從通知物件類別移除通知方法定義。
rmprtsv(1)	取消配置用戶端或伺服器機器上的列印服務。
rmps(1)	從系統移除分頁空間。
rmque(1)	從系統移除印表機佇列。
rmquedev(1)	從系統移除印表機或繪圖機佇列裝置。
rmserver(1)	從子伺服器物件類別中移除子伺服器定義。
rmssys(1)	從子系統物件類別中移除子系統定義。
rmuser(1)	移除使用者帳戶。
rmvfs(1)	移除 /etc/vfs 檔案中的項目。
rmvirprt(1)	移除虛擬印表機。
savebase(1)	將 ODM 中的基本自訂裝置資料儲存到開機裝置上。
swapoff(1)	取消啟動一或多個分頁空間。
swapon(1)	指定用於分頁及交換的其他裝置。
syncvg(1)	同步化非現行的邏輯磁區副本。
usrck(1)	驗證使用者定義的正確性。
varyoffvg(1)	取消啟動磁區群組。
varyonvg(1)	啟動磁區群組。

相關概念

4.3 BSD 與 AIX 之間的主要差異

下列是 AIX 與 4.3 BSD 系統之間的主要差異摘要。

BSD 4.3 系統管理程式的 Cron

此作業系統的 **cron** 常駐程式與 System V 版次 2 **cron** 常駐程式類似。

/etc/inittab 檔案中的項目會啟動 **cron** 常駐程式。

BSD 4.3 系統管理程式的裝置

以下討論 BSD 4.3 系統管理程式的裝置。

僅當在下列情況下，應用程式才可存取 4.3 BSD 系統上的裝置：

- 裝置的實體已安裝完畢且運作正常。
- 裝置的驅動程式位於核心。
- 裝置的裝置特殊檔案存在於 /dev 目錄。

僅當在下列情況下，應用程式才可存取 AIX 上的裝置：

- 裝置的實體已安裝完畢且運作正常。
- 裝置的驅動程式位於核心或位於已載入的核心擴充。
- 裝置的裝置特殊檔案存在於 /dev 目錄。
- /etc/objrepos 目錄中之物件資料庫包含的裝置項目符合實體配置。

在 /etc/methods 目錄中找到之稱為方法的裝置特有程式會維護物件資料庫。方法是由「配置管理程式」（透過 **cfgmgr** 指令來存取）及其他指令呼叫。

如果應用程式無法再存取某個裝置，則可能表示硬體有錯誤，或可能表示 /etc/objrepos 目錄中的配置資料庫已經損壞。

cfgmgr 指令會處理 **/etc/objrepos** 目錄中的配置資料庫，並且在啟動時，會由 **cfgmgr** 指令（配置管理程式）進行處理。

下面的虛擬程式碼顯示「配置管理程式」邏輯：

```
/* Main */
While there are rules in the Config_Rules database
        {
                Get the next rule and execute it
                Capture stdout from the last execution
                                Parse_Output(stdout)
        }
/* Parse Output Routine */
/* stdout will contain a list of devices found */
Parse_OutPut(stdout)
        {
                While there are devices left in the list
                {
                                Lookup the device in the database
                                if (!defined)
                                                Get define method from database and execute
                if (! configured)
                        {
                                                Get config method from database and execute
                                                Parse_Output(stdout)
                         }
                }
        }
```

BSD 4.3 系統管理程式的 UUCP

下表列出 UUCP 指令和檔案。

項目	說明
Dialers(4)	列出用於 BNU 遠端通訊鏈結的數據機
Maxuuxqts(4)	限制可以執行之 BNU **uuxqt** 常駐程式的案例數
Permissions(4)	針對遠端系統指定 BNU 指令許可權
Poll(4)	指定 BNU 程式應輪詢遠端系統的時間
Systems(4)	列出本端系統可以與其通訊的遠端電腦
rmail(1)	處理透過 BNU 接收的遠端郵件
uucheck(1)	檢查 BNU 所需的檔案及目錄
uuclean(1)	將檔案從 BNU 排存目錄移除
uucleanup(1)	將所選檔案從 BNU 排存目錄刪除
uucpadm(1)	輸入基本 BNU 配置資訊
uudemon.admin(1)	提供有關 BNU 檔案轉送狀態的週期性資訊
uudemon.cleanu(1)	清除 BNU 排存目錄及日誌檔
uudemon.hour(1)	使用 BNU 程式起始對遠端系統的檔案傳輸呼叫
uudemon.poll(1)	輪詢 BNU「輪詢」檔案中列出的系統
uulog(1)	提供系統上 BNU 檔案轉送活動的相關資訊
uupoll(1)	強制輪詢遠端 BNU 系統
uuq(1)	顯示 BNU 工作佇列，並從該佇列刪除指定的工作

項目	說明
uusnap(1)	顯示與遠端系統之 BNU 聯絡的狀態
uustat(1)	報告 BNU 作業的狀態，並提供對 BNU 作業的限制控制

AIX 還會提供 4.3 BSD **uuencode** 及 **uudecode** 指令。不支援 HDB **uugetty** 指令。如需這些指令的相關資訊，請參閱 **uuencode** 及 **uudecode**。

相關資訊
BNU 檔案及目錄結構

BSD 4.3 系統管理程式的檔案系統
使用類似的指令來裝載及卸載檔案系統。

AIX 使用 /etc/filesystem 檔案來列出檔案系統裝置資訊，且使用類似的指令來裝載及卸載檔案系統。

***/etc/filesystems* 檔案及 */etc/fstab* 檔案**
4.3 BSD 系統將區塊裝置及裝載點的清單儲存在 /etc/fstab 檔案中。SVR4 系統將區塊裝置及裝載點資訊儲存在 /etc/vfstab 檔案中。AIX 則將區塊裝置及裝載點資訊儲存在 /etc/filesystems 檔案中。

crfs、**chfs** 及 **rmfs** 指令會更新 /etc/filesystems 檔案。

4.3 BSD 系統管理者可能會對 /etc/filesystems 檔案中的 *check* 變數感興趣。*check* 變數可設為值 TRUE、FALSE，或設為數字。例如，您可以 在 /etc/filesystems 檔案中指定 check=2。此數字會指定將通過檢查此檔案系統之 **fsck** 指令。*check* 參數對應於 /etc/fstab 檔案記錄中的第五個欄位。

在 /etc/filesystems 檔案中沒有傾出頻率參數。

***AIX* 上的檔案系統支援**
AIX 支援各種檔案系統。

AIX 支援磁碟限額。

AIX 不允許將磁片裝載為檔案系統。

AIX 之 **mount** 及 **umount** 指令的語法，與這些指令的 SVR4 版本及 4.3 BSD 版本不同。下表會顯示所有三個系統之同時裝載及卸載所有檔案系統的指令：

mount 及 **unmount** 指令			
功能	**此作業系統的語法**	**4.3 BSD 語法**	**SVR4 語法**
裝載所有檔案系統	**mount all**	**mount -a**	**mountall**
卸載所有檔案系統	**umount all**	**umount -a**	**umountall**

請參閱 ../devicemanagement/file_sys.dita#file_sys，以取得進一步資訊。

BSD 4.3 系統管理程式的終端機
下列討論 BSD 4.3 系統管理程式的終端機。

傳統上，4.3 BSD 系統管理程式會藉由修改 /etc/ttys 檔案，並將 HUP 信號傳送到 **init** 程式，來啟用或停用終端機埠。

AIX 會將終端機埠資訊儲存在 ODM 中，並在 **init** 程式讀取 /etc/inittab 檔案時，啟動終端機。在 AIX 中，則是使用 SMIT 介面來配置終端機埠。

埠與 /dev 目錄中的裝置特殊檔案名稱之間沒有固定的對映。因此，不熟悉 AIX 的系統管理員會搞不清要配置哪個埠。當使用 SMIT 時，第一個介面板序列埠（實際上標記為 **s1**) 在 SMIT 功能表中是指位置 **00-00-S1**、配接卡 **sa0**，以及埠 **s1**。第二個介面板序列埠（實際上標記為 **s2**）指的是位置 **00-00-S2**、配接卡 **sa1**，以及埠 **s2**。

使用 **penable** 及 **pdisable** 指令可以啟用及停用埠。

termcap* 及 *terminfo
像 System V 一樣，此作業系統會使用 /usr/lib/terminfo/?/* 檔案中的 terminfo 項目。

「4.3 BSD 系統」的使用者可能會發現下列指令很有幫助：

captoinfo(1)
將 termcap 檔案轉換為 terminfo 檔案

tic(1)
將 terminfo 檔案從來源轉換為編譯格式。

此作業系統包含許多 terminfo 項目的來源。其中部分可能需要使用 **tic** 指令進行編譯。 在 /lib/libtermcap/termcap.src 檔案中提供了 termcap 檔案。

輸入及輸出重新導向

AIX 作業系統可讓您使用特定的 I/O 指令與符號，操作您的系統之輸入和輸出 (I/O) 資料。

您可藉由指定彙集資料的位置，以資料控制輸入。例如，您可以指定在鍵盤上輸入資料（標準輸入）時，讀取輸入資料，或是從檔案來讀取輸入資料。您可藉由指定於何處顯示或儲存資料，以控制資料的輸出。您可指定寫入輸出資料至畫面上（標準輸出），或寫入至檔案中。

因為 AIX 是多工作業系統，因此它設計為可以處理互相合併的程序。

相關概念
顯示檔案內容的指令（pg、more、page 及 cat 指令）
這些 **pg**、**more** 和 **page** 指令可讓您檢視某個檔案內容，並且控制顯示檔案內容的速度。

Korn shell 或 POSIX shell 中的輸入及輸出重新導向
在 Korn shell 執行指令前，它會先掃描指令行尋找重新導向字元。這些特殊的表示法會將 shell 導入重新導向輸入及輸出。

標準輸入、標準輸出及標準錯誤檔案

當指令開始執行時，通常會預期下列檔案都已開啟：標準輸入、標準輸出和標準錯誤（有時稱為錯誤輸出 或診斷 輸出）。

一個稱為檔案描述子的數字與這些檔案相關，如下所示：

項目	說明
檔案描述子 0	標準輸入
檔案描述子 1	標準輸出
檔案描述子 2	標準錯誤（診斷）輸出

子處理程序通常從它的母項繼承這三個檔案。這三個檔案最初是指派給工作站（0 指派給鍵盤，1 和 2 指派給顯示器）。 在將控制權移轉給指令之前，shell 允許將它們重新導向其他地方。

當您輸入指令時，如果沒有提供檔名，則鍵盤是標準輸入，有時會以 *stdin* 來表示。完成指令後，畫面上會顯示結果。

您的畫面是標準輸出，有時會以 *stdout* 來表示。依預設，指令是從標準輸入中取得輸入，並將結果傳給標準輸出。

錯誤訊息會被導向標準錯誤，有時表示成 *stderr*。就預設值而言，這是指您的畫面。

可以改變這些輸入及輸出的預設動作。您可將檔案當做輸入來使用，並將指令結果寫入檔案。這被稱為 輸入/輸出重新導向。

指令的輸出（通常會前往顯示裝置）可重新導向至檔案。此動作稱為輸出重新導向。當輸出太多導致畫面上不易閱讀時，或者當您要集中數個檔案來建立一個大型檔案時，此動作就很有用。

雖然指令輸入（通常是來自鍵盤）的重新導向不如輸出重新導向般常用， 但仍可從檔案重新導向。此動作稱為輸入重新導向。重新導向輸入可讓您事先準備檔案，然後讓指令讀取該檔案。

標準輸出重新導向
在指令尾端加入表示法 >*filename* 時，指令的輸出會寫入指定的檔名。> 符號即所謂的輸出重新導向運算子。

將其結果輸出到畫面的任何指令，可將其輸出重新導向至檔案。

將輸出重新導向到檔案
鍵入指令，接著鍵入輸出重新導向運算子及檔案名稱，即可將處理程序的輸出重新導向到檔案中。

例如，若要將 **who** 指令的結果重新導向至名為 `users` 的檔案，請鍵入：

```
who > users
```

註：如果 `users` 檔案已存在，則會刪除並置換該檔案，除非是指定 **set** 內建 **ksh** (Korn shell) 或 **csh** (C shell) 指令的 **noclobber** 選項。

若要查看 `users` 檔案內容，請鍵入：

```
cat users
```

會顯示類似以下的清單：

```
denise    lft/0 May 13 08:05
marta     pts/1 May 13 08:10
endrica   pts/2 May 13 09:33
```

重新導向輸出以附加至檔案
當表示法 >> *filename* 新增至指令尾端時，該指令的輸出會附加至指定的檔名中，而不會改寫任何現有的資料。>> 符號即所謂的附加重新導向運算子。

例如，若要將 `file2` 附加到 `file1`，請鍵入：

```
cat file2 >> file1
```

註：如果 `file1` 檔案不存在，則會建立該檔案，除非有指定 **set** 內建 **ksh** (Korn shell) 或 **csh** (C shell) 指令的 **noclobber** 選項。

從鍵盤中建立一個重新導向的文字檔
單獨使用，無論您在鍵盤上鍵入什麼，**cat** 指令都會把它當做輸入。您可將此輸入重新導向至某檔案。

在換行時按 Ctrl-D 來結束文字。

在提示時鍵入：

```
cat > filename
This is a test.
^D
```

文字檔連結
您可以將多個檔案結合成一個檔案。將不同檔案併成一個檔案，即所謂的連結。

下列範例建立了 `file4`，它是由 `file1`、`file2` 及 `file3` 所組成， 依給定的順序來附加。

請參閱下列範例：

· 在提示時鍵入：

```
cat file1 file2 file3 > file4
```

· 下列範例顯示連結檔案時的常見錯誤：

```
cat file1 file2 file3 > file1
```

小心： 在此範例中，您可能會預期 **cat** 指令將 file1、file2 及 file3 的內容附加至 file1。**cat** 指令會先建立輸出檔，讓它能夠實際消除 file1 的內容， 然後將 file2 及 file3 附加至其中。

標準輸入重新導向
當表示法 < *filename* 新增至指令尾端時，會從指定的檔名中讀取該指令的輸入。< 符號即所謂的輸入重新導向運算子。

註： 通常從鍵盤取得輸入的指令，才能將它們的輸入重新導向。

例如，若要使用 **mail** 指令，以訊息形式來傳送檔案 letter1 給使用者 denise，請鍵入：

```
mail denise < letter1
```

使用 /dev/null 檔案來捨棄輸出
/dev/null 檔是特殊檔案。本檔案具有一獨特性質：它一定是空的。會捨棄傳送至 /dev/null 的任何資料。 當您所執行的程式或指令產生您要忽略的輸出時，這個特性就很有用。

例如，您有一個程式 myprog，該程式接受畫面的輸入，並在執行時產生您想忽略的訊息。若要從 myscript 檔案讀取輸入，並捨棄標準輸出訊息，請鍵入：

```
myprog < myscript >/dev/null
```

在本例中，myprog 使用檔案 myscript 來作為輸入，並捨棄所有標準輸出。

標準錯誤及其他輸出重新導向
除了標準輸入及標準輸出之外，指令常會產生其他輸出類型，例如錯誤或狀態訊息，即所謂的診斷輸出。類似標準輸出，會將標準錯誤輸出寫入畫面（除非有重新導向它）。

若要重新導向標準錯誤或其他輸出，請使用檔案描述子。檔案描述子 是一個數字，它與指令所慣用的每一個 I/O 檔案有關。檔案描述子也可以被指定來重新導向標準輸入和標準輸出。下列是標準輸入、輸出與錯誤的相關數字：

項目	說明
0	標準輸入（鍵盤）
1	標準輸出（顯示器）
2	標準錯誤（顯示器）

若要重新導向標準錯誤輸出，請在輸出之前輸入檔案描述子號碼 2，或在符號之後附加重新導向符號（> 或 > >）及檔名。例如，下列指令會從用來編譯 testfile.c 檔案的 **cc** 指令取得標準錯誤輸出，並將其附加至 ERRORS 檔案結尾：

```
cc testfile.c 2 >> ERRORS
```

您也可以使用從 0 到 9 的檔案描述子來重新導向其他的輸出類型。比方說，如果 **cmd** 指令會將輸出寫入至檔案描述子 9，則您可以使用下列指令將輸出重新導向至 savedata 檔案：

```
cmd 9> savedata
```

如果某指令寫入一個以上的輸出，則您可以個別地對每一個輸出重新導向。假設某指令將標準輸出導向至檔案描述子 1、將標準錯誤輸出導向至檔案描述子 2、並且在檔案描述子 9 上建立資料檔。下面指令行將這些輸出分別導向另一個檔案：

```
command > standard 2> error 9> data
```

將輸出重新導向到列入輸入 (here) 文件
您可以將輸出重新導向至列入輸入 (here) 文件。

如果指令格式如下：

```
command << eofstring
```

且 *eofstring* 是不含型樣相符字元的任何字串， shell 就會一直將後續行當作 *command* 的標準輸入， 直到 shell 讀取到只包含 *eofstring* 的那一行為止（前面可能會有一個以上跳格字元）。在第一個 *eofstring* 與第二個之間的行，通常稱為列入輸入文件 或 *here* 文件。如果 << 重新導向字元之後緊接著一個連字號 (-)，則 shell script 會從 **here** 文件的每一行中刪除每一行的前導跳格字元， 之後才將該行傳送至 *command*。

shell 建立一個包含 **here** 文件的暫用檔，並且先對內容執行變數和指令替代， 之後才將檔案傳送給指令。它會對檔名執行型樣匹配，而這些檔名是指令替代中指令行的一部分。欲禁止全部替代，請用引號括住 *eofstring* 的全部字元：

```
command << \eofstring
```

對於存放於 shell 程序比保留在個別檔案中（如編輯器 Script）更方便的少量輸入資料而言，**here** 文件特別有用。例如，您可以輸入：

```
cat <<- xyz
      This message will be shown on the
      display with leading tabs removed.
   xyz
```

相關概念

Korn shell 或 POSIX shell 中的輸入及輸出重新導向
在 Korn shell 執行指令前，它會先掃描指令行尋找重新導向字元。這些特殊的表示法會將 shell 導入重新導向輸入及輸出。

使用管線及過濾重新導向輸出

您可以連接兩個以上的指令，將一個指令的標準輸出用作另一個指令的標準輸入。以這種方式連接的指令集就稱為管線。

參與指令集的連線稱為管線。管線是很有用的，因為它可讓您將許多單一目的的指令結合成一個強大的指令。您可使用管線將某個指令的輸出轉向，成為另一個指令的輸入。指令是以管線 (|) 符號連接。

當某個指令從另一個指令取得輸入、 經過修改、並將結果傳送至標準輸出時，此動作稱為過濾。您可以單獨使用過濾， 但是它們在管線中特別有用。最常見的過濾程式如下：

- sort
- more
- pg

請參閱下列範例：

- **ls** 指令以一個捲動資料串流的方式將現行目錄的內容寫入畫面。當出現一個以上的畫面資訊時，就無法看到某些資料。若要控制輸出以逐頁顯示畫面內容，您可以使用一個管線來將 **ls** 指令的輸出導向至 **pg** 指令， 該指令可控制畫面的輸出格式。例如，輸入：

  ```
  ls | pg
  ```

  在這個範例中，**ls** 指令的輸出會變成 **pg** 指令的輸入。請按 Enter 鍵來繼續下一個畫面。

  管線只能單向作業（左到右）。一個管線中的每一個指令皆當做個別處理程序來執行，並且可同時執行全部處理程序。當處理程序沒有輸入可讀取，或下一個處理的管線已滿時，處理程序即暫停。
- 使用管線符號的另一個例子是使用 **grep** 指令。**grep** 指令會在檔案中搜尋含有特定型樣之字串的指令行。若要顯示您在 7 月建立或修改的所有檔案，請鍵入：

  ```
  ls -l | grep Jul
  ```

  在這個範例中，**ls** 指令的輸出會變成 **grep** 指令的輸入。

顯示程式輸出並將它複製到檔案中（tee 指令）

tee 指令（與管線符號一起使用）可讀取標準輸入， 然後將程式的輸出寫至標準輸出，同時複製到指定的檔案。請使用 **tee** 指令來立刻檢視您的輸出，並同時將它儲存，以供日後使用。

例如，輸入：

```
ps -ef | tee program.ps
```

這會在顯示裝置上顯示 **ps -ef** 指令的標準輸出，同時將其副本儲存在 program.ps 檔中。如果 program.ps 檔案已存在，則除非指定 **set** 內建指令的 **noclobber** 選項，否則會刪除並取代該檔案。

例如，若要檢視指令的輸出，並將輸出儲存至現存檔案：

```
ls -l | tee -a program.ls
```

此指令會將 **ls -l** 的標準輸出顯示在顯示裝置，並同時將其副本附加至 program.ls 檔案尾端。

系統會顯示類似下列的資訊，且 program.ls 檔含有相同的資訊：

```
-rw-rw-rw-   1 jones   staff   2301   Sep 19    08:53 161414
-rw-rw-rw-   1 jones   staff   6317   Aug 31    13:17 def.rpt
-rw-rw-rw-   1 jones   staff   5550   Sep 10    14:13 try.doc
```

請參閱 *Commands Reference, Volume 5* 中的 **tee** 指令，以取得完整語法。

清除您的畫面（clear 指令）

使用 **clear** 指令來清空訊息畫面及鍵盤輸入。

在提示時鍵入：

```
clear
```

系統就會清除畫面並顯示提示。

傳送訊息到標準輸出

使用 **echo** 指令，在畫面上顯示訊息。

例如，若要將訊息寫入標準輸出，請在提示時鍵入：

```
echo Please insert diskette . . .
```

會顯示下列訊息：

```
Please insert diskette . . .
```

例如，若要使用 **echo** 指令搭配型樣相符字元，請在提示時鍵入：

```

echo The back-up files are: *.bak
```

系統會顯示訊息 The back-up files are:， 其後跟著現行目錄中，副檔名為 .bak 的檔名。

將單行文字附加到檔案（echo 指令）

使用 **echo** 指令，與附加重新導向運算子搭配使用，將單行文字新增到檔案。

例如，在提示時鍵入：

```
echo Remember to back up mail files by end of week.>>notes
```

這會將訊息 Remember to back up mail files by end of week. 新增至檔案 notes 的尾端。

將畫面複製到檔案（capture 及 script 指令）

使用 **capture** 指令，這會模擬 VT100 終端機， 將在您的終端機上所列印的所有內容複製到您指定的檔案。使用 **script** 指令，將在您的終端機上所列印的所有內容複製到您指定的檔案，而沒有模擬 VT100 終端機。

對於列印終端機對話的記錄而言，這兩個指令都很有用。

例如，若要在模擬 VT100 時擷取終端機畫面，請在提示時鍵入：

```
capture screen.01
```

系統會顯示如下的資訊：

```
Capture command is started. The file is screen.01.
Use ^P to dump screen to file screen.01.
You are now emulating a vt100 terminal.
Press Any Key to continue.
```

輸入資料並傾出畫面內容之後，請按 Ctrl-D 或，或鍵入 `exit` 並按 Enter 鍵，來停止 **capture** 指令。系統會顯示如下的資訊：

```
Capture command is complete. The file is screen.01.
You are NO LONGER emulating a vt100 terminal.
```

使用 **cat** 指令來顯示檔案的內容。

例如，若要擷取終端機畫面，請在提示時鍵入：

```
script
```

系統會顯示如下的資訊：

```
Script command is started. The file is typescript.
```

顯示在畫面上的所有項目，現在都已複製到 `typescript` 檔案。

若要停止 **script** 指令，請按下 Ctrl-D 或鍵入 `exit`，然後按 Enter 鍵。系統會顯示如下的資訊：

```
Script command is complete. The file is typescript.
```

使用 **cat** 指令來顯示檔案的內容。

請參閱 位於 *Commands Reference* 中的 **capture** 及 **script** 指令，以取得完整語法。

在畫面上以大寫字母顯示文字的指令（banner 指令）

banner 指令在畫面上以大寫字母顯示 ASCII 字元。

輸出中的每一行長度最多可達 10 個數字（或大小寫字元）。

例如，在提示時鍵入：

```

banner GOODBYE!
```

系統會在您的畫面上以大寫字母顯示 `GOODBYE!`。

輸入及輸出重新導向的指令摘要

下列是輸入及輸出重新導向的指令。

項目	說明
>	第 305 頁的『標準輸出重新導向』
<	第 306 頁的『標準輸入重新導向』
> >	第 305 頁的『重新導向輸出以附加至檔案』

項目	說明
|	第 307 頁的『使用管線及過濾重新導向輸出』
banner	以大寫字母將 ASCII 字串寫入標準輸出
capture	讓終端機畫面傾出至檔案
clear	清除終端機畫面
echo	將字串寫入標準輸出
script	讓終端機輸入和輸出複製到檔案
tee	顯示程式的標準輸出，並將其複製至檔案

AIX 核心回復

從 AIX 6.1 開始，核心就可以選擇性地從所選常式的錯誤中回復，從而避免發生不在計劃中的系統中斷。

根據預設值，會停用核心回復。如果已啟用核心回復，系統可能會在核心回復動作期間暫停一小段時間。這段時間通常小於 2 秒。核心回復動作之後，會立即發生下列動作：

- 系統主控台會顯示下列訊息：

```
-------------------------------------------------------------------------
        A kernel error recovery action has occurred. A recovery log
         has been logged in the system error log.
-------------------------------------------------------------------------
```

- AIX 將項目新增到錯誤日誌中。您可以將錯誤日誌資料傳送到 IBM 以尋求服務，類似於從完整系統終止傳送資料。下列是範例回復錯誤日誌項目：

```
LABEL:          RECOVERY
Date/Time:       Fri Feb 16 14:04:17 CST 2007
Type:            INFO
Resource Name:   RMGR
  說明
Kernel Recovery Action
Detail Data
Live Dump Base Name
RECOV_20070216200417_0000
Function Name
w_clear
FRR Name
w_init_clear_frr
Symptom String
273
EEEE00009627A072
F10001001B18BBC0
w_clear+D0
wdog0030+288
test_index+4C
Recovery Log Data
0001 0000 0000 0000 F000 0000 2FFC AEB0 0000 0111 0000 0000 0000 0000 0021 25BC
8000 0000 0002 9032 EEEE 0000 9627 A072 F100 0100 1B18 BBC0 0000 0000 0000 0000
0000 0001 0000 0000 0006 0057 D2FF 8C00 0001 0148 0500 0000 8000 0000 0002 9032
.....
```

- AIX 會產生即時傾出。根據預設值，即時傾出的資料位於 `/var/adm/ras/livedump` 目錄中，且檔案命名為 **RECOV_*timestamp*_*sequence***，其中 *timestamp* 指定發生核心回復的時間，而 *sequence* 指定呼叫核心回復的次數。您可以將即時傾出資料傳送到 IBM 以尋求服務，類似於從完整系統終止傳送資料。如需 Live Dump 的相關資訊，請參閱 Live Dumps 位於 *Kernel Extensions and Device Support Programming Concepts*。

注意：核心回復之後可能會遺失部分功能，但是作業系統仍保持處於穩定的狀態。必要的話，請關閉並重新啟動系統，以還原遺失的功能。

記憶體及處理器注意事項

AIX 在主線核心作業期間會將資料維持在核心回復的狀態。啟用核心回復時，需要其他處理器指示來維持資料，並需要其他記憶體來儲存資料。這對處理器使用情況的影響最小。其他記憶體用量可透過下列方程式進行判斷，其中 *maxthread* 是在系統上執行的執行緒數目上限，而 *procnum* 是處理器數目：

```
memory required = 4 KB x maxthread + 128 KB x procnum
```

如下列範例所示，具有 16 個處理器且最多 1000 個執行緒的系統會額外耗用 6304 KB：

```
4 x 1000 + 128 x 16 = 6304 KB
```

啟用及停用核心回復

您可以從 SMIT 路徑介面啟用或停用核心回復。

若要啟用或停用核心回復，請使用下列 SMIT 路徑：

問題判斷 > 核心回復 > 變更核心回復狀態 > 變更下一個開機核心回復狀態

您也可以使用下列 SMIT 路徑，來顯示現行核心回復狀態：

問題判斷 > 核心回復 > 顯示核心回復狀態

AIX Event Infrastructure for AIX 及 AIX 叢集 - AHAFS

AIX Event Infrastructure for AIX 及 AIX 叢集包含用來監視預先定義及使用者定義事件的事件監視架構。

AIX Event Infrastructure 簡介

AIX Event Infrastructure 是用於監視預先定義及使用者定義事件的事件監視架構。

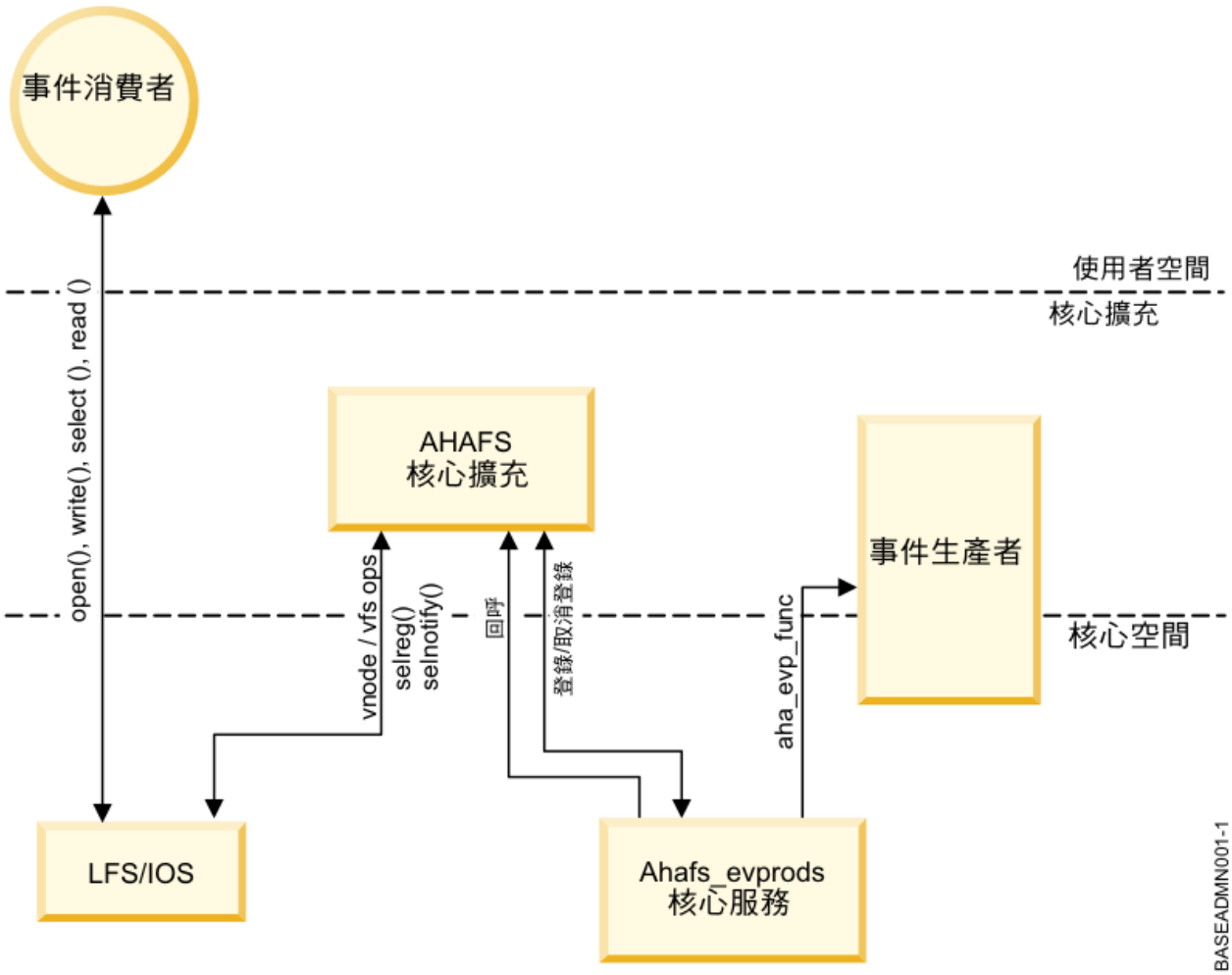
在 AIX Event Infrastructure 中，事件是定義為核心或核心擴充可以在發生變更時偵測到的任何狀態或值變更。可以監視的事件是以虛擬檔案系統中的檔案形式呈現。AIX Event infrastructure 的部分優點如下：

- 不需要進行持續的輪詢。事件發生時，監視那些事件的使用者會獲得通知。
- 會將事件的相關詳細資訊（如堆疊追蹤以及使用者和處理程序資訊）提供給監視事件的使用者。
- 使用現有的檔案系統介面，因此不需要新的應用程式設計介面 (API)。
- 在事件發生的確切時間，將控制權轉給 AIX Event Infrastructure。

AIX Event Infrastructure 元件

AIX Event Infrastructure 是由下列四個元件所構成：

- 實作虛擬檔案系統的核心擴充。
- 耗用事件的事件消費者。
- 產生事件的事件生產者。
- 作為核心擴充與事件生產者之間介面的核心元件。



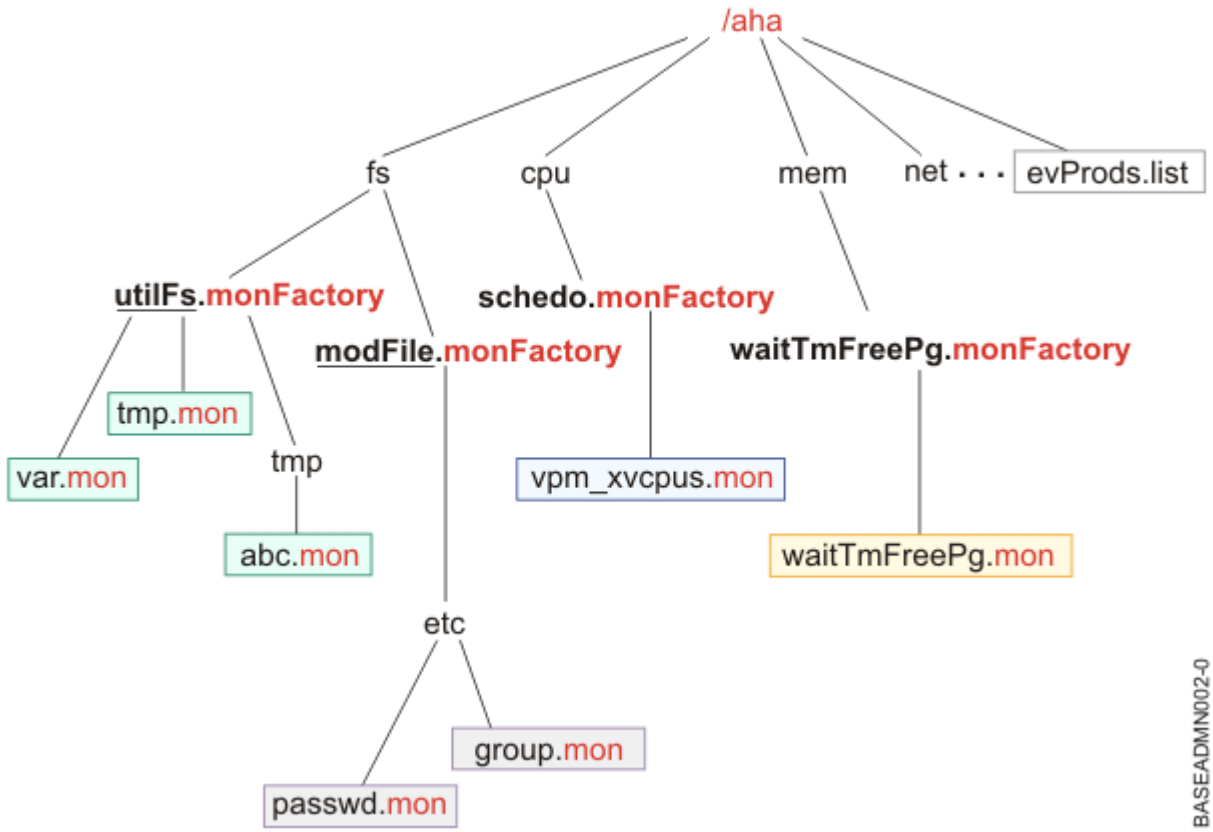
AIX Event Infrastructure 核心擴充
AIX Event Infrastructure 核心擴充會實作虛擬檔案系統。

所有事件都是以此檔案系統中的檔案形式呈現。檔案物件類型有四種：

- **.list 檔案**：在虛擬檔案系統 **evProds.list** 中，只有一個 **.list** 檔案。這是特殊檔案，在讀取此檔案時會傳回所有目前定義的事件生產者的名稱。
- **.monFactory 目錄**：監視器 Factory 是特殊類型的目錄。這些是事件生產者的目錄呈現。會自動為使用者建立監視器 Factory 目錄及其上層子目錄。
- **子目錄**：子目錄的用途是要簡化管理，以及呈現監視器檔案的完整路徑名稱（請參閱 **.mon 檔案**）。
- **.mon 檔案**：監視器檔案會呈現可以監視的事件。上層監視器 Factory 的監視器檔案的完整路徑名稱，去掉副檔名 **.mon**，就是所監視事件的完整呈現。例如，檔案 **/aha/fs/modFile.monFactory/etc/password.mon** 是用來監視對 **/etc/passwd** 檔案的修改。 監視器檔案只能存在於監視器 Factory 下方。

在此虛擬檔案系統中，無法建立其他一般檔案。 因為 AIX Event Infrastructure 檔案系統是記憶體內檔案系統，所以最大可能會有 32 KB 的 inode。使用的 inode 數目會顯示在 **df** 指令輸出中。

AIX Event Infrastructure 檔案系統佈置的範例顯示如下：



註：

evProds.list 檔案直接存在於檔案系統的根目錄下方，而且包含在此作業系統實例下定義以及可使用的事件生產者清單。

使用 LFS 介面，AIX Event Infrastructure 會將寫入監視器檔案的文字輸入，轉換為詳細說明，載明使用者在事件發生時想如何獲得通知。一旦使用者發出 **select()** 或封鎖 **read()** 呼叫以表示其監視開始之後，AIX Event Infrastructure 就會通知對應的事件生產者開始監視指定的事件。

偵測到發生事件時，AIX Event Infrastructure 會通知符合其監視準則的所有等待中消費者。

事件消費者
事件消費者是等待事件發生的使用者空間處理程序。

消費者會將資訊寫入監視器檔案來設定事件監視作業，以指定如何及何時應該通知他們。消費者可以在 **select()** 呼叫或封鎖 **read()** 呼叫中等待事件通知。

AIX Event Infrastructure 並非執行緒安全。處理程序不應該使用多個執行緒來監視相同的事件。

事件生產者
事件生產者是可以偵測事件的核心或核心擴充內的程式碼區段。

發生監視的事件時，事件生產者會通知 AIX Event Infrastructure 核心擴充，並將有關要傳遞之事件的任何相關資訊傳送給消費者。

目前，事件生產者有兩種主要類別：

- 監視狀態變更的事件生產者
- 監視值是否超出使用者所指定臨界值的事件生產者

ahafs_evprods 核心服務
ahafs_evprods 核心服務可促進 AIX Event Infrastructure 核心擴充與事件生產者之間的通訊。

若要促進 AIX Event Infrastructure 核心擴充與事件生產者之間的通訊，會匯出 **ahafs_evprods** 核心服務。在核心內，會使用已登錄事件生產者的清單來查閱事件生產者，以及在適當的事件生產者與核心擴充之間傳遞資訊。

設定 AIX Event Infrastructure

設定 AIX Event Infrastructure 的必要步驟。

設定 AIX Event Infrastructure 唯一需要的步驟如下：

1. 安裝 **bos.ahafs** 檔案集。
2. 建立所要裝載點的目錄。
3. 執行下列指令：

```
mount –v ahafs <mount point> <mount point>
```

範例

```
mkdir /aha
mount -v ahafs /aha /aha
```

裝載 AIX Event Infrastructure 檔案系統會自動載入核心擴充，以及建立所有監視器 Factory。一次可能只會裝載一個 AIX Event Infrastructure 檔案系統實例。AIX Event Infrastructure 檔案系統可能會裝載於任何一般目錄上，但是建議使用者使用 **/aha**。

AIX Event Infrastructure 運作方式的高階視圖

一位消費者可能會監視多個事件，而多位消費者可能會監視相同的事件。每位消費者都可能會監視具有不同臨界值的數值型事件。為了處理這種情況，AIX Event Infrastructure 核心擴充會保留每位消費者資訊的清單，包括：

- 指定的等待類型（**WAIT_IN_READ** 或 **WAIT_IN_SELECT**）
- 要求的資訊層次
- 要監視的臨界值（如果監視臨界值事件的話）
- 用來保留事件發生相關資訊的緩衝區。

事件資訊是根據處理程序來儲存，如此一來，監視相同事件的不同處理程序便不會變更事件資料。消費者處理程序在讀取監視器檔案時，也只會讀取它自己的那份事件資料副本。

監視事件的一般流程

本主題說明監視事件的步驟。

1. 處理程序會嘗試開啟或建立監視器檔案。
2. AIX Event Infrastructure 會將監視器檔案的路徑名稱傳遞給適當的事件生產者。 事件生產者會驗證監視器檔案是否呈現有效的事件，以及處理程序是否具有監視事件的存取權。
3. 處理程序會將資訊寫入至檔案，並指定：
   a. 等待類型（**WAIT_TYPE=WAIT_IN_READ** 或 **WAIT_TYPE=WAIT_IN_SELECT**）。 預設等待類型是 **WAIT_IN_SELECT**。
   b. 何時通知。針對狀態變更事件，使用者必須指定 **CHANGED=YES**。 針對臨界值事件，使用者可以根據相關聯事件生產者的功能，指定 **THRESH_HI**=<value> 及（或）**THRESH_LO**=<value>。 此指定沒有預設值，而且可能未指定 **CHANGED=YES** 及 **THRESH_***=<value> 兩者。
4. AIX Event Infrastructure 接著會配置每個處理程序的區塊以儲存此資訊（如果此處理程序還沒有這類區塊的話），並在其中填入使用者所寫入的資訊。
5. 處理程序會在監視器檔案上發出 **select()** 或封鎖 **read()**
6. AIX Event Infrastructure 會呼叫 **ahafs_evprods**，以檢查指定的臨界值是否適用於此特定事件。例如，**utilFs** 事件生產者不容許 > 100% 的值。如果臨界值無效，則 **select()** 或 **read()** 呼叫會傳回 **RC_FROM_EVPROD**，而在讀取監視器檔案時則會傳回 **EINVAL**。

7. 針對臨界值事件生產者，只會將一個值傳送至每個臨界值的事件生產者（**hi** 或 **lo**），以進行監視。在 **select()** 或封鎖 **read()** 時，如果符合下列其中一項，則 AIX Event Infrastructure 會向事件生產者登錄此新臨界值：
   a. 如果沒有其他處理程序正在監視此事件，則會向事件生產者登錄這個消費者所指定的臨界值。
   b. 如果有其他處理程序正在監視此事件，且當消費者所指定的 **THRESH_LO** 高於目前監視的低臨界值，或是消費者所指定的 **THRESH_HI** 低於目前監視的高臨界值時，則 AIX Event Infrastructure 會呼叫 **ahafs_evprods** 核心服務以更新目前監視的臨界值。
8. 在部分情況下，從 **ahafs_evprods** 核心服務返回時，會傳回事件的實際值。如果傳回的實際值已符合或超出任一臨界值，則 **read()** 或 **select()** 呼叫會立即返回，而事件緩衝區中所記載的 **RC_FROM_EVPROD** 會是 **EALREADY**。**read()** 或 **select()** 呼叫會傳回 0。
9. 針對狀態變更事件生產者，一律會呼叫 **ahafs_evprods** 函數，以登錄事件。
10. 登錄成功時，AIX Event Infrastructure 會設定通知。針對 **select()** 中等待的消費者，會透過 **selreg()** 設定通知。針對 **read()** 呼叫中封鎖的消費者，會使用 **e_sleep_thread()**，讓執行緒進入休眠狀態。
11. 一旦事件生產者偵測到發生事件之後，便會向 AIX Event Infrastructure 通知事件相關資訊（即觸發事件的處理程序、現行值、回覆碼等相關資訊）。
12. 在事件生產者的這個回呼期間，AIX Event Infrastructure 會：
    a. 判斷 **ahaNode** 是否對應於事件
    b. 搜尋等待中消費者的清單，判斷哪些消費者的臨界值已符合或已超出，以使用 **selnotify()** 或 **e_wakeup()** 呼叫進行通知。 會通知所有等待狀態變更事件的消費者。
13. 一旦處理程序獲得發生事件的通知時，它會讀取監視器檔案以取得事件資料。事件輸出的範例如下。

    已指定應該進行堆疊追蹤的狀態變更事件生產者的輸出範例：

```
BEGIN_EVENT_INFO
TIME_tvsec=1269377315
TIME_tvnsec=955475223
SEQUENCE_NUM=0
PID=2490594
UID=0
UID_LOGIN=0
GID=0
PROG_NAME=cat
RC_FROM_EVPROD=1000
END_EVENT_INFO
```

    臨界值事件的範例：

```
BEGIN_EVENT_INFO
TIME_tvsec=1269378095
TIME_tvnsec=959865951
SEQUENCE_NUM=0
CURRENT_VALUE=2
RC_FROM_EVPROD=1000
END_EVENT_INFO
```

註：由於處理程序通知的非同步本質，傳回的現行值可能在處理程序讀取監視器檔案時便已過時。第一次符合或超出臨界值時，會通知使用者，但是不會封鎖其他可能會變更所監視值的作業。

使用 AIX Event Infrastructure

AIX Event Infrastructure 檔案系統中所有目錄的存取模式為 1777，而所有檔案的存取模式為 0666。

目前，AIX Event Infrastructure 檔案系統中所有目錄的模式為 1777，而所有檔案的模式為 0666。無法變更這些模式，但是可以變更檔案及目錄的所有權。用於監視事件的存取控制是在事件生產者層次進行。建立/修改時間未進行維護，而且一律會以在虛擬檔案系統的檔案物件上發出 **stat ()** 的現行時間來傳回。任何修改這些時間的嘗試，都會傳回錯誤。

監視事件

建立監視器檔案

必須建立對應於事件的監視器檔案，以監視事件。

監視事件之前，必須建立對應於事件的監視器檔案。AIX Event Infrastructure 確實支援具有 **O_CREAT** 旗標的 **open()**。例如，我們會遵循必要的步驟，以監視檔案系統 **/filesys/clj-fs** 的使用率是否達到 90%。

· 也必須建立必要的子目錄：

```
mkdir /aha/fs/utilFs.monFactory/filesys
```

· 開啟檔案 **/aha/fs/utilFs.monFactory/filesys/clj-fs.mon**。

可以建立監視器檔案之前，AIX Event Infrastructure 核心擴充會呼叫事件生產者，以判斷所要求的事件是否有效，以及判斷使用者是否具有足夠的權限可以監視指定的事件。這裡是建立或開啟監視器檔案時，可能傳回的一些常見錯誤：

表 *62.* 回覆碼

回覆碼	**詳細資料**
ENODEV	沒有對應於所指定路徑的事件。 **註 :** 如果在事件不再存在時，嘗試開啟現有的監視器檔案，則可能還是會傳回 **ENODEV** 錯誤。
EPERM	使用者沒有許可權可以監視指定的事件。
ENOTSUP	指定的事件不支援由 AIX Event Infrastructure 進行的監視。

寫入監視器檔案

消費者處理程序會寫入監視器檔案，以指定應該如何及何時通知發生事件。

一旦建立及開啟想要的監視器檔案之後，消費者處理程序便會寫入監視器檔案，以指定應該如何及何時通知發生事件。此資料是以 **<索引鍵>=<值>** 配對寫入，而配對可能以 ; 或空格隔開。 可接受的 **<索引鍵>=<值>** 配對如下：

表 *63.* 可接受的 <索引鍵>=<值> 配對

索引鍵	**可接受的值**	**詳細資料**
CHANGED	YES	指定所要監視事件的類型為 **AHAFS_THRESHOLD_STATE**，而且應該在事件狀態變更時通知消費者。
THRESH_HI	無正負號、64 位元整數、以十進位格式指定	此索引鍵指定事件的高臨界值。一旦事件已達此臨界值（等於或大於）之後，便會通知消費者。 **註 :** 雖然這是一個 64 位元整數，但是部分事件生產者可能會限制可以實際監視的值。例如，**utilFs** 事件生產者的 **THRESH_HI** 的可接受值介於 1 與 100（含）之間。在寫入時（而不是在 **select()** 或封鎖 **read()** 時），不會檢查事件生產者的臨界值有效性。

表 *63.* 可接受的 <索引鍵>=<值> 配對 *(*繼續*)*		
索引鍵	**可接受的值**	**詳細資料**
THRESH_LO	無正負號、64 位元整數、以十進位格式指定	此索引鍵指定事件的低臨界值。一旦事件已達此臨界值（等於或小於）之後，便會通知消費者。 **註 :** 雖然這是一個 64 位元整數，但是部分事件生產者可能會限制可以實際監視的值。例如，**utilFs** 事件生產者的 **THRESH_LO** 的可接受值介於 1 與 100（含）之間。在寫入時（而不是在 **select()** 或封鎖 **read()** 時），不會檢查事件生產者的臨界值有效性。
WAIT_TYPE	WAIT_IN_SELECT（預設）、WAIT_IN_READ	指定消費者等待事件的方式。如果消費者想要封鎖 **select()** 呼叫中的事件，則應該指定 **WAIT_IN_SELECT**。如果消費者想要封鎖 **read()** 呼叫中的事件，則應該指定 **WAIT_IN_READ**。
INFO_LVL	1、2（預設）或 3	指定應該記載至使用者緩衝區中的事件資料： · **INFO_LVL=1** 會記載事件的時間戳記、序號、事件生產者回覆碼、使用者資訊*、處理程序資訊*、程式名稱*，以及事件的現行值（適用時）。 · **INFO_LVL=2** 會記載層次 1 的所有資料，以及來自事件生產者的訊息（適用時）。 · **INFO_LVL=3** 會記載層次 2 的所有資料，以及事件的堆疊（適用時）。 **註 :** 使用者資訊、處理程序資訊、程式名稱及堆疊追蹤，僅可供已指定 **AHAFS_STKTRACE_AVAILABLE** 旗標的事件生產者使用。並非所有事件生產者都會傳遞訊息。請參閱事件生產者文件，以判斷可用的資訊。事件輸出範例顯示於第 320 頁的『讀取事件資料』小節中。

表 63. 可接受的 <索引鍵>=<值> 配對 (繼續)		
索引鍵	**可接受的值**	**詳細資料**
NOTIFY_CNT	-1（預設）或 1 與 32767（含）之間的任何值	**NOTIFY_CNT** 指定事件在通知處理程序之前應該發生的次數。如果指定 -1 值，則會在每次發生事件時通知消費者，而且每一次的事件發生都會記載至使用者緩衝區中。如果消費者指定非零的正數值，則會在事件發生指定的次數之前，先封鎖消費者。一旦事件發生指定的次數之後，在消費者於另一個 **select()** 或封鎖 **read()** 呼叫中封鎖之前，不會再記載其他事件。如需相關資訊，請參閱第 319 頁的『等待事件』小節。
CLUSTER	YES	如果系統是叢集的一部分，而且叢集作用中，則消費者可能會指定此索引鍵，以便叢集的其他節點上發生此事件時收到通知。並非所有事件生產者都支援整個叢集的監視。此特性預設為 **off**。如需相關資訊，請參閱第 338 頁的『叢集事件』小節。
BUF_SIZE	正整數、最大為 1048576	此索引鍵指定應該用來記錄事件資料的緩衝區大小（以位元組為指定單位）。預設大小是 2048，而配置的最小大小會是 1024 個位元組，即使消費者要求較小的大小也是一樣。

將資訊寫入監視器檔案，只會準備後續的 **select()** 或封鎖 **read()** 呼叫。在 **select()** 或封鎖 **read()** 完成之前，不會啟動監視。

例如，若要監視檔案系統 **/filesys/clj-fs** 的封鎖 **read()** 呼叫中何時第一次出現使用率 90%，則會將下列字串寫入 **/aha/fs/utilFs.monFactory/filesys/clj-fs.mon** 檔案：

```
WAIT_TYPE=WAIT_IN_READ THRESH_HI=90 NOTIFY_CNT=1
```

對監視器檔案的 **write()** 呼叫的可能回覆碼：

表 64. 回覆碼	
回覆碼	**詳細資料**
EINVAL	如果提供無效值給上述任何索引鍵，則寫入監視器檔案會失敗，並產生 **EINVAL**。 此外，如果指定的通知參數（**CHANGED** 或 **THRESH_HI/LO**）不符合事件生產者的功能，則寫入會失敗，並產生 **EINVAL**。例如，如果消費者指定 **utilFs** 事件生產者的 **CHANGED=YES**（只會監視 **THRESH_HI/LO**），則寫入呼叫會傳回 **EINVAL**。指定沒有作用中叢集的 **CLUSTER=YES**，也會導致 **EINVAL**。

表 64. 回覆碼 (繼續)

回覆碼	詳細資料
EBUSY	如果處理程序中有另一個執行緒目前正在等待事件，則任何其他執行緒寫入監視器檔案，都會傳回 **EBUSY**。
ESTALE	已刪除監視器檔案。若要監視此事件，則需要關閉檔案描述子，然後使用 **O_CREAT** 重新開啟
ENOMEM	無法配置暫存記憶體或事件緩衝區的記憶體。
ENOSPC	最多可能會有 512 個處理程序來監視監視器檔案。如果有 512 個已開啟此檔案且已寫入至其中的處理程序，則寫入會失敗，並產生 **ENOSPC**。

等待事件
監視指定會寫入監視器檔案。

一旦將監視指定順利寫入監視器檔案之後，消費者處理程序便會使用 **select()** 或 **read()** 來封鎖事件發生。只在 **select()** 或 **read()** 中封鎖發生的事件之後，才會通知消費者發生事件。 處理程序可以利用三種方式，從 **select()** 或封鎖 **read()** 返回：

1. 事件發生指定的次數。

   · 非錯誤的情況。消費者應該讀取事件資料，以判斷如何處理事件。

2. 設定 AIX Event Infrastructure 核心擴充內的事件時發生問題。

   向事件生產者登錄事件以進行監視之前，可能發生錯誤：

   · **read()**

   – 如果有另一個執行緒正在等待讀取，則讀取會失敗並產生 **EBUSY**
   – 如果在此讀取之前未完成任何寫入，則讀取只會傳回 0（即讀取 0 個位元組）。

   · **select()**

   註 :

   因為實作 select 系統呼叫，所以若要讓 **select()** 傳回錯誤，則基礎檔案系統作業必須傳回 **EBADF**。因此，如果符合下列任一條件，則 **select()** 會傳回 **EBADF**。

   – 另一個執行緒正在嘗試 select
   – 已刪除監視器檔案
   – 指定監視指定時未完成任何寫入
   – 向 IOS 子系統登錄時發生錯誤

   在這些情況下，沒有要讀取的事件資料。

3. 向事件生產者設定事件時發生問題。

   如果嘗試向事件生產者登錄事件，則會在緩衝區中記載項目，以供消費者讀取。 若要判斷發生的錯誤，則應該在事件生產者的文件中參照事件資料中所傳回的 **RC_FROM_EVPROD**。 請注意，此情況的事件輸出只會包含事件生產者的時間戳記、序號及回覆碼，而不論指定的 **INFO_LVL** 為何。如需範例，請參閱第 320 頁的『讀取事件資料』。

   在此情況下，**select()** 會傳回 **EBADF**，但 **read()** 會傳回基礎 **uio_move** 作業的回覆碼。

如果消費者處理程序已指定大於 1 的 **NOTIFY_CNT**，則在發生事件要求次數之前，每個事件發生的相關資訊都會記載至消費者的緩衝區。只有在事件發生所要求的次數或是發生無法使用的事件時，才會起動消費者處理程序。一旦起動消費者處理程序之後，除非重新發出監視器檔案的 **select()** 或封鎖 **read()** 呼叫，否則不會再監視事件。

如果消費者已將 **NOTIFY_CNT** 指定為 -1，則會在每次發生事件之後起動消費者處理程序，而且任何在初始成功的 **select()** 或封鎖 **read()** 之後發生的事件，都會記載至消費者緩衝區。

如果緩衝區中有未讀取的事件資料，就不會封鎖 **select()** 及 **read()** 呼叫。

無法使用的事件發生
對於部分事件生產者而言，可能會有事件發生而導致受監視事件變成無效。

部分範例如下：

- **processMon** 及 **pidProcessMon** 的處理程序損毀。
- 卸載含有 **modDir** 及 **modFile** 的受監視檔案的檔案系統。
- 卸載由 **utilFs** 進行監視的檔案系統。
- 移除或重新命名由 **modDir** 或 **modFile** 進行監視的檔案
- 移除目前用來監視事件的事件生產者（在此情況下，**RC_FROM_EVPROD** 會是 **ENODEV**）。

一旦觸發無法使用的事件發生之後，除非事件重新變成有效，否則消費者可能無法繼續監視該事件。重新變成有效的事件範例：

- 重新裝載受監視檔案系統。
- 重建已刪除的受監視檔案。
- 重建受監視的處理程序。

觸發本端無法使用的事件時，AIX Event Infrastructure 核心延伸會移除受影響的監視器檔案。刪除監視器檔案時，仍然開啟檔案的消費者可以讀取其事件資料，但無法寫入或封鎖該監視器檔案上的等待事件發生。消費者發現無法使用的事件發生時，應該採取適當的動作（很可能會導致事件重新變成有效）、關閉監視器檔案的檔案描述子，以及使用 **O_CREAT** 旗標重新開啟監視器檔案。

如果消費者已指定 **NOTIFY_CNT > 1**，則在觸發要求的事件發生次數之前，本端無法使用的事件發生也會導致 **select()** 及 **read()** 取消封鎖。例如，如果消費者正使用 **NOTIFY_CNT=3** 來監視檔案 **/foo**，則在移除 **/foo** 時，消費者會從 **select()** 或 **read()** 返回，即使它是具有 **/foo** 的事件第一次發生也是一樣。

使用 AIX Event Infrastructure 進行輪詢
AIX Event Infrastructure 不需要事件生產者永遠維護可能受監視之事件的現行值。

這樣可以有較高的效能，因為如果沒有人監視是否發生事件，事件生產者就沒有維護此值的額外負擔。

這樣會在使用同步輪詢時產生問題。因為不可能一直會取得事件在每個時間點的現行值，所以會使用下列方式來處理 **poll()** 或同步 **select()** 呼叫：

- 處理程序第一次在監視器檔案上發出 **select()** 或 **poll()** 時，AIX Event Infrastructure 核心擴充會向事件生產者登錄該事件，以進行監視。
  - 對於維護現行值的臨界值事件生產者，會在事件登錄時將現行值傳回給 AIX Event Infrastructure 核心擴充。此時，會針對消費者臨界值來檢查此值。如果已超出消費者的臨界值，則 **select()** 或 **poll()** 會指出事件已發生，而且 **RC_FROM_EVPROD** 會是 **EALREADY**。
- POLLSYNC 旗標會遭忽略。除非事件發生指定的次數，或使用者關閉檔案，否則還是會持續向事件生產者登錄事件。
- 後續 **poll()** 呼叫的行為如下：
  - 如果尚未發生事件，則呼叫會返回，但無任何傳回事件
  - 如果事件自前次 **poll()** 呼叫之後已發生指定的次數，則會設定傳回事件，以指出事件已發生。

讀取事件資料
AIX Event Infrastructure 中的事件資料包含關鍵字/值配對。

事件資料可能只會讀取一次，而且在單一 **read()** 呼叫中，不會傳回一個以上事件發生的有用資料。例如，假設在消費者讀取監視器檔案之前發生兩個事件，而且每個事件都有 256 個位元組的有用資料。 如果消費者呼叫 **read()** 以取得 4096 個位元組，則只會將第一個事件的 256 個位元組傳回給使用者。因此，需要執行第二個 **read()** 呼叫，以取得第二個事件的資料。給定的任何偏移都會予以忽略，而且會從最後一個未讀取的位元組開始讀取資料。

雖然大部分的事件會小的多（< 512 個位元組），但是事件資料最多會有 4096 個位元組。建議在讀取事件時，應該使用夠大的緩衝區，以避免只讀取到事件的一部分。

AHAFS 中的事件資料包含**關鍵字 = 值**的配對，但 **BUF_WRAP**、**EVENT_OVERFLOW**、**BEGIN_EVENT_INFO**、**END_EVENT_INFO**、**BEGIN_EVPROD_INFO**、**END_EVPROD_INFO** 及 **STACK_TRACE** 例外，這些是沒有任何值的特殊關鍵字。這裡是您可能會在事件資料中看到的關鍵字：

表 *65.* 關鍵字

索引鍵	**值**	**詳細資料**
BUF_WRAP	無	消費者緩衝區的處理方式與循環緩衝區類似。如果最新的事件資料改寫任何未讀取的資料，則此關鍵字會是 **read()** 呼叫所傳回的下一個字串，即使消費者已讀取前一個項目的某些部分也是一樣。後續的 **read()** 呼叫會傳回下一個完整事件。
EVENT_OVERFLOW	無	如果事件資料太大以致於無法放入消費者的事件資料緩衝區，則會從第一個 **read()** 傳回此關鍵字。後續的 **read()** 則會傳回可以放入緩衝區的事件資料。 **註 :** 如果發生 **EVENT_OVERFLOW**，則不會有結束字串 **END_EVENT_INFO**。
BEGIN_EVENT_INFO	無	此關鍵字表示事件發生的資料開始。
END_EVENT_INFO	無	此關鍵字表示這個特定事件發生的資料結束。
TIME_tvsec TIME_tvnsec	整數	這兩個欄位會記錄自新紀元之後的事件發生時間戳記（秒及奈秒）。
SEQUENCE_NUM	整數	此欄位記錄自第一個成功 **select()** 或封鎖 **read()** 之後的事件發生次數。 如果 **select()** 或封鎖 **read()** 呼叫失敗，或如果消費者停止監視事件（透過改寫事件監視指定，或是事件發生計數等於指定的 **NOTIFY_CNT**），則此數字會重設為 0。
PID	整數	已觸發事件發生的處理程序的處理程序 ID。只有已指定 **AHAFS_STKTRACE_AVAILABLE** 功能而非 **AHAFS_CALLBACK_INTRCNTX** 功能的事件生產者才能使用。

表 65. 關鍵字 (繼續)

索引鍵	值	詳細資料
UID	整數	已觸發事件發生的使用者的有效使用者 ID。只有已指定 **AHAFS_STKTRACE_AVAILABLE** 功能而非 **AHAFS_CALLBACK_INTRCNTX** 功能的事件生產者才能使用。
UID_LOGIN	整數	已觸發事件發生的使用者的登入使用者 ID。只有已指定 **AHAFS_STKTRACE_AVAILABLE** 功能而非 **AHAFS_CALLBACK_INTRCNTX** 功能的事件生產者才能使用。
GID	整數	已觸發事件發生的使用者的有效群組 ID。只有已指定 **AHAFS_STKTRACE_AVAILABLE** 功能而非 **AHAFS_CALLBACK_INTRCNTX** 功能的事件生產者才能使用。
PROG_NAME	字串	已觸發事件發生的處理程序的名稱。只有已指定 **AHAFS_STKTRACE_AVAILABLE** 功能而非 **AHAFS_CALLBACK_INTRCNTX** 功能的事件生產者才能使用。
CURRENT_VALUE	64 位元無正負號整數、十進位	此索引鍵僅適用於 **AHAFS_THRESHOLD_VALUE** 事件生產者，而且會傳回事件在偵測到事件發生時的現行值。請注意，因為通知處理程序的時間與它們讀取事件資料的時間之間的延遲，所以事件的實際現行值可能已變更。
RC_FROM_EVPROD	32 位元整數、十進位	此回覆碼來自事件生產者，原因是嘗試設定事件時發生錯誤，或是發生事件。一般而言，小於 256 的回覆碼指出嘗試向事件生產者登錄事件時發生錯誤。部分事件生產者會傳回大於 256 的代碼，以提供事件發生的更多相關資訊。這些回覆碼都是記載在 **sys/ahafs_evProds.h** 中
BEGIN_EVPROD_INFO **END_EVPROD_INFO**	字串*	這兩個關鍵字會標示事件生產者所傳回字串的開始及結束。而在 **BEGIN_EVPROD_INFO** 後面及 **END_EVPROD_INFO** 前面，一定會是換行字元。 對於已指定 **CLUSTER=YES** 的消費者而言，這是要提供節點資訊的位置。

表 65. 關鍵字 (繼續)

索引鍵	值	詳細資料
STACK_TRACE	字串*	對於已指定 **INFO_LVL=3** 的消費者，以及已指定 **AHAFS_STKTRACE_AVAILABLE** 功能而非 **AHAFS_CALLBACK_INTRCNTX** 功能的事件生產者而言，會提供事件發生的堆疊追蹤。除非字串 **END_EVENT_INFO** 是事件發生的堆疊，否則關鍵字 **STACK_TRACE** 會指出其餘的事件資料。
NUM_EVDROPS_INTRCNTX	整數	這個關鍵字代表報告中 TIME0_tvsec 和 TIME0_tvnsec 關鍵字所指定的時間以來，所捨棄的岔斷環境定義事件出現項目數。唯有當事件出現項目的頻率太高時，才會捨棄事件出現項目。
TIME0_tvsec TIME0_tvnsec	整數	這些關鍵字記錄自新紀元以來，第一次捨棄事件出現項目的時間戳記（以秒及奈秒為單位）。這些關鍵字會連同 NUM_EVDROPS_INTRCNTX 關鍵字一起報告。

重複事件合併

如果在消費者讀取資料之前，相同的事件發生多次，則會將重複項目合併為一個項目。此合併是由沒有對應 **BUF_WRAP** 關鍵字的非循序序號所指出。時間戳記及序號會反映出最近發生的事件。

事件資料範例

對於已指定 **AHAFS_THRESHOLD_STATE** 及 **AHAFS_STKTRACE_AVAILABLE**，而且會將訊息傳遞給事件消費者的事件生產者而言，三種輸出層次如下：

INFO_LVL=1	INFO_LVL=2	INFO_LVL=3
BEGIN_EVENT_INFO TIME_tvsec=1269863383 TIME_tvnsec=455993143 SEQUENCE_NUM=0 PID=6947038 UID=0 UID_LOGIN=0 GID=0 PROG_NAME=cat RC_FROM_EVPROD=1000 END_EVENT_INFO	BEGIN_EVENT_INFO TIME_tvsec=1269863383 TIME_tvnsec=455993143 SEQUENCE_NUM=0 PID=6947038 UID=0 UID_LOGIN=0 GID=0 PROG_NAME=cat RC_FROM_EVPROD=1000 BEGIN_EVPROD_INFO event producer message here END_EVPROD_INFO END_EVENT_INFO	BEGIN_EVENT_INFO TIME_tvsec=1269863383 TIME_tvnsec=455993143 SEQUENCE_NUM=0 PID=6947038 UID=0 UID_LOGIN=0 GID=0 PROG_NAME=cat RC_FROM_EVPROD=1000 BEGIN_EVPROD_INFO event producer message here END_EVPROD_INFO STACK_TRACE ahafs_prod_callback+3C4 ahafs_cbfn_wrapper+30 ahafs_vn_write+204 vnop_rdwr+7E4 vno_rw+B4 rwuio+12C rdwr+184 kewrite+16C .svc_instr write+1A4 _xwrite+6C _xflsbuf+B0 __flsbuf+9C copyopt_ascii+2C0 scat+388 main+11C __start+68 END_EVENT_INFO

對於已指定 **AHAFS_THRESHOLD_VALUE_HI** 但未指定 **AHAFS_STKTRACE_AVAILABLE**，而且會將訊息傳遞給事件消費者的事件生產者而言，三種輸出層次如下：

INFO_LVL=1	INFO_LVL=2	INFO_LVL=3
BEGIN_EVENT_INFO TIME_tvsec=1269866715 TIME_tvnsec=16678418 SEQUENCE_NUM=0 CURRENT_VALUE=3 RC_FROM_EVPROD=1000 END_EVENT_INFO	BEGIN_EVENT_INFO TIME_tvsec=1269866715 TIME_tvnsec=16678418 SEQUENCE_NUM=0 CURRENT_VALUE=3 RC_FROM_EVPROD=1000 BEGIN_EVPROD_INFO event producer message here END_EVPROD_INFO END_EVENT_INFO	BEGIN_EVENT_INFO TIME_tvsec=1269866715 TIME_tvnsec=16678418 SEQUENCE_NUM=0 CURRENT_VALUE=3 RC_FROM_EVPROD=1000 BEGIN_EVPROD_INFO event producer message here END_EVPROD_INFO END_EVENT_INFO

錯誤格式

如果事件生產者發生一個錯誤，則所有事件生產者的所有 **INFO_LVL** 都會有下列格式：

```
BEGIN_EVENT_INFO
TIME_tvsec=1269868036
TIME_tvnsec=966708948
SEQUENCE_NUM=0
RC_FROM_EVPROD=20
END_EVENT_INFO
```

如果消費者正在監視 **AHAFS_THRESHOLD_VALUE** 事件，而且現行值已超出所要求的臨界值，則也會使用錯誤格式來記錄此 **EALREADY** 事件：

```
BEGIN_EVENT_INFO
TIME_tvsec=1269868036
TIME_tvnsec=966708948
SEQUENCE_NUM=0
CURRENT_VALUE=1
RC_FROM_EVPROD=56
END_EVENT_INFO
```

BUF_WRAP 及 EVENT_OVERFLOW

如果新事件發生的資料改寫未讀取的資料，則關鍵字 **BUF_WRAP** 會是監視器檔案上 **read()** 的第一個輸出。如果發生緩衝區折返「及」事件溢位，則一律會先出現 **BUF_WRAP**，然後再出現 **EVENT_OVERFLOW**。 這裡是同時發生緩衝區折返及事件溢位時的 **read()** 輸出範例：

第一個 **read()** 會傳回：

```
BUF_WRAP
```

第二個 **read()** 會傳回：

```
EVENT_OVERFLOW
```

第三個 **read()** 會傳回可以放入緩衝區的事件資料：

```
BEGIN_EVENT_INFO
TIME_tvsec=1269863383
TIME_tvnsec=455993143
SEQUENCE_NUM=0
PID=6947038
UID=0
UID_LOGIN=0
GID=0
PROG_NAME=cat
RC_FROM_EVPROD=1000
BEGIN_EVPROD_INFO
event producer message here
END_EVPROD_INFO
STACK_TRACE
ahafs_prod_callback+3C4
ahafs_cbfn_wrapper+30
ahafs_vn_write+204
vnop_rdwr+7E4
vno_rw+B4
rwuio+12C
rdwr+184
kewrite+16C
.svc_instr
write+1A4
_xwri
```

如果事件資訊的出現速度夠快，則可能會連續接收到兩個 **BUF_WRAP** 項目。如果您看到 **BUF_WRAP**，請增加緩衝區大小（寫入監視器檔案時，使用 **BUF_SIZE**）。

NUM_EVDROPS_INTRCNTX

如果任何岔斷環境定義事件出現項目由於事件出現項目的頻率太高而遭捨棄，則對事件檔案的 **read()** 呼叫的輸出（代表該事件），會在包含 BEGIN_EVENT_INFO 關鍵字的那一行後面，包含 NUM_EVDROPS_INTRCNTX 關鍵字。

下列範例代表 **read()** 呼叫的輸出：

```
BEGIN_EVENT_INFO
BEGIN_EVENT_INFO
NUM_EVDROPS_INTRCNTX=5508
TIME0_tvsec=1353437661
TIME0_tvnsec=75494625
TIME_tvsec=1353437661
TIME_tvnsec=741365037
SEQUENCE_NUM=6663
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
...msg from event-producer...
END_EVPROD_INFO
END_EVENT_INFO
```

此範例輸出包含下列資訊集：

- NUM_EVDROPS_INTRCNTX=5508 值是自 TIME0_tvsec 和 TIME0_tvnsec 欄位指定的時間以來，所捨棄的岔斷環境定義事件出現項目數。

· 其餘資訊（亦即，SEQUENCE_NUM=6663、RC_FROM_EVPROD=0、...msg from event-producer... 等等）則與在 TIME_tvsec 和 TIME_tvnsec 欄位指定的時間發生的事件有關。

預先定義的事件生產者

modFile
modFile 事件生產者會監視對檔案內容的修改。

概觀

modFile 事件生產者位於 **fs** 目錄下，並監視對檔案的修改。下列是受監視的 **vnode** 作業：**vnop_rdwr()**、**vnop_map_lloff()**、**vnop_remove()**、**vnop_ftrunc()**、**vnop_fclear()** 及 **vnop_rename()**。無法監視不會通過 LFS 層的修改（即寫入至對映檔）。

如果是下列情況，則可能不會監視檔案：

· 它們位於遠端檔案系統中。

· 它們位於類型為 **ahafs**、**procfs** 或 **namefs** 的檔案系統中。

· 它們是符號鏈結。

· 它們位於結尾為 AIX Event Infrastructure 副檔名（**.mon**、**.list**、**.monFactory**）的目錄下。

· 無法監視 AIX Event Infrastructure 虛擬檔案系統中完整路徑名稱大於 **MAXPATHLEN** 的監視器檔案。

功能

```
AHAFS_THRESHOLD_STATE
AHAFS_STKTRACE_AVAILABLE
AHAFS_REMOTE_EVENT_ENABLED
```

回覆碼

modFile 事件生產者使用在 **<sys/ahafs_evProds.h>** 中定義的回覆碼。

這些回覆碼用來指出如何修改受監視目錄的內容：

AHAFS_MODFILE_WRITE
已寫入受監視檔案。

AHAFS_MODFILE_UNMOUNT
已卸載含有受監視檔案的檔案系統。 這是無法使用的事件。

AHAFS_MODFILE_MAP
處理程序已對映受監視檔案的一部分，以進行寫入。

AHAFS_MODFILE_REMOVE
已移除受監視檔案。這是無法使用的事件。

AHAFS_MODFILE_RENAME
已重新命名受監視檔案。這是無法使用的事件。

AHAFS_MODFILE_FCLEAR
處理程序已針對受監視檔案發出 **fclear**。

AHAFS_MODFILE_FTRUNC
處理程序已針對受監視檔案發出 **ftrunc**。

AHAFS_MODFILE_OVERMOUNT
已過度裝載受監視檔案。

事件生產者訊息

此事件生產者不會傳遞任何訊息，作為其事件資料的一部分。

可接受的監視器檔案

若要監視檔案修改，則應該會在 **modFile.monFactory** 目錄下建立監視器檔案，其路徑與您想要監視的檔案相同。例如，若要監視 **/etc/passwd**，則會使用監視器檔案 **/aha/fs/modFile.monFactory/etc/passwd.mon**。

事件資料範例

下列事件資料是從寫入受監視檔案的處理程序而產生。這是查看到的輸出，且 **INFO_LVL** 為 3：

```
BEGIN_EVENT_INFO
TIME_tvsec=1271703118
TIME_tvnsec=409201093
SEQUENCE_NUM=0
PID=5701678
UID=0
UID_LOGIN=0
GID=0
PROG_NAME=cat
RC_FROM_EVPROD=1000
STACK_TRACE
aha_cbfn_wrapper+30
ahafs_evprods+510
aha_vn_write+154
vnop_rdwr+7E8
vno_rw+B4
rwuio+100
rdwr+188
kewrite+104
.svc_instr
write+1A4
_xwrite+6C
_xflsbuf+A8
__flsbuf+C0
copyopt+2E8
scat+22C
main+11C
__start+68
END_EVENT_INFO
```

modFileAttr

modFileAttr 事件生產者會監視檔案屬性的修改。

概觀

modFileAttr 事件生產者位於 **fs** 目錄下，並監視檔案屬性或目錄屬性的修改（模式、所有權及 ACL）。下列是受監視的 vnode 作業：**vnop_setattr()**（僅適用於 **V_OWN** 及 **V_MODE** 作業）**vnop_setacl()**、**vnop_setxacl()**、**vnop_remove()**、**vnop_rename()** 及 **vnop_rmdir()**。

如果是下列情況，則可能不會監視檔案或目錄：

- 它們位於遠端檔案系統中
- 它們位於類型為 **ahafs**、**procfs** 或 **namefs** 的檔案系統中
- 它們位於結尾為 AIX Event Infrastructure 副檔名（**.mon**、**.list**、**.monFactory**）的目錄下
- 無法監視 AIX Event Infrastructure 虛擬檔案系統中完整路徑名稱大於 **MAXPATHLEN** 的監視器檔案。

功能

```
AHAFS_THRESHOLD_STATE

AHAFS_STKTRACE_AVAILABLE

AHAFS_REMOTE_EVENT_ENABLED
```

回覆碼

modFileAttr 事件生產者使用在 **<sys/ahafs_evProds.h>** 中定義的回覆碼。

這些回覆碼用來指出如何修改受監視目錄的內容：

AHAFS_MODFILEATTR_UNMOUNT
未裝載包含受監視檔案或目錄的檔案系統。這是無法使用的事件。

AHAFS_MODFILEATTR_REMOVE
已移除受監視檔案或目錄。這是無法使用的事件。

AHAFS_MODFILEATTR_RENAME
已重新命名受監視檔案或目錄。這是無法使用的事件。

AHAFS_MODFILEATTR_OVERMOUNT
已過度裝載受監視檔案或目錄。這是無法使用的事件。

AHAFS_MODFILEATTR_SETACL
已修改受監視檔案或目錄的 ACL。

AHAFS_MODFILEATTR_SETOWN
已修改受監視檔案或目錄的所有權。

AHAFS_MODFILEATTR_SETMODE
已修改受監視檔案或目錄的模式。

事件生產者訊息

此事件生產者不會傳遞任何訊息，作為其事件資料的一部分。

可接受的監視器檔案

若要監視檔案修改，則應該在 **modFileAttr.monFactory** 目錄下建立監視器檔案，其路徑與您想要監視的檔案相同。例如，若要監視 **/etc/passwd**，則會使用監視器檔案 **/aha/fs/modFileAttr.monFactory/etc/passwd.mon**。

事件資料範例

下列事件資料是從變更受監視檔案模式的處理程序而產生。這是查看到的輸出，且 **INFO_LVL** 為 3：

```
BEGIN_EVENT_INFO
TIME_tvsec=1291994430
TIME_tvnsec=760097298
SEQUENCE_NUM=0
PID=5767216
UID=0
UID_LOGIN=0
GID=0
PROG_NAME=chmod
RC_FROM_EVPROD=1010
STACK_TRACE
ahafs_evprods+70C
aha_process_attr+120
vnop_setattr+21C
vsetattr@AF13_1+20
setnameattr+B4
chmod+110
.svc_instr
change+3C8
main+190
__start+68
END_EVENT_INFO
```

modDir
modDir 事件生產者會監視對目錄內容的修改。

概觀

modDir 事件生產者位於 **fs** 目錄下，並監視對目錄內容的修改。下列是受監視的 **vnode** 作業：**vnop_create()**、**vnop_link()**、**vnop_symlink()**、**vnop_remove()**、**vnop_rename()**、**vnop_mkdir()** 及 **vnop_rmdir()**。

如果是下列情況，則可能不會監視目錄：

- 它們位於遠端檔案系統中
- 它們位於類型為 **ahafs**、**procfs** 或 **namefs** 的檔案系統中
- 它們是符號鏈結
- 它們位於結尾為 AIX Event Infrastructure 副檔名（**.mon**、**.list**、**.monFactory**）的目錄下
- 無法監視 AIX Event Infrastructure 虛擬檔案系統中完整路徑名稱大於 **MAXPATHLEN** 的監視器檔案。

modDir 事件生產者不會遞迴地監視目錄修改。只會監視對所指定目錄的修改。

功能

```
AHAFS_THRESHOLD_STATE
AHAFS_STKTRACE_AVAILABLE
AHAFS_REMOTE_EVENT_ENABLED
```

回覆碼

modDir 事件生產者使用在 **<sys/ahafs_evProds.h>** 中定義的回覆碼。

這些回覆碼用來指出如何修改受監視目錄的內容：

AHAFS_MODDIR_CREATE
已在受監視目錄下建立新的檔案系統物件。

AHAFS_MODDIR_UNMOUNT
已卸載含有受監視目錄的檔案系統。 這是無法使用的事件。

AHAFS_MODDIR_REMOVE
已移除受監視目錄內的檔案系統物件。

AHAFS_MODDIR_REMOVE_SELF
已移除或重新命名受監視目錄本身。這是無法使用的事件。

事件生產者訊息

事件資料中包含觸發事件的檔案系統物件名稱。

可接受的監視器檔案

若要監視對目錄內容的修改，則應該會在 **modDir.monFactory** 目錄下建立監視器檔案，其路徑與您想要監視的目錄相同。例如，若要監視目錄 **/home/cheryl** 是否進行修改，則會使用監視器檔案 **/aha/fs/modDir.monFactory/home/cheryl.mon**。

事件資料範例

下列事件資料是從受監視目錄中所建立的新檔案 **file1** 而產生。這是查看到的輸出，且 **INFO_LVL** 為 3：

```
BEGIN_EVENT_INFO
TIME_tvsec=1271780397
TIME_tvnsec=24369022
SEQUENCE_NUM=0
PID=6095102
UID=0
UID_LOGIN=0
GID=0
PROG_NAME=touch
RC_FROM_EVPROD=1000
BEGIN_EVPROD_INFO
file1
END_EVPROD_INFO
STACK_TRACE
aha_cbfn_wrapper+30
ahafs_evprods+510
aha_process_vnop+138
vnop_create_attr+4AC
openpnp+418
openpath+100
copen+294
kopen+1C
.svc_instr
open+F8
creat64+1C
main+1EC
__start+68
END_EVENT_INFO
```

utilFs
utilFs 事件生產者會監視檔案系統的使用率。

概觀

utilFs 事件生產者會監視檔案系統的使用率（單位為百分比）。它位於 **fs** 目錄下。 目前只有 JFS2 檔案系統才支援 **utilFs** 監視。 在每次寫入、建立及刪除檔案時，都會檢查檔案系統的使用率，以確認它符合

或超出給定的臨界值。可能有一些檔案系統特有的作業會影響檔案系統的使用率，但在下次寫入、建立或刪除檔案之前，**utilFs** 可能無法偵測到它們。在下次寫入、建立或刪除檔案之前，不會通知因檔案物件刪除而超出的臨界值。

無法監視 AHAFS 中監視器檔案路徑名稱大於 **MAXPATHLEN** 的檔案系統。

為了避免大量的事件通知及可能的效能影響，強烈建議監視 **utilFs** 事件，且 **NOTIFY_CNT** 為 1。

功能

```
AHAFS_THRESHOLD_VALUE_HIGH
AHAFS_THRESHOLD_VALUE_LOW
AHAFS_REMOTE_EVENT_ENABLED
```

指定的臨界值必須介於 1 與 100（含）之間。

回覆碼

utilFs 事件生產者使用在 **<sys/ahafs_evProds.h>** 中定義的回覆碼。

這些回覆碼用來指出如何修改受監視目錄的內容：

AHAFS_UTILFS_THRESH_HIT
受監視的檔案系統已達指定的臨界值。

AHAFS_UTILFS_UNMOUNT
已卸載受監視的檔案系統。這是無法使用的事件。

事件生產者訊息

此事件生產者不會傳遞任何訊息，作為其事件資料的一部分。

可接受的監視器檔案

若要監視檔案系統使用率，則應該會在 **utilFs.monFactory** 目錄下建立監視器檔案，其路徑與要監視的檔案系統的裝載點相同。例如，若要監視檔案系統 **/data/fs1**，則會使用監視器檔案 **/aha/fs/utilFs.monFactory/data/fs1.mon**。

事件資料範例

下列的事件資料是來自 **AHAFS_UTILFS_THRESH_HIT** 事件，其 **INFO_LVL** 為 3：

```
BEGIN_EVENT_INFO
TIME_tvsec=1271705858
TIME_tvnsec=704241888
SEQUENCE_NUM=0
CURRENT_VALUE=10
RC_FROM_EVPROD=1000
END_EVENT_INFO
```

waitTmCPU
waitTmCPU 事件生產者會監視可執行執行緒的平均等待時間。

概觀
waitTmCPU 事件生產者會監視在一秒間隔內，等待以取得 CPU 時間的可執行執行緒的平均等待時間（以毫秒測量）。**waitTmCPU** 位於 **cpu** 目錄下。

功能

```
AHAFS_THRESHOLD_VALUE_HIGH
AHAFS_CALLBACK_INTRCNTX
AHAFS_REMOTE_EVENT_ENABLED
```

指定的臨界值必須大於 0。

回覆碼

此事件生產者一律會在事件發生時傳回 0。

事件生產者訊息
此事件生產者不會傳遞任何訊息，作為其事件資料的一部分。

可接受的監視器檔案

若要監視此事件，則應該會使用下列監視器檔案：

```
/aha/cpu/waitTmCPU.monFactory/waitTmCPU.mon
```

可能不會在此目錄中建立任何其他監視器檔案。

事件資料範例

下列的事件資料是來自 **waitTmCPU** 事件，且 **INFO_LVL** 為 3：

```
BEGIN_EVENT_INFO
TIME_tvsec=1271779504
TIME_tvnsec=18056777
SEQUENCE_NUM=0
CURRENT_VALUE=4
RC_FROM_EVPROD=0
END_EVENT_INFO
```

waitersFreePg

waitersFreePg 事件生產者會監視等待可用訊框的執行緒數目。

概觀

waitersFreePg 事件生產者會監視在一秒間隔內，等待可用訊框的執行緒數目。**waitersFreePg** 位於 **mem** 子目錄下。

功能

```
AHAFS_THRESHOLD_VALUE_HIGH
AHAFS_CALLBACK_INTRCNTX
AHAFS_REMOTE_EVENT_ENABLED
```

指定的臨界值必須大於 0。

回覆碼

此事件生產者一律會在事件發生時傳回 0。

事件生產者訊息

此事件生產者不會傳遞任何訊息，作為其事件資料的一部分。

可接受的監視器檔案

若要監視此事件，則應該會使用下列監視器檔案：

```
/aha/mem/waitersFreePg.monFactory/waitersFreePg.mon
```

可能不會在此目錄中建立任何其他監視器檔案。

事件資料範例

下列的事件資料是來自 **waitersFreePg** 事件，且 **INFO_LVL** 為 3：

```
BEGIN_EVENT_INFO
TIME_tvsec=1271779680
TIME_tvnsec=347233732
SEQUENCE_NUM=0
CURRENT_VALUE=19843
RC_FROM_EVPROD=0
END_EVENT_INFO
```

waitTmPgInOut

waitTmPgInOut 事件生產者會監視等待頁面輸入或頁面輸出作業的執行緒的平均等待時間（毫秒）。

概觀

waitTmPgInOut 事件生產者會監視在一秒期間內，等待頁面輸入或頁面輸出作業完成的執行緒的平均等待時間（毫秒）。**waitTmPgInOut** 事件生產者位於 **mem** 目錄下。

功能

```
AHAFS_THRESHOLD_VALUE_HIGH
AHAFS_CALLBACK_INTRCNTX
AHAFS_REMOTE_EVENT_ENABLED
```

指定的臨界值必須大於 0。

回覆碼

此事件生產者一律會在事件發生時傳回 0。

事件生產者訊息

此事件生產者不會傳遞任何訊息，作為其事件資料的一部分。

可接受的監視器檔案

若要監視此事件，則應該會使用下列監視器檔案：

```
/aha/mem/waitTmPgInOut.monFactory/waitTmPgInOut.mon
```

可能不會在此目錄中建立任何其他監視器檔案。

事件資料範例

下列的事件資料是來自 **waitTmPgInOut** 事件，且 **INFO_LVL** 為 3：

```
BEGIN_EVENT_INFO
TIME_tvsec=1271779359
TIME_tvnsec=941699413
SEQUENCE_NUM=0
CURRENT_VALUE=12
RC_FROM_EVPROD=0
END_EVENT_INFO
```

vmo

vmo 事件生產者會監視對 **vmo** 可調整參數的變更。

概觀

vmo 事件生產者位於 **mem** 目錄下，並監視對下列 **vmo** 可調整參數的變更。

註 : vmo 指令是自我記錄指令。下列清單所列出的一些可調整參數可能不受支援。

- **npskill**
- **npswarn**
- **force_relalias_lite**
- **low_ps_handling**
- **maxpin%**（應該作為 **maxpin_pct.mon** 檔案來監視）
- **nokilluid**
- **relalias_percentage**
- **vmm_default_pspa**
- **npsrpgmin**
- **npsrpgmax**
- **npsscrubmin**
- **npsscrubmax**
- **scrubclean**
- **rpgcontrol**
- **rpgclean**
- **vm_modlist_threshold**
- **vmm_fork_policy**

· **lru_poll_interval**

功能

```
AHAFS_THRESHOLD_STATE
AHAFS_STKTRACE_AVAILABLE
AHAFS_REMOTE_EVENT_ENABLED
```

回覆碼

此事件生產者一律會在事件發生時傳回 0。

事件生產者訊息

此事件生產者不會傳遞任何訊息，作為其事件資料的一部分。

可接受的監視器檔案

若要監視上述任何可調整性，則應該使用下列格式的監視器檔案：

```
/aha/mem/vmo.monFactory/<tunable>.mon
```

無法在此目錄下建立未對應至上述事件的檔案。

事件資料範例

下列的事件資料是來自受監視可調整性的修改，且 **INFO_LVL** 為 3。

```
BEGIN_EVENT_INFO
TIME_tvsec=1271770698
TIME_tvnsec=787565808
SEQUENCE_NUM=0
PID=5701808
UID=0
UID_LOGIN=0
GID=0
PROG_NAME=vmo
RC_FROM_EVPROD=0
STACK_TRACE
aha_cbfn_wrapper+30
ahafs_evprods+510
vm_mon_tunable+B0
vm_chk_mod_tun+5CC
_vmgetinfo+53C
vmgetinfo+48
.svc_instr
vmo_write_vmsetkervars+134
vmo_write_dynamic_values+404
main+BC
__start+70
END_EVENT_INFO
```

schedo

此事件生產者會監視對 **schedo** 可調整性的變更。

概觀

目前可能只會監視 **vpm_xvcpus** 可調整性。 此事件生產者會在事件發生時，傳回堆疊追蹤及使用者資訊。此事件生產者位於 **cpu** 目錄下。

功能

```
AHAFS_THRESHOLD_STATE
AHAFS_STKTRACE_AVAILABLE
AHAFS_REMOTE_EVENT_ENABLED
```

回覆碼

此事件生產者一律會在事件發生時傳回 0。

事件生產者訊息

此事件生產者不會傳遞任何訊息，作為其事件資料的一部分。

可接受的監視器檔案

用來監視這個可調整性的監視器檔案是：

```
/aha/cpu/schedo.monFactory/vpm_xvcpus.mon
```

可能不會在此目錄中建立任何其他監視器檔案。

事件資料範例

下列的事件資料是來自 **vpm_xvcpus** 可調整性的修改，且 **INFO_LVL** 為 3：

```
BEGIN_EVENT_INFO
TIME_tvsec=1271771009
TIME_tvnsec=251723285
SEQUENCE_NUM=0
PID=7143474
UID=0
UID_LOGIN=0
GID=0
PROG_NAME=schedo
RC_FROM_EVPROD=0
STACK_TRACE
aha_cbfn_wrapper+30
ahafs_evprods+510
schedtune+394
.svc_instr
schedo_write_schedparams+94
schedo_write_dynamic_values+6F0
main+1B0
__start+68
END_EVENT_INFO
```

pidProcessMon

pidProcessMon 事件生產者會根據 PID 來監視處理程序毀損。

概觀

pidProcessMon 事件生產者位於 **cpu** 目錄下，並根據 PID 來監視處理程序毀損。

功能

```
AHAFS_THRESHOLD_STATE
AHAFS_CALLBACK_INTRCNTX
```

回覆碼

pidProcessMon 事件生產者只會傳回單一回覆碼 0。

事件生產者訊息

此事件生產者會傳遞 **PROCESS_DOWN** 訊息，作為其事件資料的一部分。

可接受的監視器檔案

若要監視處理程序毀損，則應該會在 **pidProcessMon.monFactory** 目錄下建立監視器檔案。必須使用格式為

```
/aha/cpu/pidProcessMon.monFactory/<process_PID>.mon
```

的監視器檔案名稱。

事件資料範例

下列事件資料是由於受監視處理程序的毀損而產生。這是查看到的輸出，具有預設 **INFO_LVL**。

```
BEGIN_EVENT_INFO
TIME_tvsec=1272348759
TIME_tvnsec=379259175
SEQUENCE_NUM=0
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=PROCESS_DOWN
END_EVPROD_INFO
END_EVENT_INFO
```

processMon

processMon 事件生產者會監視處理程序毀損。

概觀

processMon 事件生產者位於 cpu 目錄下，並根據處理程序名稱來監視處理程序毀損。只會監視同名的給定處理程序的母處理程序。 這表示，如果我們有處理程序樹狀結構 **abc (pid 123)->xyz (pid 345)->xyz (pid 567)**，且某人要求監視 **xyz** 處理程序，則實際上會監視 (**pid = 345**)。

功能

```
AHAFS_THRESHOLD_STATE
AHAFS_REMOTE_EVENT_ENABLED
AHAFS_CALLBACK_INTRCNTX
```

回覆碼

processMon 事件生產者只會傳回單一回覆碼 0。

事件生產者訊息

此事件生產者會傳遞 **PROCESS_DOWN** 訊息，作為其事件資料的一部分。

可接受的監視器檔案

若要監視處理程序毀損，則應該會在 **processMon.monFactory** 目錄下建立監視器檔案，其路徑與用來啟動處理程序的路徑相同。 例如，若要監視放在 **/usr/samples/ahafs** 目錄下且名為 **test** 的處理程序，則會使用監視器檔案 **/aha/cpu/processMon.monFactory/usr/samples/ahafs/test.mon**。

事件資料範例

下列事件資料是由於受監視處理程序的毀損而產生。這是查看到的輸出，具有預設 **INFO_LVL**。

```
BEGIN_EVENT_INFO
TIME_tvsec=1272348909
TIME_tvnsec=482502597
SEQUENCE_NUM=0
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=PROCESS_DOWN
END_EVPROD_INFO
END_EVENT_INFO
```

inetsock

`inetsock` 事件生產者會監視「傳輸控制通訊協定 (TCP)」及「使用者資料封包通訊協定 (UDP)」Socket 作業。

概觀

`inetsock` 事件生產者位於網路目錄下，並監視 Socket 作業。

對於 TCP，會監視下列 Socket 作業：

- 建立 Socket
- 連結埠或位址至 Socket
- 接聽 Socket
- 接受並建立 Socket 連線
- 連接至 Socket
- 切斷來自 Socket 的連線
- 關閉 Socket

對於 UDP，會監視下列 Socket 作業：

- 建立 Socket
- 連結埠或位址至 Socket
- 關閉 Socket

功能

```
AHAFS_THRESHOLD_STATE
AHAFS_CALLBACK_INTRCNTX
AHAFS_REMOTE_EVENT_ENABLED
```

事件生產者訊息

此事件生產者會將通訊協定控制區塊及 Socket 中可用的資訊，當作其事件資料的一部分來傳遞。

對於 TCP Socket 作業，會傳遞下列資料：

Socket 作業	**資料**
PRU_ATTACH	共用資訊： · PROG_NAME · SO_FAMILY · SO_PID · SO_PROTO · SO_TYPE · SO_UID
PRU_BIND	共用資訊加上下列項目： · SO_LADDR · SO_LPORT
PRU_LISTEN	共用資訊加上下列項目： · SO_LADDR · SO_LPORT
PRU_ACCEPT	共用資訊加上下列項目： · SO_FADDR · SO_FPORT · SO_LADDR · SO_LPORT
PRU_CONNECT	共用資訊加上下列項目： · SO_FADDR · SO_FPORT · SO_LADDR · SO_LPORT
PRU_DISCONNECT	共用資訊加上下列項目： · SO_FADDR · SO_FPORT · SO_LADDR · SO_LPORT

Socket 作業	資料
PRU_DETACH、 PRU_ABORT	共用資訊加上下列項目： · SO_LADDR · SO_LPORT · SO_FADDR · SO_FPORT

對於 UDP Socket 作業，會傳遞下列資料：

Socket 作業	資料
PRU_ATTACH	共用資訊： · PROG_NAME · SO_FAMILY · SO_PID · SO_PROTO · SO_TYPE · SO_UID
PRU_BIND、 PRU_DYNBIND	共用資訊加上下列項目： · SO_LADDR · SO_LPORT
PRU_DETACH、 PRU_ABORT	共用資訊加上下列項目： · SO_LADDR · SO_LPORT

可接受的監視器檔案

若要監視 Socket 作業，必須在 `inetsock.monFactory` 目錄中建立一個監視檔案，其具有您要監視之 Socket 作業的名稱。例如，若要監視 TCP Socket 建立作業，則會使用 `/aha/net/inetsock.monFactory/streamCreate.mon` 監視檔案。相同地，若要監視 UDP Socket 建立作業，則會使用 `/aha/net/inetsock.monFactory/dgramCreate.mon` 監視檔案。

下列檔案用於所有 Autonomic Health Advisor File System (AHAFS) 可監視 TCP Socket 作業：

- `/aha/net/inetsock.monFactory/streamCreate.mon`
- `/aha/net/inetsock.monFactory/streamBind.mon`
- `/aha/net/inetsock.monFactory/streamListen.mon`
- `/aha/net/inetsock.monFactory/streamAccept.mon`
- `/aha/net/inetsock.monFactory/streamConnect.mon`
- `/aha/net/inetsock.monFactory/streamDisconnect.mon`
- `/aha/net/inetsock.monFactory/streamClose.mon`

下列檔案用於所有 AHAFS 可監視 UDP Socket 作業：

- `/aha/net/inetsock.monFactory/dgramCreate.mon`
- `/aha/net/inetsock.monFactory/dgramBind.mon`
- `/aha/net/inetsock.monFactory/dgramClose.mon`

事件資料範例

下列事件資料是在建立 Socket 時從處理程序產生。下列範例是與輸出層次 2 (INFO_LVL=2) 一起顯示的輸出：

```
AHAFS event: /aha/net/inetsock.monFactory/streamCreate.mon
---------------------------------------------------

BEGIN_EVENT_INFO
Time          : Mon Jan 23 23:04:06 2012
Sequence Num: 1
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
PROG_NAME=xmtopas
SO_FAMILY=2
SO_TYPE=1
SO_PROTO=6
SO_UID=0
SO_PID=5243048
END_EVPROD_INFO
END_EVENT_INFO
```

叢集事件

系統是叢集的一部分時，它可以在屬於相同叢集的其他節點上發生事件時接收到通知。指定 **AHAFS_REMOTE_EVENT_ENABLED** 功能的事件生產者，支援整個叢集的監視。所有 AIX Event Infrastructure 事件生產者（不含 **pidProcessMon** 及 **diskState**）都可以提供這類遠端通知。

mkcluster 指令搭配使用 AIX Event Infrastructure 的行為：

如果未在系統上載入 AIX Event Infrastructure，而且已執行 **mkcluster** 指令，則 AIX Event Infrastructure 虛擬檔案系統會裝載於 **/aha** 目錄下，因此監視器檔名會從 **/aha** 目錄開始。如果 AIX Event Infrastructure 已載入至系統，而且已執行 **mkcluster** 指令，則不會重新裝載 AIX Event Infrastructure 虛擬檔案系統，因此監視器檔名會從已裝載 AIX Event Infrastructure 虛擬檔案系統的目錄開始。消費者應用程式必須檢查已裝載 AIX Event Infrastructure 虛擬檔案系統的位置，以取得監視器檔案路徑。

為了接收叢集事件，消費者處理程序在寫入監視器檔案，以呈現要在叢集中監視之事件時，必須指定 **CLUSTER=YES**。為了偵測到遠端事件，消費者處理程序必須監視每個已指定 **CLUSTER=YES** 的節點上的事件。

接收自遠端節點的事件不包括使用者或處理程序資訊或堆疊追蹤，即使事件生產者支援也是一樣。

針對遠端節點上接收到的事件，並不提供堆疊追蹤，即使事件生產者支援也是一樣。

叢集資訊 **NODE_NUMBER**、**NODE_ID** 及 **CLUSTER_ID** 將可以在所有叢集事件的 **BEGIN_EVPROD_INFO** 及 **END_EVPROD_INFO** 定界字元之間使用。這可協助監視程式識別發生事件的節點。從 **lscluster -m** 指令輸出的欄位（節點的叢集縮寫 ID、節點的 **uuid** 及叢集 **uuids**）所傳回的資訊，會分別傳回 AIX Event Infrastructure 事件輸出中的 **NODE_NUMBER**、**NODE_ID**、**CLUSTER_ID** 欄位。

下面的輸出範例是來自本端及遠端發生的事件且 **INFO_LVL** 為 2，以及指定 **AHAFS_STKTRACE_AVAILABLE** 功能的事件生產者。

本端事件資料	遠端事件資料
BEGIN_EVENT_INFO TIME_tvsec=1262670289 TIME_tvnsec=453840229 SEQUENCE_NUM=0 PID=4194474 UID=0 UID_LOGIN=0 GID=0 PROG_NAME=rpc.statd RC_FROM_EVPROD=0 BEGIN_EVPROD_INFO NODE_NUMBER=1 NODE_ID=0xF079E8C801C11DF CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404 EVENT_TYPE=PROCESS_DOWN END_EVPROD_INFO END_EVENT_INFO	BEGIN_EVENT_INFO TIME_tvsec=1262670289 TIME_tvnsec=248144872 SEQUENCE_NUM=0 RC_FROM_EVPROD=0 BEGIN_EVPROD_INFO EVENT_TYPE=PROCESS_DOWN NODE_NUMBER=1 NODE_ID=0xF079E8C801C11DF CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404 END_EVPROD_INFO END_EVENT_INFO

支援叢集功能的 AIX 實例的預先定義事件生產者

只有在系統是作用中叢集的一部分時，才能使用這些事件生產者。

nodeList

nodeList 事件生產者會監視叢集成員資格的變更。

概觀

nodeList 事件生產者位於叢集目錄下，並監視從叢集新增或移除的節點。只有在系統是叢集的一部分時，才能使用此事件生產者。從叢集新增或移除節點（例如，透過 **chcluster** 指令）時，會產生此事件。

功能

```
AHAFS_THRESHOLD_STATE
AHAFS_REMOTE_EVENT_ENABLED
AHAFS_CALLBACK_INTRCNTX
```

回覆碼

nodeList 會傳回 0 作為回覆碼。只有在移除叢集時，才會傳回 **AHAFS_CLUSTER_REMOVE (-1)**。

事件生產者訊息

此事件生產者會傳遞 **NODE_ADD** 及 **NODE_DELETE** 訊息，作為其事件資料的一部分。同時，因為它是叢集事件生產者，所以還會另外傳遞 **NODE_NUMBER**、**NODE_ID** 及 **CLUSTER_ID** 資訊。

可接受的監視器檔案

若要監視節點清單的變更，則應該會在 **nodeList.monFactory** 目錄下建立監視器檔案。必須使用監視器檔案名稱

```
/aha/cluster/nodeList.monFactory/nodeListEvent.mon
```

。可能不會在此目錄中建立任何其他監視器檔案。

事件資料範例

下列的事件資料是來自 **nodeList** 事件且具有預設 **INFO_LVL**。

```
BEGIN_EVENT_INFO
TIME_tvsec=1271922590
TIME_tvnsec=886742634
SEQUENCE_NUM=1
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=NODE_ADD
NODE_NUMBER=1
NODE_ID=0xF079E8C801C11DF
CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404
END_EVPROD_INFO
END_EVENT_INFO
```

clDiskList
clDiskList 事件生產者會監視叢集成員資格的變更。

概觀
clDiskList 事件生產者位於磁碟目錄下，並監視從叢集新增或移除的磁碟。只有在系統是叢集的一部分時，才能使用此事件生產者。從叢集新增或移除磁碟（例如，使用 **chcluster** 指令）時，會產生此事件。

功能

```
AHAFS_THRESHOLD_STATE
AHAFS_REMOTE_EVENT_ENABLED
AHAFS_CALLBACK_INTRCNTX
```

回覆碼
clDiskList 會傳回 0 作為回覆碼。只有在移除叢集時，才會傳回 **AHAFS_CLUSTER_REMOVE (-1)**。

事件生產者訊息
此事件生產者會傳遞 **DISK_ADD** 及 **DISK_DELETE** 訊息，作為其在 **EVENT_TYPE** 欄位中的事件資料的一部分。它會傳遞相關磁碟的 **DISK_NAME** 及 **DISK_UID**。同時，因為它是叢集事件生產者的一部分，所以還會另外傳遞 **NODE_NUMBER**、**NODE_ID** 及 **CLUSTER_ID** 資訊。

可接受的監視器檔案
若要監視磁碟清單的變更，則應該會在 **clDiskList.monFactory** 目錄下建立監視器檔案。必須使用監視器檔案名稱

```
/aha/disk/clDiskList.monFactory/clDiskListEvent.mon
```

。可能不會在此目錄中建立任何其他監視器檔案。

事件資料範例

下列的事件資料是來自 **clDiskList** 事件且具有預設 **INFO_LVL**。

```
BEGIN_EVENT_INFO
TIME_tvsec=1271927983
TIME_tvnsec=696543410
SEQUENCE_NUM=0
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=DISK_ADD
DISK_NAME=cldisk1
DISK_UID=3E213600A0B800016726C000000FF4B8677C80F1724-100 FAStT03IBMfcp
NODE_NUMBER=2
NODE_ID=0xF079E8C801C11DF
CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404
END_EVPROD_INFO
END_EVENT_INFO
```

linkedCl
叢集鏈結至另一個叢集，或解除叢集與另一個叢集的鏈結時，會產生 **linkedCl** 事件生產者。

概觀
linkedCl 事件生產者位於叢集目錄下，並監視鏈結狀態變更。只有在系統是叢集的一部分時，才能使用此事件生產者。叢集鏈結至另一個叢集，或解除叢集與另一個叢集的鏈結時，會產生此事件。

功能

```
AHAFS_THRESHOLD_STATE
AHAFS_REMOTE_EVENT_ENABLED
AHAFS_CALLBACK_INTRCNTX
```

回覆碼
linkedCl 會傳回 0 作為回覆碼。只有在移除叢集時，才會傳回 **AHAFS_CLUSTER_REMOVE (-1)**。

事件生產者訊息
此事件生產者會傳遞 **LINK_UP** 或 **LINK_DOWN** 訊息，作為其事件資料的一部分。它會傳遞 **LINK_ID** 資訊。 同時，因為它是叢集事件生產者，所以還會另外傳遞 **NODE_NUMBER**、**NODE_ID** 及 **CLUSTER_ID** 資訊。

可接受的監視器檔案

若要監視節點清單的變更，則應該會在 **linkedCl.monFactory** 目錄下建立監視器檔案。必須使用監視器檔案名稱

```
/aha/cluster/linkedCl.monFactory/linkedClEvent.mon
```

。可能不會在此目錄中建立任何其他監視器檔案。

事件資料範例

下列的事件資料是來自 **linkedCl** 事件且具有預設 **INFO_LVL**。

```
BEGIN_EVENT_INFO
TIME_tvsec=1271224025
TIME_tvnsec=795042625
SEQUENCE_NUM=0
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=LINK_DOWN
LINK_ID=0x7BE9C1BD
NODE_NUMBER=1
NODE_ID=0xF079E8C801C11DF
CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404
END_EVPROD_INFO
END_EVENT_INFO
```

nodeContact

nodeContact 事件生產者會監視叢集中節點的最後一個聯絡狀態。

概觀

nodeContact 事件生產者位於叢集目錄下，並監視叢集中節點的最後一個聯絡狀態。只有在系統是叢集的一部分時，才能使用此事件生產者。

功能

```
AHAFS_THRESHOLD_STATE
AHAFS_REMOTE_EVENT_ENABLED
AHAFS_CALLBACK_INTRCNTX
```

回覆碼

nodeContact 會傳回 0 作為回覆碼。只有在移除叢集時，才會傳回 **AHAFS_CLUSTER_REMOVE (-1)**。

事件生產者訊息

此事件生產者會傳遞 **CONNECT_UP** 及 **CONNECT_DOWN** 訊息，作為其事件資料的一部分。它會傳遞相關的 **INTERFACE_NAME**。同時，因為它是叢集事件生產者，所以還會另外傳遞 **NODE_NUMBER**、**NODE_ID** 及 **CLUSTER_ID** 資訊。

可接受的監視器檔案

若要監視節點清單的變更，則應該會在 **nodeContact.monFactory** 目錄下建立監視器檔案。必須使用監視器檔案名稱

```
/aha/cluster/nodeContact.monFactory/nodeContactEvent.mon
```

。可能不會在此目錄中建立任何其他監視器檔案。

事件資料範例

下列的事件資料是來自 **nodeContact** 事件且具有預設 **INFO_LVL**。

```
BEGIN_EVENT_INFO
TIME_tvsec=1271921874
TIME_tvnsec=666770128
SEQUENCE_NUM=0
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=CONNECT_DOWN
INTERFACE_NAME=en0
NODE_NUMBER=2
NODE_ID=0xF079E8C801C11DF
CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404
```

```
END_EVPROD_INFO
END_EVENT_INFO
```

nodeState

nodeState 事件生產者會監視叢集中節點的狀態。

概觀

nodeState 事件生產者位於叢集目錄下，並監視叢集中節點的狀態。只有在系統是叢集的一部分時，才能使用此事件生產者。例如，節點損毀或關閉時，會產生此事件。

功能

```
AHAFS_THRESHOLD_STATE
AHAFS_REMOTE_EVENT_ENABLED
AHAFS_CALLBACK_INTRCNTX
```

回覆碼

nodeState 會傳回 0 作為回覆碼。只有在移除叢集時，才會傳回 **AHAFS_CLUSTER_REMOVE (-1)**。

事件生產者訊息

此事件生產者會傳遞 **NODE_UP** 及 **NODE_DOWN** 訊息，作為其事件資料的一部分。同時，因為它是叢集事件生產者，所以還會另外傳遞 **NODE_NUMBER**、**NODE_ID** 及 **CLUSTER_ID** 資訊。

可接受的監視器檔案

若要監視節點狀態的變更，則應該會在 **nodeState.monFactory** 目錄下建立監視器檔案。 必須使用監視器檔案名稱

```
/aha/cluster/nodeState.monFactory/nodeStateEvent.mon
```

。可能不會在此目錄中建立任何其他監視器檔案。

事件資料範例

下列的事件資料是來自 **nodeState** 事件且具有預設 **INFO_LVL**。

```
BEGIN_EVENT_INFO
TIME_tvsec=1271921536
TIME_tvnsec=68254861
SEQUENCE_NUM=1
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=NODE_UP
NODE_NUMBER=2
NODE_ID=0xF079E8C801C11DF
CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404
END_EVPROD_INFO
END_EVENT_INFO
```

nodeAddress

nodeAddress 事件生產者會監視節點的網路位址。

概觀

nodeAddress 事件生產者位於叢集目錄下，並監視節點的網路位址。只有在系統是叢集的一部分時，才能使用此事件生產者。例如，從網路介面新增或移除別名時，會產生此事件。

功能

```
AHAFS_THRESHOLD_STATE
AHAFS_REMOTE_EVENT_ENABLED
AHAFS_CALLBACK_INTRCNTX
```

回覆碼

nodeAddress 會傳回 0 作為回覆碼。只有在移除叢集時，才會傳回 **AHAFS_CLUSTER_REMOVE (-1)**。

事件生產者訊息

此事件生產者會傳遞 **ADDRESS_ADD** 及 **ADDRESS_DELETE** 訊息，作為其事件資料的一部分。它會傳遞相關介面的 **INTERFACE_NAME**，以及傳遞 IP 位址的 **FAMILY**、**ADDRESS** 及 **NETMASK**。同時，因為它是叢集事件生產者，所以還會另外傳遞 **NODE_NUMBER**、**NODE_ID** 及 **CLUSTER_ID** 資訊。

可接受的監視器檔案

若要監視節點清單的變更，則應該會在 **nodeAddress.monFactory** 目錄下建立監視器檔案。必須使用監視器檔案名稱

```
/aha/cluster/nodeAddress.monFactory/nodeAddressEvent.mon
```

。可能不會在此目錄中建立任何其他監視器檔案。

事件資料範例

下列的事件資料是來自 **nodeAddress** 事件且具有預設 **INFO_LVL**。

```
BEGIN_EVENT_INFO
TIME_tvsec=1271922254
TIME_tvnsec=9053410
SEQUENCE_NUM=0
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=ADDRESS_ADD
INTERFACE_NAME=et0
FAMILY=2
ADDRESS=0x0A0A0A0A
NETMASK=0xFF000000
NODE_NUMBER=2
NODE_ID=0xF079E8C801C11DF
CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404
END_EVPROD_INFO
END_EVENT_INFO
```

networkAdapterState

networkAdapterState 事件生產者會監視叢集中節點的網路介面。

概觀

networkAdapterState 事件生產者位於叢集目錄下，並監視叢集中節點的網路介面。只有在系統是叢集的一部分時，才能使用此事件生產者。網路介面關閉或啟動時，都會產生此事件。

功能

```
AHAFS_THRESHOLD_STATE
AHAFS_REMOTE_EVENT_ENABLED
AHAFS_CALLBACK_INTRCNTX
```

回覆碼

networkAdapterState 會傳回 0 作為回覆碼。只有在移除叢集時，才會傳回 **AHAFS_CLUSTER_REMOVE (-1)**。

事件生產者訊息

此事件生產者會傳遞 **ADAPTER_UP**、**ADAPTER_DOWN**、**ADAPTER_ADD** 及 **ADAPTER_DEL** 訊息，作為其事件資料的一部分。它會傳遞相關的 **INTERFACE_NAME**。同時，因為它是叢集事件生產者，所以還會另外傳遞 **NODE_NUMBER**、**NODE_ID** 及 **CLUSTER_ID** 資訊。

可接受的監視器檔案

若要監視節點清單的變更，則應該會在 **networkAdapterState.monFactory** 目錄下建立監視器檔案。必須使用監視器檔案名稱

```
/aha/cluster/networkAdapterState.monFactory/networkAdapterStateEvent.mon
```

。可能不會在此目錄中建立任何其他監視器檔案。

事件資料範例

下列的事件資料是來自 **networkAdapterState** 事件且具有預設 **INFO_LVL**。

```
BEGIN_EVENT_INFO
TIME_tvsec=1271920539
TIME_tvnsec=399378269
SEQUENCE_NUM=1
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=ADAPTER_UP
INTERFACE_NAME=en0
```

```
NODE_NUMBER=2
NODE_ID=0xF079E8C801C11DF
CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404
END_EVPROD_INFO
END_EVENT_INFO
```

clDiskState

clDiskState 事件生產者會監視叢集磁碟。

概觀

clDiskState 事件生產者位於磁碟目錄下，並監視叢集磁碟。只有在系統是叢集的一部分時，才能使用此事件生產者。叢集磁碟關閉或啟動時，都會產生此事件。

功能

```
AHAFS_THRESHOLD_STATE
AHAFS_REMOTE_EVENT_ENABLED
AHAFS_CALLBACK_INTRCNTX
```

回覆碼

clDiskState 會傳回 0 作為回覆碼。只有在移除叢集時，才會傳回 **AHAFS_CLUSTER_REMOVE (-1)**。

事件生產者訊息

此事件生產者會傳遞 **DISK_UP** 及 **DISK_DOWN** 訊息，作為其在 **EVENT_TYPE** 欄位中的事件資料的一部分，以及相關叢集磁碟名稱。同時，因為它是叢集事件生產者，所以還會另外傳遞 **NODE_NUMBER**、**NODE_ID** 及 **CLUSTER_ID** 資訊。

可接受的監視器檔案

若要監視叢集磁碟，則應該會在 **clDiskState.monFactory** 目錄下建立監視器檔案。必須使用監視器檔案名稱

```
/aha/disk/clDiskState.monFactory/clDiskStateEvent.mon
```

。可能不會在此目錄中建立任何其他監視器檔案。

事件資料範例

下列的事件資料是來自 **clDiskState** 事件且具有預設 **INFO_LVL**。

```
BEGIN_EVENT_INFO
TIME_tvsec=1271935734
TIME_tvnsec=265210314
SEQUENCE_NUM=1
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=DISK_DOWN
DISK_NAME=cldisk1
NODE_NUMBER=2
NODE_ID=0xF079E8C801C11DF
CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404
END_EVPROD_INFO
END_EVENT_INFO
```

repDiskState

repDiskState 事件生產者會監視儲存庫磁碟。

概觀

repDiskState 事件生產者位於磁碟目錄下，並監視儲存庫磁碟。只有在系統是叢集的一部分時，才能使用此事件生產者。儲存庫磁碟關閉或啟動時，都會產生此事件。

功能

```
AHAFS_THRESHOLD_STATE
AHAFS_REMOTE_EVENT_ENABLED
AHAFS_CALLBACK_INTRCNTX
```

回覆碼

repDiskState 會傳回 0 作為回覆碼。只有在移除叢集時，才會傳回 **AHAFS_CLUSTER_REMOVE (-1)**。

事件生產者訊息

此事件生產者會傳遞 **REP_UP** 及 **REP_DOWN** 訊息，作為其在 **EVENT_TYPE** 欄位中的事件資料的一部分，以及相關儲存庫磁碟的磁碟名稱。同時，因為它是叢集事件生產者，所以還會另外傳遞 **NODE_NUMBER**、**NODE_ID** 及 **CLUSTER_ID** 資訊。

可接受的監視器檔案

若要監視儲存庫磁碟，則應該會在 **repDiskState.monFactory** 目錄下建立監視器檔案。必須使用監視器檔案名稱

```
/aha/disk/ repDiskState.monFactory/repDiskStateEvent.mon
```

。可能不會在此目錄中建立任何其他監視器檔案。

事件資料範例

下列的事件資料是來自 **repDiskState** 事件且具有預設 **INFO_LVL**。

```
BEGIN_EVENT_INFO
TIME_tvsec=1271933757
TIME_tvnsec=134003703
SEQUENCE_NUM=1
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=REP_UP
DISK_NAME=hdisk2
NODE_NUMBER=2
NODE_ID=0xF079E8C801C11DF
CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404
END_EVPROD_INFO
END_EVENT_INFO
```

diskState

diskstate 事件生產者會監視本端磁碟變更。

概觀

diskState 事件生產者位於磁碟目錄下，並監視本端磁碟變更。只有在系統是叢集的一部分時，才能使用此事件生產者。本端磁碟關閉或啟動時，都會產生此事件。只有對儲存體架構所支援的磁碟，才會通知此事件。

功能

```
AHAFS_THRESHOLD_STATE
AHAFS_CALLBACK_INTRCNTX
```

回覆碼

diskState 會傳回 0 作為回覆碼。只有在移除叢集時，才會傳回 **AHAFS_CLUSTER_REMOVE** (-1)。

事件生產者訊息

此事件生產者會傳遞 **LOCAL_UP** 及 **LOCAL_DOWN** 訊息以及相關本端磁碟名稱，作為其事件資料的一部分。 同時，因為是叢集事件生產者，所以還會另外傳遞 **NODE_NUMBER**、**NODE_ID** 及 **CLUSTER_ID** 資訊。

可接受的監視器檔案

若要監視本端磁碟，則應該會在 **diskState.monFactory** 目錄下建立監視器檔案。必須使用格式為

```
/aha/disk/diskState.monFactory/<hdiskn>.mon
```

的監視器檔案名稱（具有必須監視的本端磁碟名稱）。

事件資料範例

下列的事件資料是來自 **diskState** 事件且具有預設 **INFO_LVL**。

```
BEGIN_EVENT_INFO
TIME_tvsec=1271935029
TIME_tvnsec=958362343
SEQUENCE_NUM=1
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=LOCAL_UP
DISK_NAME=hdisk4
```

```
NODE_NUMBER=2
NODE_ID=0xF079E8C801C11DF
CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404
END_EVPROD_INFO
END_EVENT_INFO
```

vgState

vgstate 事件生產者可以驗證磁碟上 VG 的狀態。

概觀

vgState 事件生產者位於磁碟目錄下。 只有在系統是叢集的一部分時，才能使用此事件生產者。只要發生本端（使用 **diskState** 登錄）或叢集磁碟啟動或關閉事件，就會針對位於該磁碟上的磁區群組，觸發對應的 **VG_UP** 及 **VG_DOWN** 事件。使用此事件生產者，應用程式可以驗證磁碟上含有 LVM 子系統的 VG 的狀態。

功能

```
AHAFS_THRESHOLD_STATE
AHAFS_REMOTE_EVENT_ENABLED
AHAFS_CALLBACK_INTRCNTX
```

回覆碼

vgState 會傳回 0 作為回覆碼。只有在移除叢集時，才會傳回 **AHAFS_CLUSTER_REMOVE** (-1)。

事件生產者訊息

此事件生產者會傳遞 **VG_UP** 及 **VG_DOWN** 訊息，作為其事件資料的一部分。它會傳遞相關的磁碟名稱及磁區群組名稱。同時，因為這是叢集事件生產者，所以還會另外傳遞 **NODE_NUMBER**、**NODE_ID** 及 **CLUSTER_ID** 資訊。

可接受的監視器檔案

若要監視節點清單的變更，則應該會在 **vgState.monFactory** 目錄下建立監視器檔案。必須使用監視器檔案名稱

```
/aha/disk/vgState.monFactory/vgStateEvent.mon
```

。可能不會在此目錄中建立任何其他監視器檔案。

事件資料範例

下列的事件資料是來自 **vgstate** 事件且具有預設 **INFO_LVL**。

```
BEGIN_EVENT_INFO
TIME_tvsec=1271915408
TIME_tvnsec=699408296
SEQUENCE_NUM=0
RC_FROM_EVPROD=0
BEGIN_EVPROD_INFO
EVENT_TYPE=VG_UP
DISK_NAME=hdisk3
VG_NAME=myvg
NODE_NUMBER=2
NODE_ID=0xF079E8C801C11DF
CLUSTER_ID=0x6EA7B08888D811DFB918BEB25635B404
END_EVPROD_INFO
END_EVENT_INFO
```

注意事項

本資訊係針對 IBM 在美國所提供之產品與服務所開發。

在其他國家，IBM 不見得有提供本文件所提及之各項產品、服務或特性。請洽詢當地的 IBM 業務代表，以取得當地目前提供的產品和服務之相關資訊。本文件在提及 IBM 的產品、程式或服務時，不表示或暗示只能使用 IBM 的產品、程式或服務。只要未侵犯 IBM 之智慧財產權，任何功能相當之產品、程式或服務皆可取代 IBM 之產品、程式或服務。不過，任何非 IBM 之產品、程式或服務，使用者必須自行負責作業之評估和驗證責任。

本文件所說明之主題內容，IBM 可能擁有其專利或專利申請案。使用者不得享有本書內容之專利權。 您可以用書面方式來查詢特許權限，來函請寄到：

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785US

若要查詢有關雙位元組字元 (DBCS) 資訊的授權查詢事宜，請聯絡您國家的 IBM 智慧財產部門，或書面提出授權查詢，來函請寄到：

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

International Business Machines Corporation 只依「現況」提供本出版品，不提供任何明示或默示之保證，其中包括且不限於未涉侵權、可商用性或特定目的之適用性的隱含保證。有些司法管轄區在特定交易上，不允許排除明示或暗示的保證，因此，這項聲明不一定適合 貴客戶。

本書中可能會有技術上或排版印刷上的訛誤。因此，IBM 會定期修訂；並將修訂後的內容納入新版中。IBM 隨時會改進及/或變更本出版品所提及的產品及/或程式，不另行通知。

本資訊中任何對非 IBM 網站的敘述僅供參考，IBM 對該網站並不提供任何保證。該網站上的資料，並非本 IBM 產品所用資料的一部份，如因使用該網站而造成損害，其責任由 貴客戶自行負責。

IBM 得以各種 IBM 認為適當的方式使用或散佈由 貴客戶提供的任何資訊，而無需對 貴客戶負責。

如果本程式之獲授權人為了 (i) 在個別建立的程式和其他程式（包括本程式）之間交換資訊，以及 (ii) 相互使用所交換的資訊，因而需要相關的資訊，請洽詢：

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785US

上述資料之取得有其特殊要件，在某些情況下必須付費方得使用。

IBM 基於 IBM 客戶合約、IBM 國際程式授權合約或雙方之任何同等合約的條款，提供本文件所提及的授權程式與其所有適用的授權資料。

引用的效能資料與客戶範例僅供舉例說明之用。實際的效能結果可能會因為特定的配置與運作條件而有差異。

本書所提及之非 IBM 產品資訊，係一由產品的供應商，或其出版的聲明或其他公開管道取得。IBM 並未測試過這些產品，也無法確認這些非 IBM 產品的執行效能、相容性或任何對產品的其他主張是否完全無誤。如果您對非 IBM 產品的性能有任何的疑問，請逕向該產品的供應商查詢。

關於 IBM 未來方針或意向之聲明，僅代表 IBM 的目標與目的，隨時可能變動或撤消，而不另行通知。

本出版品中所顯示的所有 IBM 價格皆為 IBM 的現行建議零售價，隨時可能變更，恕不另行通知。經銷商的價格可能與此不同。

本資訊僅供規劃之用。在所述的產品上市之前，本文之資訊隨時可能更改。

本資訊中包含日常商業活動使用的資料與報告範例。為了盡可能完整地說明，範例中包括了個人、公司行號、品牌以及產品等的名稱。此等名稱皆屬虛構，若與實際的人員或企業有任何雷同之處，純屬巧合。

著作權：

本資訊含有原始語言之範例應用程式，用以說明各作業平台中之程式設計技術。貴客戶可以為了研發、使用、銷售或散布符合範例應用程式所適用的作業平台之應用程式介面的應用程式，以任何形式複製、修改及散布這些範例程式，不必向 IBM 付費。該等範例並未在一切情況下完整測試。因此，IBM 不保證或默示保證這些樣本程式之可靠性、服務性或功能。這些程式範例以「現狀」提供，且無任何保證。 IBM 對因使用這些程式範例而產生的任何損害概不負責。

這些程式範例或是任何衍生著作的每一份副本或任何部份，都必須包括如下所示的版權聲明：

©（貴公司名稱）（年）。

本程式碼之若干部分係衍生自 IBM 公司的範例程式。

© Copyright IBM Corp. _enter the year or years_.

隱私權條款考量

IBM 軟體產品（包括軟體即服務解決方案，下面簡稱「軟體供應項目」）可能會使用 Cookie 或其他技術來收集產品使用情況資訊，以協助改良一般使用者體驗、修正與一般使用者的互動，或用於其他用途。在許多情況下，「軟體供應項目」並不會收集個人識別資訊。本公司的部分「軟體供應項目」可協助讓您收集個人識別資訊。 如果本「軟體供應項目」使用 Cookie 來收集個人識別資訊，以下將規定關於這類供應項目使用 Cookie 的特定資訊。

本「軟體供應項目」不會使用 Cookie 或其他技術來收集個人識別資訊。

如果本「軟體供應項目」所部署的配置向 貴客戶提供了通過 Cookie 及其他技術來收集一般使用者個人可識別資訊的能力， 貴客戶應該自行尋求關於此類資料收集所適用的任何法律建議，其中包括對於注意事項及同意的任何需求。

如需將各種技術（包括 Cookie）用於這些目的的相關資訊，請參閱《IBM 隱私權條款》(http://www.ibm.com/privacy) 和《IBM 線上隱私權聲明》(http://www.ibm.com/privacy/details) 以及 http://www.ibm.com/software/info/product-privacy 中標題為「Cookie、Web 訊號指標及其他技術」和「IBM 軟體產品及軟體即服務 (SaaS) 的隱私權聲明」的章節。

商標

IBM、IBM 標誌及 ibm.com 是 International Business Machines Corp. 在世界許多管轄區註冊的商標或註冊商標。其他產品及服務名稱可能是 IBM 或其他公司的商標。如需最新的 IBM 商標清單，請造訪位於 www.ibm.com/legal/copytrade.shtml 的 Copyright and trademark information 網頁。

Linux 是 Linus Torvalds 在美國及（或）其他國家的註冊商標。

UNIX 是 The Open Group 在美國及（或）其他國家的商標。

Windows 是 Microsoft Corporation 在美國及（或）其他國家/地區的商標。