IRIX® Admin: Disks and Filesystems
（日本語版）

007-2825-009JP

編集協力者
執筆 : Susan Ellis, Steven Levine
イラスト : Dany Galgani
制作 : Glen Traefald

表紙デザイン : Sarah Bolles, Sarah Bolles Design, and Dany Galgani, SGI Technical Publications

本マニュアルの変更点

追加された新機能

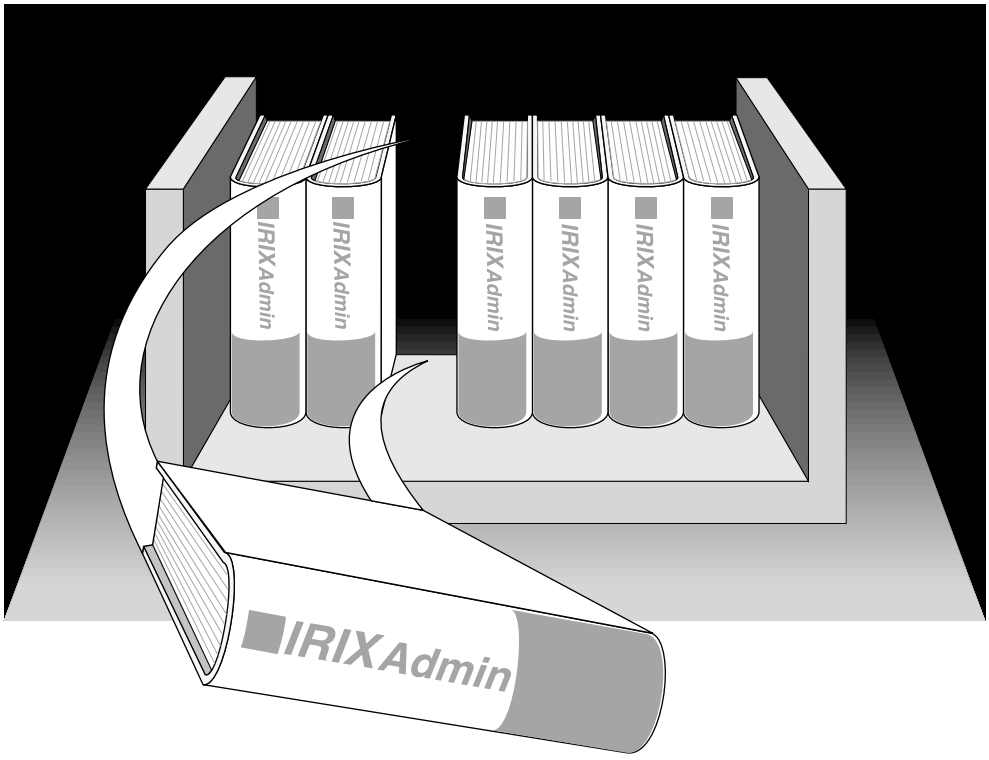
IRIX リリース 6.5.14 では、mkfs コマンドを使って新規作成されるファイルシステムはすべて、デフォルトで XFS version 2 ディレクトリとなります。これに合わせて mkfs の用例が更新されています。

改訂履歴

バージョン	説明
005	1999 年 7 月 IRIX リリース 6.5.5 についての情報を追加
006	1999 年 12 月 IRIX リリース 6.5.7 についての情報を追加
007	2000 年 7 月 IRIX リリース 6.5.9 についての情報を追加
008	2001 年 6 月 IRIX リリース 6.5.13 についての情報を追加
009	2001 年 9 月 IRIX リリース 6.5.14 についての情報を追加

目次

IRIX Admin マニュアル・セット



このマニュアルは、*IRIX Admin* マニュアル・セットの中の 1 冊です。このマニュアルは、サーバ、マルチ・システム、およびファイル構造 (ユーザのホーム・ディレクトリと作業用ディレクトリを除く) を管理するシステム管理者を対象としています。ほかのユーザのためにシステムを管理する場合や、IRIX に関して一般のエンド・ユーザ向けのマニュアルよりさらに専門的な知識が必要な場合はこのマニュアル・セットを参照してください。

IRIX Admin マニュアルは、オンラインの IRIS InSight で参照できます。IRIX Admin マニュアル・セットは、次のマニュアルで構成されています。

- 『IRIX Admin: Software Installation and Licensing』— このマニュアルでは、IRIX で実行するソフトウェアのインストール方法とライセンス管理方法について説明します。IRIX は、SGI 社の UNIX オペレーテイング・システムです。IRIX のインストール・ユーティリティのコマンド行インタフェースである *inst* を使用して、ミニルート・インストールとライブ・インストールを行う手順について説明します。また、IRIX で実行する特定のアプリケーションへのアクセスを制限するライセンス管理製品とそのマニュアルも紹介します。
- 『IRIX Admin: System Configuration and Operation』— このマニュアルでは、標準的なシステム管理方法について説明します。また、システム管理に関する作業として、オペレーティング・システムの設定、ユーザ・アカウント、ユーザ・プロセス、ディスク・リソースの管理、PROM モニタを介したシステムの操作、システム・パフォーマンスについても説明します。
- 『IRIX Admin: Disks and Filesystems』— このマニュアルでは、ディスク、ファイルシステム、および論理ボリュームの概念を説明します。また、SCSI ディスク、XFS™ ファイルシステムと EFS ファイルシステム、XLV 論理ボリューム、および帯域保証 I/O についてのシステム管理手順についても説明します。
- 『IRIX Admin: Networking and Mail』— このマニュアルでは、メール送信、UUCP、SLIP、PPP などを含むネットワーク・システムとメール・システムの計画、設定、使用、管理について説明します。
- 『IRIX Admin: Backup, Security, and Accounting』— このマニュアルでは、ファイルのバックアップとリストア、システムとネットワークのセキュリティ、ユーザ別のシステムの利用記録について説明します。
- 『IRIX Admin: Resource Administration』— システム・リソースの管理に関する基礎知識と、IRIX job limits、Miser Batch Processing System、Cpuset System、および Comprehensive System Accounting (CSA) など IRIX のさまざまなリソース管理機能の使用法を説明します。
- 『IRIX Admin: Peripheral Devices』— このマニュアルでは、端末、モデム、プリンタ、CD-ROM、テープ・ドライブなどの周辺デバイス用のソフトウェアの設定と管理方法について説明します。また、周辺デバイスに付随するケーブルの仕様についても説明します。
- 『IRIX Admin: Selected Reference Pages』— このマニュアルは、InSight では利用できません。このマニュアルは、マン・ページ（マニュアル・ページ）をまとめたものです。システムがダウンしているときに必要となるコマンドについて説明します。各マン・ページでは 1 つのコマンドを説明していますが、関連のある複数のコマンドをまとめて説明したマン・ページもあります。オンラインのマン・ページにアクセスするには、man(1) コマンドを使用します。

このマニュアルについて

『IRIX Admin: Disks and Filesystems』は、IRIX システム管理マニュアルである *IRIX Admin* シリーズの中の 1 冊です。このマニュアルでは、ディスク、ファイルシステム、論理ボリューム、および帯域保証 I/O の管理手順の重要な概念について説明します。

このマニュアルの内容

次に、このマニュアルで説明するディスク、ファイルシステム、および論理ボリュームの種類を示します。

- SCSI ディスク。IRIX 6.2 以降のバージョンのシステムでは、SCSI ディスクのみ使用されます。
- XFS ファイルシステム (eXtent File System)。XFS ファイルシステムは、SGI 社が開発した EFS ファイルシステムに代わる高性能のファイルシステムです。IRIX 5.3 で初めてリリースされました。
- EFS ファイルシステム (Extent File System)。EFS ファイルシステムは、SGI 社が開発したファイルシステムで、長年にわたり IRIX で使用されています。
- XLV 論理ボリューム。XLV 論理ボリューム・システムは、SGI 社が開発した、多数の高度な機能を持つ高性能の論理ボリューム・システムです。IRIX 5.3 で初めてリリースされました。

メモ：このマニュアルは CXFS ファイルシステム、XVM 論理ボリュームの管理については触れていません。CXFS ファイルシステムいては、『CXFS Software Installation and Administration Guide』を、XVM 論理ボリュームについては『XVM Volume Manager Administrator's Guide』を参照してください。

このマニュアルは、参照情報（概念を説明）を提供する章、およびディスクとファイルシステムの管理タスクの手順を提供する章で構成されています。付録では、整合性に欠けるファイルシステムの修復方法の詳細について説明します。このマニュアルは次の章と付録で構成されています。

- 第 1 章「ディスクの概念」では、ディスクの構造、ディスク・パーティションの分割、およびディスク・パーティションのデバイス・ファイルについて説明します。

- 第 2 章「ディスク管理手順の実行」では、ディスクのリスト表示、ディスクの初期化、ボリューム・ヘッダの変更、ディスク・パーティションの再分割、デバイス・ファイルの作成、およびシステムへの新規ディスクの追加など、ディスク管理タスクについて説明します。
- 第 3 章「XLV 論理ボリュームの概念」では、論理ボリュームに関する一般的な概念と XLV 論理ボリュームの仕様について説明します。
- 第 4 章「XLV 論理ボリュームの作成と管理」では、XLV 論理ボリュームの作成と管理、および *lv* 論理ボリュームから XLV 論理ボリュームへの変換を行う管理手順について説明します。lv 論理ボリュームは旧式の論理ボリュームで、現在はサポートされていません。
- 第 5 章「ファイルシステムの概念」では、IRIX ファイルシステムのレイアウト、一般的なファイルシステムの概念、XFS ファイルシステムの詳細について説明します。また、ファイルシステムの作成、マウント、チェック、および拡張の方法についても説明します。
- 第 6 章「ファイルシステムの作成と拡張」では、ファイルシステムの作成、マウント、拡張、および EFS から XFS への変換など、ファイルシステムの管理手順について説明します。
- 第 7 章「ファイルシステムの保守」では、ファイルシステムのチェック、およびディスクの空き領域が不足しているときのディスク使用量の管理など、定期的に、または必要に応じて実行するファイルシステムの管理手順について説明します。
- 第 8 章「帯域保証 I/O のシステム管理」では、帯域保証 I/O に関する情報と、アプリケーションで帯域保証 I/O をサポートするために必要な管理手順について説明します。
- 付録 A「EFS ファイルシステム」では、EFS ファイルシステムとその管理方法について説明します。

表記上の決まり

このマニュアルでは、次の表記法を用いています。

command	この固定ピッチフォントは、文字アイテム（コマンド、ファイル、ルーチン、パス名、シグナル、メッセージ、プログラム言語の構造、Email アドレスなど）およびスクリーンに表示されるアイテムを表します。
variable	斜体文字は、変数名、定義される用語やコンセプトなどを表します。
user input	太字の固定ピッチフォントは、インタラクティブなセッションでユーザが実際に入力する文字を表します。出力結果は太字ではない固定ピッチフォントで示します。
[ ]	カッコで囲まれた部分は、コマンドまたは命令文のオプション部分を表します。
...	省略記号は、繰り返しが可能な前出要素を表します。
manpage(*x*)	コマンド名 *manpage* の後に、カッコ付きでマン・ページのセクション番号が示されます。

ほかのマニュアルへのリンクや、アプリケーションなどの実行可能な語句は赤く表示されます。

本書のほかの章、節、または図などへのリンクは青く表示されます。

このマニュアルで説明する手順が「ツールチェスト (Toolchest)」の「システム (System)」->「ディスク・マネージャ (Disk Manager)」でも実行可能な場合、またはトピックに関する追加情報が『Personal System Administration Guide』に記載されている場合、「アドバイス」で次のように示します。

アドバイス：「システム (System)」の「ディスク・マネージャ (Disk Manager)」を使用し、システムにあるディスクの情報を取得できます。手順については、『Personal System Administration Guide』の 第 3 章「ディスク・マネージャ」を参照してください。

手順が正しく実行されなかったり、経験のあるユーザによって実行されなかったりすると、ファイルが損失する場合があります。そのような場合は、「注意」をその手順の説明の前に記載し、次のように示します。

注意：ここでの手順は、正しく行わないとデータが消去されてしまうことがあります。したがって、この操作は IRIX システム管理者が行うことをお薦めします。

このマニュアルで説明する機能を使用するには、別売のソフトウェアの購入が必要な場合があります。そのような場合は、「注記」に次のように示します。

メモ: オプションのプレックスを使用するための XLV プレックス化機能は、Disk Plexing Option ソフトウェア・オプションを購入してある場合にかぎり使用できます。

このマニュアルの使い方

『IRIX Admin: Disks and Filesystems』は、ディスク、ファイルシステム、および論理ボリュームに対して管理タスクを行う必要があるシステム管理者、およびその他の IRIX の専門知識があるユーザを対象にしたマニュアルです。ここでは、管理タスクを行うためのコマンド行の手順について説明します。これらの管理タスクは多数のディスクを格納している管理サーバとワークステーションに深く関係しています。「ディスク・マネージャ (Disk Manager)」によって提供されるグラフィカル・ユーザ・インタフェースを使用した簡単なディスクとファイルシステムの管理については、『Personal System Administration Guide』で説明します。

IRIX の基礎知識があるユーザであればだれでもこのマニュアルを使用して、ディスクおよびファイルシステムの基本的な管理手順について学習し、実践できます。ただし、いくつかの手順は、正しく実行しないとシステムのファイルを破損する可能性があります。このような手順は次に該当するユーザが行ってください。

- IRIX ファイルシステムの管理手順をよく理解しているユーザ
- *fx* を使用してディスクをパーティションに分割した経験があるユーザ
- *inst* が提供するミニルート環境でシェルを使用して管理タスクを行った経験のあるユーザ
- ファイルシステムのバックアップの概念とその手順をよく理解している（特に *dump* を使用できる）ユーザ

専門知識のある管理者が行う必要のある手順に関しては、説明の前に「注意」と記載してあります。システム管理の詳細については、『IRIX Admin: System Configuration and Operation』を参照してください。

このマニュアルで説明する機能を使用するには、ソフトウェア・オプションを別途購入して FLEX*lm* ライセンスを取得する必要があります。次に、FLEX*lm* ライセンスを必要とする機能を示します。

- XLV 論理ボリュームのプレックス化機能。この機能によってディスクのミラーリングが行われ、最大で 4 つのコピーが作成されます。この機能は Disk Plexing Option ソフトウェア・オプションで提供されます。
- 帯域保証 I/O。帯域保証 I/O (GRIO) は、IRIX が提供する機能です。この機能を使用すると、アプリケーションは固定の I/O 転送速度を要求し、認可されると目的の I/O 転送速度が保証されます。デフォルトでは、システムは最大 4 つの帯域保証 I/O ストリームを許可しています。最大 40 のストリームを取得するには、高性能の帯域保証 I/O（5-40 ストリーム・ソフトウェア・オプション）を購入する必要があります。無限のストリーム数は、高性能の帯域保証 I/O（ストリーム無制限のソフトウェア・オプション）から提供されます。

プロダクト・サポート

日本シリコングラフィックス株式会社は、製品に関する充実したプロダクト・サポートと保守プログラムを提供しています。IRIX およびこのマニュアルに記載したその他の製品に関するサポート・サービスの利用方法の詳細については、IRIX と eoe のリリース・ノートを参照してください。

参考文献

IRIX のディスク管理の詳細については、次を参照してください。

- 『Personal System Administration Guide』。SGI のシステム管理についての基本的な情報を提供します。このマニュアルでは、XFS と XLV に関する情報が改訂されていませんが、多くのシステム管理タスクについての基本的な情報が提供されています。
- 各種ディスク情報と管理コマンドに関するオンライン・マン・ページ。この情報は、標準のシステム・ソフトウェアにあり、*man* および *xman* コマンド、または「ツールチェスト (Toolchest)」の「システム (System)」->「ヘルプ (Help)」を選択し、マン・ページの項目を使用してオンラインで表示できます。
- 『CXFS Software Installation and Administration Guide』は、CXFS ファイルシステムの管理について説明しています。
- 『XVM Volume Manager Administrator's Guide』は、XVM Volume Manage を使用した XVM 論理ボリュームの設定、管理について説明しています。

XFS にアクセスするアプリケーションを開発するための情報については、次のソースを参照してください。

- XFS と GRIO に関係するシステム・コールとライブラリ・ルーチンのオンライン・マン・ページ。これは、IRIS Developer's Option(IDO) ソフトウェアで提供されます。

- 『REACT/Pro Programmer's Guide』では、GRIO を使用するアプリケーションの開発に関する情報を提供します。

ミニルートのロード手順については、『IRIX Admin: Software Installation and Licensing』を参照してください。

ディスクのプレックス化と高性能の帯域保証 I/O ソフトウェア・オプションの使用を許可する NetLS ライセンスの獲得とインストールの詳細については、『IRIX Admin: Software Installation and Licensing』を参照してください。

このマニュアルでドキュメント化されたソフトウェア・リリースの変更の追加情報については、次の製品のリリース・ノートを参照してください。

- IRIX
- eoe
- NFS
- dev

読者からのコメント

本書に記載されている情報の技術的な正確さ、内容、または構成に関してコメントがございましたらお知らせください。その際、マニュアル名とドキュメント番号も併せて明記してください。(ドキュメント番号は、オンライン・ドキュメントの場合はマニュアルの前付部分に、印刷されたマニュアルの場合は裏表紙に記載されています。)

弊社へご連絡の際は、以下の方法をご利用いただけます。

- E-mail

  `techpubs@sgi.com`
- 「Technical Publications Library」Web ページの「フィードバック (Feedback)」オプション

  `http://techpubs.sgi.com`
- 最寄りのカスタマー・サービス代理店。SGI インシデント・トラッキング・システムへのご登録をお申し込みください。
- 郵送

Technical Publications
SGI
1600 Amphitheatre Pkwy., M/S 535
Mountain View, California 94043-1351, USA

- FAX (宛先 : Technical Publications/FAX 番号 : +1 650 932 0801)

皆様からのコメントは重要な参考情報とさせていただき、すみやかに対応いたします。

ディスクの概念

この章では、システムにディスクおよびディスク・デバイス・ファイルを正しく設定するための情報を提供します。

この章では、次の項目について説明します。

ディスク・ドライブをインストールする場合は、ハードウェア付属のインストール・ガイドを参照してください。ディスク管理手順については、第 2 章「ディスク管理手順の実行」で説明します。論理ボリュームとファイルシステムの詳細については、第 3 章「XLV 論理ボリュームの概念」以降の章で説明します。

メモ：XVM Volume Manager を使ったディスク・レイアウトおよびディスク・パーティションについては『XVM Volume Manager Administrator's Guide』を参照してください。

SGI システムのディスク・ドライブ

図 1-1 では、ディスク・ドライブとその他の周辺デバイスがシステムのコントローラにどのように接続されているかを示します。

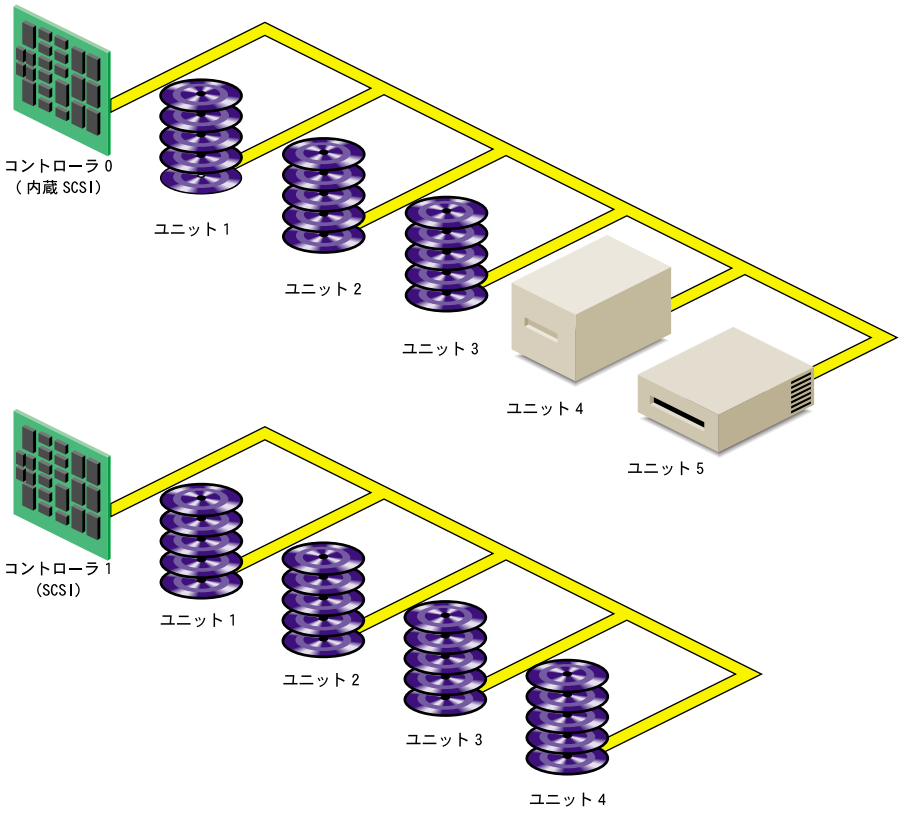


図 1-1 コントローラとディスク・ドライブ

各ディスク・ドライブは、コントローラによって管理されます。各コントローラは、それぞれ決められた数のドライブをサポートしています。使用中のワークステーションがサポートするコントローラの数も決められています。サポートされているコントローラの数およびコントローラのタイプについては、ハードウェアのオーナーズ・ガイドを参照してください。たとえば、SCSI コントローラの場合、SCSI コントローラ・タイプによって、7 ディスクまで、または 15 ディスクまでサポートします。また、VME コントローラは、14 ディスクまでサポートします。

各ディスクには、ドライブ・アドレスが割当てられます。ドライブ・アドレスとは、`hinv` コマンドの出力ではユニット番号と呼ばれ、一般的に SCSI ID と呼ばれているものです。このアドレスは、ディスクのスイッチ、ダイアル、またはジャンパによって、あるいはディスクの物理的な位置によって設定されます。ディスクのドライブ・アドレスの設定については、システム付属のハードウェアのオーナーズ・ガイドを参照してください。

RAID（ディスクアレイ）など、論理ユニット番号または *lun* と呼ばれる追加の識別番号を持つ SCSI デバイスもあります。この識別番号は、デバイス内のディスクのアドレスを指定するために使用されます。

ディスクの物理的な構造

図 1-2 に、ディスクの物理的な構造を示します。ディスクはプラッタと呼ばれる円盤から構成されています。各プラッタの上下の面をサーフェスと呼び、一般的に酸化物でコーティングされています。各サーフェスには少なくとも 1 つの記録ヘッドがあり、プラッタの中心からさまざまな位置へ移動できるアームに取付けられています。プラッタが回転しているときには、ヘッドはプラッタからごくわずか浮いた状態にあり、決してプラッタとは接触することなくデータの読み書きを行います。

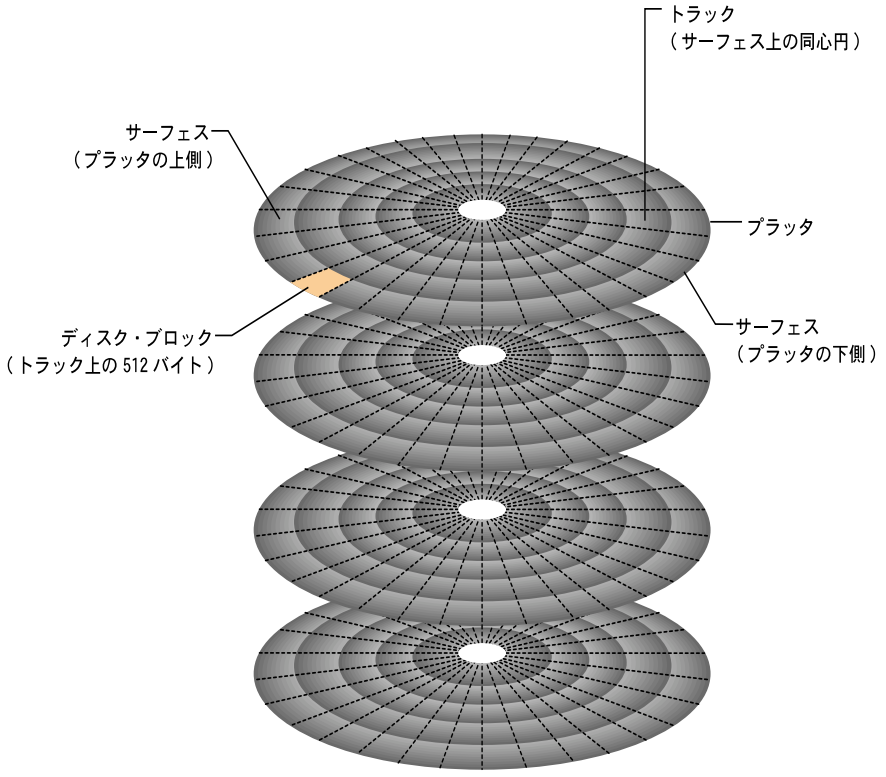


図 1-2 ディスクの物理的な構造

図 1-2 に示すように、サーフェス上のリングはトラックと呼ばれます。各トラックはディスク・ブロックに分かれています。このディスク・ブロックは、セクタと呼ばれることもあります。なお、ディスク上の物理ブロックは、ファイルシステム・ブロックとは異なります。

ディスクをフォーマットすると、ディスクがトラックとディスク・ブロックに分割され、ディスク・コントローラによりアドレス指定が可能になります。また、タイミング・マークの書込み、ディスクの不良領域（不良ブロック）の確認も可能になります。SCSI ディスク・ドライブは、すでにフォーマットされています。したがって、フォーマットする必要はありません。不良ブロック処理は、SCSI ディスクによって自動的に実行されます。不良ブロックとは、データを確実には格納できないディスク領域です。不良ブロック処理では、不良ブロックをマッピングして、通常の IRIX コマンドでは用いることのない予約領域のブロックと置換えます。

ディスク・パーティション

ディスクは、パーティションと呼ばれる論理的な単位に分割されます。図 1-3 に、パーティションに分割されたディスクの例を示します。パーティションによって、ディスクは固定サイズに分割され、IRIX またはユーザが目的に応じて使用できるようになります。パーティションのサイズは、512 バイトのディスク・ブロックで分割されます。SCSI ディスクのパーティションでは、ディスク・ブロック数は必ず整数である必要があります。

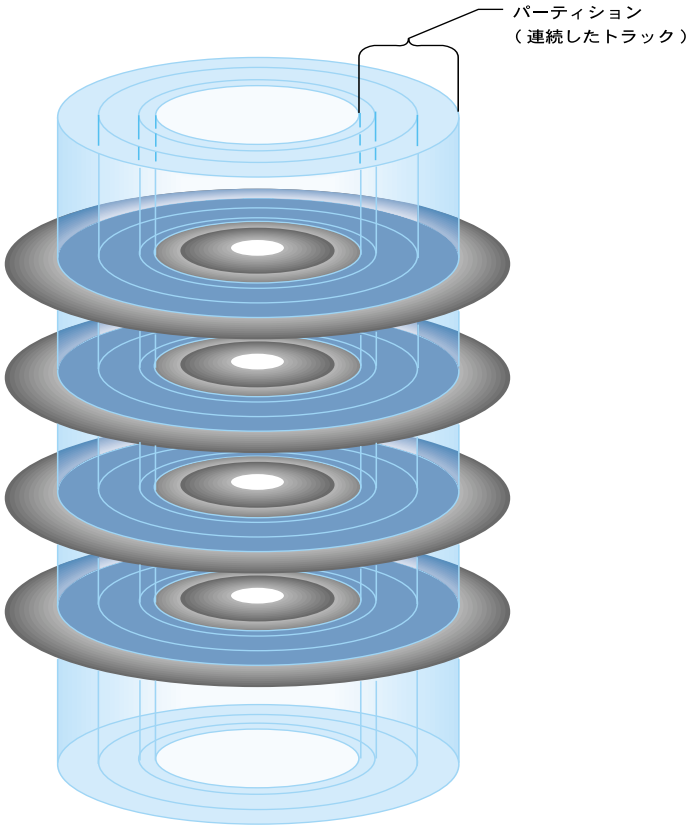


図 1-3　ディスク・パーティション

各ディスク・ブロックは複数のパーティションに属することもできます。また、パーティションに全く属さない場合もありますが、この場合ディスク領域は未使用、または無駄になります。複数のパーティションに属するとは、それらのパーティションがオーバーラップするということで

す。たとえば、ディスクをオーバーラップしないパーティションに分割し、さらにそのディスク全体を別のパーティションとして定義できます。

ディスクの各パーティションは、0 ～ 15 までの番号が付いています。これらのパーティション番号には、特定の機能と名前を持つものがあります。表 1-1 に、これらの番号、名前、および機能を示します。

表 1-1 標準のパーティション番号、パーティション名、およびその機能

パーティション番号	パーティション名	機能
0	root	システム・ディスクの root ファイルシステムに使用される root パーティション。
1	swap	プロセスに必要な物理メモリが足りないときに、IRIX によって使用される一時記憶領域のスワップ・パーティション。
6	usr	root ファイルシステムと usr ファイルシステムが別々に使用されているときに、システム・ディスクに使用される usr パーティション。
7	なし	ボリューム・ヘッダと xfslog パーティションを除くディスク全体。
8	volhdr	ボリューム・ヘッダ。「ボリューム・ヘッダ」を参照。
9	なし	予約パーティション。非 SCSI ドライブでは、このパーティションは不良ブロックのパーティションでした。
10	volume	ボリューム・ヘッダを含むディスク全体。
15	xfslog	XFS ログに使用される小さなパーティション。「パーティションのタイプ」を参照。

システム・ディスク、オプション・ディスク、およびパーティション・レイアウト

IRIX オペレーティング・システムは、システム・ディスクにあります。システム・ディスクには、必ずボリューム・ヘッダがあり、これには *sash*（「ボリューム・ヘッダ」を参照）、root ファイルシステム、スワップ・パーティション、および場合によっては usr ファイルシステムがあります。各ワークステーションまたはサーバには、システム・ディスクが 1 つあります。システムの電源を入れると、このシステム・ディスクから IRIX が起動します。ワークステーションでは、デフォルトで、システム・ディスクはコントローラ番号が 0、ドライブ・アドレスが 1 に設定されています。サーバでは、デフォルトでシステム・ディスクはコントローラ番号が 1、ドライブ・アドレスが 1 に設定されているものもあります。システム・ディスクの位置は *nvram* コマンドによって OSLoadPartition の値で通知されます。

システム・ディスク以外のディスクはすべてオプション・ディスクです。SGI からの出荷時に、ディスクはいくつかの標準パーティション・レイアウトの 1 つに設定されています（詳細はこの節で説明されます）。ディスク・パーティションを表示するには *prtvtoc* コマンドを使用します。第 2 章の「prtvtoc によるディスク・パーティションの表示」を参照してください。

メモ：XVM Volume Manager を使ってディスクに XVM 論理ボリュームを作成する場合、初めにラベルを XVM ディスクとして設定すると、後は XVM Volume Manager がそのディスクのパーティションをコントロールします。XVM でのパーティション・レイアウトについての詳細は『XVM Volume Manager Administrator's Guide』を参照してください。

図 1-4 および 図 1-5 では、root ファイルシステムと usr ファイルシステムに別々のパーティションを持つシステム・ディスクの共通レイアウトを示します。図 1-4 で示すレイアウトは、XFS ログが独自のパーティション（内部 XFS ログ）を持たない場合、EFS ファイルシステムと XFS ファイルシステムで使用されます。図 1-5 に、XFS ログ・パーティション（外部ログ）がある場合のパーティション・レイアウトを示します。

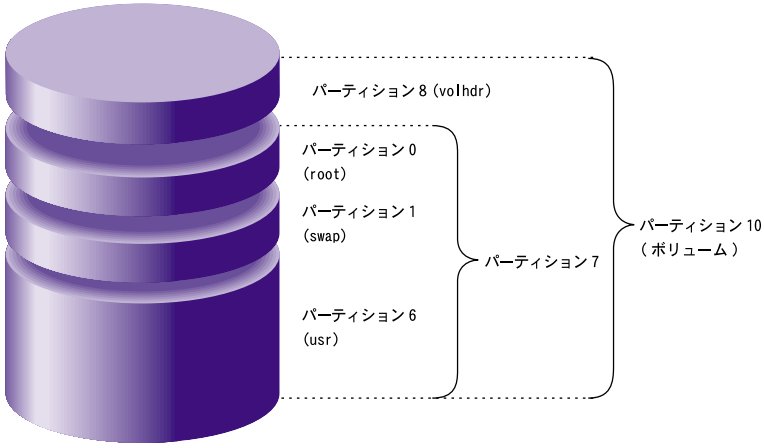


図 1-4　別々の Root と Usr を持つシステム・ディスクのパーティション・レイアウト

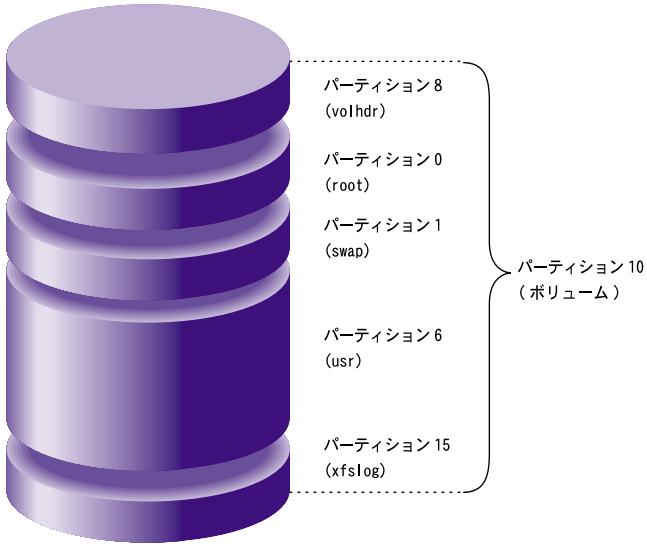


図 1-5 別々の Root と Usr および XFS ログ・パーティションを持つシステム・ディスクのパーティション・レイアウト

過去のシステムでは、root と usr 用に別のパーティションを持つのが標準的であり、サーバでは今でもこの方式が使用されています。当初の UNIX 設計では、UNIX の起動には、root ファイルシステムをマウントするだけでしたが、IRIX の場合は、両方のファイルシステムをマウントする必要があります。したがって、現在では、root ファイルシステムがオペレーティング・システム・ソフトウェアの唯一のファイルシステムとはいえなくなっています。

図 1-6 に、root および usr ファイルシステムの結合に対する単一のパーティションと スワップ・パーティションを持つシステム・ディスクのレイアウトを示します。この配置は、最新のシステムでは標準的です。ただし、XLV 論理ボリュームの root パーティション部の作成には制限があるので、単一の結合されたパーティションよりも別々の root パーティションと usr パーティションを作成することをお薦めします。XLV 論理ボリュームの制限の詳細については、第 3 章「XLV 論理ボリュームの概念」を参照してください。

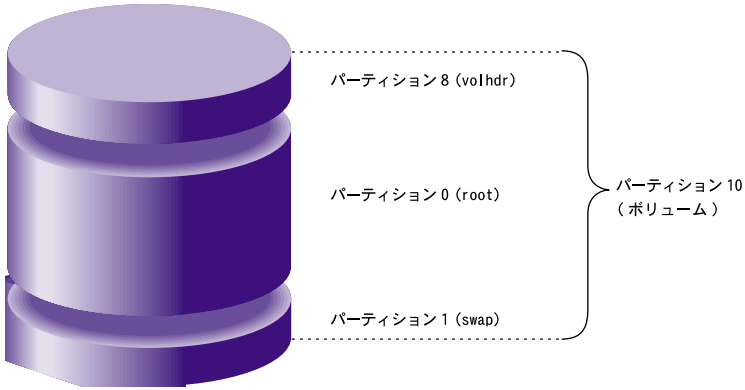


図 1-6 Root と Usr が結合されているシステム・ディスクのパーティション・レイアウト

図 1-7 に、XFS ログ・パーティションを持たないオプション・ディスクの標準レイアウトを示します。オプション・ディスクには、データ用に単一のパーティションがあります。

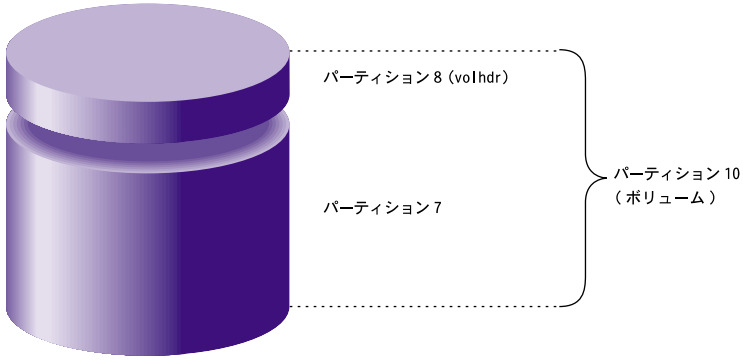


図 1-7 オプション・ディスクのパーティション・レイアウト

図 1-8 に、2 つのパーティションを持つオプション・ディスクのレイアウトを示します。1 つはデータ用で、もう 1 つは XFS ログ用です。

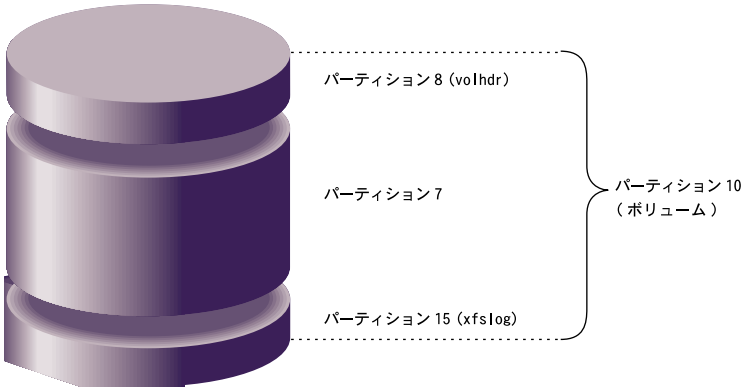


図 1-8 XLV ログ・サブボリュームを持つオプション・ディスクのパーティション・レイアウト

デフォルトのパーティション・レイアウトは、汎用的なもので、システム管理者による評価が必要です。システムを数か月間使用した後、ユーザのニーズに合ったパーティション・レイアウトを決定してください。次に、パーティション・レイアウトを決定するときに考慮すべき点を示します。

- ファイルシステムよりも大きいファイルは使用できません。
- ディスクをパーティションに分割していくつかのファイルシステムに割当てると、ファイルを書込む runaway プロセスはディスク全体にではなく、パーティションのみに書込みます。
- root に領域の大きいパーティションを割当てておくと、将来サイズの大きな IRIX システム・ソフトウェアをインストールする際に root ファイルシステムのディスク領域が不足することはありません。

ディスク・パーティションを変更（ディスクの再分割）するには、fx コマンドを使用してください。*fx* コマンドは標準パーティション・レイアウトで使用でき、また、このコマンドを使用して、カスタム・パーティション・レイアウトを作成できます。*fx* コマンドを使用してディスク・パーティションを再分割する方法の詳細については、第 2 章の「fx によるディスク・パーティションの再分割」を参照してください。

ディスクをパーティションに分割すると、パーティションをファイルシステムや XLV 論理ボリュームの一部、または未使用のディスク領域として使用できます。XLV 論理ボリュームについては、第 3 章「XLV 論理ボリュームの概念」を参照してください。また、ファイルシステムについては、第 5 章「ファイルシステムの概念」を参照してください。

パーティションのタイプ

各パーティションにはタイプがあり、そのタイプは *fx* および *prtvtoc* で確認できます。表 1-2 に、パーティションのタイプ、用途、およびタイプを示すパーティション番号を示します。パーティション 9 は、予約パーティションなのでこの表には示しません。xlv を除くパーティションのタイプは、*fx* により割当てられます。パーティションのタイプ xlv は、XLV 論理ボリューム・コマンドで自動的に割当てられます。

表 1-2 パーティションのタイプと用途

パーティションのタイプ	パーティションの用途	タイプを示すパーティション番号
efs	EFS ファイルシステム	0、6、7（標準パーティション）； 2、3、4、5、11、12、13、14、15（カスタム・パーティション）
xfs	XFS ファイルシステム	0、6、7（標準パーティション）； 2、3、4、5、11、12、13、14、15（カスタム・パーティション）
xfslog	XFS ファイルシステムの外部ログ（XLV ログ・サブボリュームの一部）	15（標準パーティション）； 0、2、3、4、5、6、7、11、12、13、14（カスタム・パーティション）
raw	スワップ領域	1
volhdr	ボリューム・ヘッダ	8
volume	ボリューム・ヘッダを含むボリューム全体	10
xlv	XLV データの一部、またはリアルタイムのサブボリューム	0、1、2、3、4、5、6、7、11、12、13、14、15（パーティションは XLV コマンドにより xlv タイプに変更されます）
lvol	*lv* 論理ボリュームの一部	このパーティションのタイプは現在サポートされていません。*lv* 論理ボリュームは、XLV 論理ボリュームに変換する必要があります。詳細については、第 4 章の「lv 論理ボリュームの XLV 論理ボリュームへの変換」を参照。

表 1-2 に示す標準パーティションは、パーティションを再分割する fx コマンドの機能、rootdrive、usrrootdrive、および optiondrive を使用して作成されます。パーティションのタイプを efs または xfs のどちらかにするかを尋ねるプロンプトが表示されます。また、usrrootdrive または optiondrive に xfs を指定する場合、xfslog パーティションを追加するかどうかを尋ねるプロン

プトが表示されます。xfslog パーティション（外部 XFS ログ）を使用するには、XLV ログ・サブボリュームとして xfslog パーティションを構成する必要があります。XLV の詳細については、第 4 章「XLV 論理ボリュームの作成と管理」を参照してください。xfslog パーティションを使用しない場合は、XFS ログは xfs パーティション（内部ログ）に格納されます。

表 1-2 に示すように、カスタム・パーティションとしてパーティション・タイプをパーティション番号に割当てるには、fx (fx -x) のエキスパート・モードを使用して、パーティションを作成し、そのタイプを割当てる必要があります。*fx* のエキスパート・モードの詳細については、fx(1M) マン・ページを参照してください。

ボリューム・ヘッダ

ボリューム・ヘッダと呼ばれるパーティションは、ディスク・ブロック 0 から始まるパーティションに格納されています。ディスク・ブロック 0 から始まることにより、システム・オペレーションが正しく動作します。ボリューム・ヘッダの基本ファイルシステムには、デバイス・パラメータ、パーティション・レイアウト、一番最後に使用した *fx* のバージョン番号、および論理ボリュームの情報が入った複数のファイルがあります。また、ボリューム・ヘッダには、スタンドアロン・プログラムがある場合もあります。

次に、ボリューム・ヘッダにあるファイルとスタンドアロン・プログラムを示します。

sgilabel このファイルには、*fx* のバージョン番号の情報があります。ボリューム・ヘッダからこのファイルを削除しないでください。

symmon *symmon* は、カーネルをデバッグするためのスタンドアロン・プログラムです。詳細については、symmon(1M) マン・ページを参照してください。

*xlvlab** XLV 論理ボリュームの情報は、ボリューム・ヘッダの論理ボリューム・ラベルと呼ばれるファイルに格納されています。XLV 論理ボリュームの情報は、*xlvlab* で始まる名前のファイルに格納されています。システムの起動時にシステムが XLV 論理ボリュームを認識するために、この情報が使用されます。XLV 論理ボリュームが作成されると、XLV 論理ボリューム・ラベルも自動的に作成されます。

*lvlab** 論理ボリュームの論理ボリューム・ラベルは、*lvlab* で始まる名前のファイルに格納されていました。*lv* 論理ボリュームは現在サポートされていません。

ide *ide* (integrated diagnostics environment) は、ローエンド・システム用の診断プログラムです。「System Maintenance」メニューの 3 番目の項目である「Run Diagnostics」を選択すると *ide* が実行されます。新しいシステムでは、*ide* がボリューム・ヘッダにない場合、*/stand* ディレクトリから、*ide* を実行します。

fx *fx* は、IRIX の *fx* コマンドのスタンドアロン・バージョンです。これは、主にディスク・パーティションを再分割するために使用されるディスク・ユーティリティです。過去のシステムでは、ボリューム・ヘッダに *fx* コマンドのコピーを格納しておくことがありましたが、ボリューム・ヘッダに *fx* コマンドを格納する必要はもうありません。

sash システム・ディスクでは、システムの起動時に必要なスタンドアロン・プログラムである *sash*（スタンドアロン・シェル）のコピーがボリューム・ヘッダにある必要があります。*sash* は、プロセッサ固有のプログラムです。したがって、別のシステムの */stand* ディレクトリまたはソフトウェアのディストリビューション CD の */stand* ディレクトリから *sash* をコピーする必要がある場合は、必ず正しいバージョンをコピーしてください。別のシステムから *sash* をコピーする場合、両システムのプロセッサ・タイプは同じである必要があります。ソフトウェア配布 CD からコピーする場合は、*hinv* コマンドを使用して、システムのプロセッサ・タイプを識別し、表 1-3 からそのシステムに必要な *sash* のバージョンを確認します。

表 1-3 プロセッサ・タイプと *sash* バージョン

プロセッサ・タイプ	***sash* バージョン**
IP17	sashIP17
IP19、IP20、IP22	sashARCS
IP21、IP25、IP26、IP27	sash64

fx コマンドを使用すると、デバイス・パラメータとパーティション・レイアウトを表示および修正できます。fx(1M) マン・ページおよび第 2 章の「fx によるディスク・パーティションの再分割」を参照してください。fx を使用すると、ボリューム・ヘッダに sgilabel ファイルが作成されます。

prtvtoc コマンドを使用すると、パーティション・レイアウト情報が表示されます。詳細については、第 2 章の「prtvtoc によるディスク・パーティションの表示」を参照してください。

dvhtool コマンドを使用すると、ボリューム・ヘッダからスタンドアロン・プログラムを追加、または削除できます。また、ボリューム・ヘッダから XLV 論理ボリューム・ラベルを削除することもできます。詳細については、第 2 章の「dvhtool によるボリューム・ヘッダへのファイルの追加」と 「dvhtool によるボリューム・ヘッダのファイルの削除」を参照してください。

ボリューム・ヘッダは次の場合のみ使用されます。このとき、ボリューム・ヘッダの作成、または修正による間違いが判明します。

- システムの起動時

- ファイルシステムの作成時または拡張時
- 論理ボリュームの作成時または拡張時
- スワップ領域の追加時

デバイス・ファイル

IRIX プログラムは、2 種類の特殊ファイルを使用してハードウェア・デバイスと通信します。2 種類のファイルとは、キャラクタ・デバイス・ファイル（ロー・デバイス・ファイル）とブロック・デバイス・ファイルです。概念上、ディスク・デバイスはファイルとして処理されます。実際には、標準のファイルとデバイス・ファイルは異なるため、デバイス・ファイルは特殊ファイルとして扱われます。

ハードウェア・グラフ対応として書かれたドライバは、ユーザーレベルのコマンドでは変更できない /hw に実際のデバイス・ノードを生成します。これらのノードは /dev デバイス・ノードにリンクしているため、ほとんどのプログラムや管理者が使用する標準のパス名が使えます。ディスク・デバイスはハードウェア・グラフ対応ドライバです。ハードウェア・グラフ対応ではないドライバは、/dev のみ使用します。

メモ： /dev の下にある多くのファイルおよびディレクトリは /hw にリンクしていますが、どのデバイス・ファイルを使用する際も /dev ディレクトリはパスのルートとして推奨されています。ファイルシステムをマウントしている場合、または root ファイルシステムを設定している場合は、/hw の下にデバイス名を使用しないでください。/hw ファイルシステムの詳細については、第 5 章の「/hw ファイルシステム」を参照してください。

デバイス・ファイルは、システム・ソフトウェアのインストール時、またはディスク・ドライブのパーティションの再分割の際に自動的に作成されます。また、必要に応じてシステムの起動時にも作成されます。デバイス・ファイルが自動的に作成されない場合、たとえば擬似デバイスの場合は、MAKEDEV コマンドまたは mknod コマンドを使用できます。詳細については、MAKEDEV(1M) マン・ページと mknod(1M) マン・ページを参照してください。

次の出力例は、ユーザの標準のファイルと */hw* ファイルシステムのディスク・デバイスに *ls -l* コマンドを実行した結果です。この例は、標準のファイルとデバイス・ファイルの構造の違いを示しています。これは標準のファイルの例です。

```
-rw-r----- 1 ralph raccoons 1050   4 月 23 日 08 時 14 分 scheme.notes
```

最初のカラムでダッシュ (-) が付いているのは標準のファイルです。その他の出力例については、『IRIX Admin: System Configuration and Operation』を参照してください。

次は、root のディスク・パーティションの場合のブロック・デバイスおよびキャラクタ・デバイスのデバイス・ファイルです。

```
brw-------    0 root     sys        0, 79   10月 14日 11時15分
/hw/node/io/gio/hpc/scsi_ctlr/0/target/1/lun/0/disk/partition/0/block
crw-------    0 root     sys        0, 80   10月 14日 11時14分
/hw/node/io/gio/hpc/scsi_ctlr/0/target/1/lun/0/disk/partition/0/char
```

次は、*/dev* ディレクトリにあるこれらのデバイス・ファイルへのリンクです。

```
lrw-------    0 root     sys           70   10月 14日 11時12分 /dev/dsk/dks0d1s0 ->
/hw/node/io/gio/hpc/scsi_ctlr/0/target/1/lun/0/disk/partition/0/block
lrw-------    0 root     sys           69   10月 14日 11時13分 /dev/rdsk/dks0d1s0 ->
/hw/node/io/gio/hpc/scsi_ctlr/0/target/1/lun/0/disk/partition/0/char
```

デバイス・ファイルのリストと標準のファイルのリストは似ていますが、デバイス・ファイルには追加情報があります。次に、デバイス・ファイルの特性を示します。

- リストの最初のカラムには、デバイスのタイプを示す *b* (*block*) または *c* (*character*) があります。
- 標準のファイルではバイト数が表示される位置に、デバイス・ファイルではメジャー・デバイス番号とマイナー・デバイス番号という 2 つの数字が表示されます。
- ファイル名は、ハードウェアのタイプと構成に基づいたデバイス名です。

次に、デバイス・ファイルの各特性について説明します。

ブロック・デバイスとキャラクタ・デバイス

ブロック・デバイス・ファイル（ブロック・デバイス）とキャラクタ・デバイス・ファイル（キャラクタ・デバイスまたはロー・デバイス）のアクセス方法は異なります。

ブロック・デバイスは、システムのバッファ・キャッシュから転送したブロック内のデータにアクセスします。特定のサイズのデータブロックのみブロック・デバイスから読取られます。

キャラクタ・デバイスは、キャラクタ・ベースでデータにアクセスします。端末デバイス・ドライバ、擬似端末デバイス・ドライバなど、独自に入出力バッファ処理を行うプログラムでは、キャラクタ・デバイスを使用します。ディスクやテープなど、ハードウェアの種類によっては、キャラクタ・デバイスとブロック・デバイスの両方があるものもあります。ディスク用のキャラクタ・インタフェースは、バッファ・キャッシュをバイパスする点が異なります。

「デバイス名」では、ブロック・デバイス・ファイルとキャラクタ・デバイス・ファイルの命名規則について説明します。

デバイスのパーミッションと所有者

ファイルは、*sys* グループの *root* により所有され、その他のグループまたはユーザはファイルを使用できません。つまり、*root* ID によるプロセスのみデバイス・ファイルに読み書きができます。なお、テープ・デバイス、フロッピー・ドライブ、および tty 端末はこの規則の例外です。

メジャー・デバイスとマイナー・デバイス

標準のファイルのリストでキャラクタ数が表示される位置に、メジャー・デバイス番号とマイナー・デバイスの番号が表示されます。

メジャー・デバイス番号は、特定のデバイス・ドライバを指定します。マイナー・デバイス番号は、特定の物理ユニットとそのユニットの特性を指定します。ディスクのマイナー番号は、ドライブ・アドレスとパーティションを識別します。メジャー番号とマイナー番号は、`ls -l` コマンドで表示されます。

メジャー番号とマイナー番号のペアが同一のデバイスもあります。ただし、1 つのエントリではブロック・デバイス（先頭のカラムは *b*）として指定され、もう 1 つのエントリでは、キャラクタ・デバイス（先頭のカラムは *c*）として指定されます。このようなファイルのペアは、異なるファイル名であるか、または異なるディレクトリにあります（たとえば、`/dev/dsk/dks0d1s0` と `/dev/rdsk/dks0d1s0`）。

デバイス名

ディスクのデバイス名は、ハードウェア（ディスク）のタイプ、デバイス・アクセスのタイプ（ブロックまたはキャラクタ）、デバイスのタイプ、コントローラ番号、ドライブ・アドレス、およびパーティション番号で構成されています。たとえば、SCSI システム・ディスクの root パーティ

ションのブロック・デバイス名は、*/dev/dsk/dks0d1s0* となります。表 1-4 に、このファイル名の各コンポーネントのリストとその用途を説明し、候補値を示します。

表 1-4 デバイス名の構成

デバイス名のコンポーネント	用途	候補値
dev	デバイス・ファイルのディレクトリ	dev
dsk	ハードディスク・ファイルのサブディレクトリ	dsk（ブロック・デバイス・ファイル） rdsk（キャラクタ・デバイス・ファイル。つまり、r はロー (“raw”) を意味し、dsk はキャラクタ・デバイスを意味します）
dks	ディスク・デバイスのタイプ	dks（SCSI デバイス） fd（フロッピー・ディスク） raid（SCSI RAID デバイス）
0	コントローラ番号	0 ～ *n*（*n* がシステムに依存している場合）(SCSI) (SCSI RAID)
d1	ドライブ・アドレス	SCSI 用 : d1 ～ d7 または d1 ～ d15（コントローラのタイプにより異なる） d*n*（*n* が 0 ～ 147 の範囲で 8 または 9 で終わらない場合） (SCSI RAID)
s0	パーティション番号（スライス番号）	s0（root ファイルシステムの root） s1 (swap) s2 s3 s4 s5 s6（usr ファイルシステムの usr） s7（ボリューム・ヘッダを除く使用可能なディスク全体） s8、vh（ボリューム・ヘッダ） s9（非 SCSI 不良ブロックのリスト） s10、vol（ディスク全体） s11 s12 s13 s14 s15（XFS ログ）

次に、いくつかのデバイス名とその意味を説明します。

/dev/dsk/dks0d1s0
: コントローラ 0、ドライブ・アドレス 1 に設定されている SCSI ディスクで、パーティション 0 のブロック・デバイス・ファイルです。

/dev/dsk/jag5d13s7
: コントローラ 5、ドライブ・アドレス 13 に設定されている Jaguar ディスクで、パーティション 7（ボリューム・ヘッダを除くディスク全体）のブロック・デバイス・ファイルです。

/dev/rdsk/dks0d2vh
: コントローラ 0、ドライブ・アドレス 2 に設定されている SCSI ディスクでボリューム・ヘッダ（パーティション 8）のキャラクタ・デバイス （ロー・デバイス）です。

ディスクのデバイス・ファイル名は、システム・ハードウェア・グラフへのシンボリック・リンクです。システム上のハードウェア・エンティティとその関係を記述するこの IRIX 機能の詳細については、第 5 章の「/hw ファイルシステム」を参照してください。

ディスク管理手順の実行

この章では、ディスクとそのデバイス・ファイルの管理手順について説明します。

この章では、次の項目について説明します。

ファイルシステムと XLV 論理ボリュームの管理手順については、第 3 章以降で説明します。

hinv によるシステム上のディスクのリスト表示

IRIX から次の *hinv* コマンドを実行すると、システムに接続されているディスクをリスト表示できます。

```
# hinv -c disk
```

システムのディスク・コントローラとディスクが次のように出力されます。

```
Integral SCSI controller 0: Version WD33C93B, revision D
  Disk drive: unit 2 on SCSI controller 0
  Disk drive: unit 1 on SCSI controller 0
```

この出力は、コントローラ番号 0 でディスク・ドライブが 2 つある内蔵 SCSI コントローラが 1 つあることを示しています。この 2 つのディスクのドライブ・アドレスはそれぞれ 1 と 2 です。*hinv* の出力では、ドライブ・アドレスはユニット、またはユニット番号と呼ばれます。各ディスクは、コントローラ番号とドライブ・アドレスの組合わせによって、一意に識別されます。

PROM モニタの場合は、Command Monitor から *hinv* コマンドを実行することもできます。

```
>> hinv
```

SCSI ディスクの場合、次のように出力されます。

```
SCSI Disk: scsi(0)disk(1)
SCSI Disk: scsi(0)disk(2)
```

この出力では、コントローラ番号は SCSI 番号で、ドライブ・アドレスはディスク番号です。コントローラのタイプは出力されません。一般的には、ワークステーションに内蔵コントローラがあり、サーバに内蔵 SCSI コントローラ、または SCSI か VME の非内蔵コントローラがあります。Challenge システムでは、PROM モニタの *hinv* の出力で、ブート IOP（I/O プロセッサ）のディスクしか表示されない場合もあります。

ディスクのコントローラ番号およびドライブ・アドレスは、さまざまな構文を使用して指定されます。構文には *fx*、*prtvtoc*、*dvhtool*、*mkfs* などの IRIX のディスクおよびファイルシステムのコマンドの引数が使用されます。次に、コントローラ 0、ドライブ・アドレス 1 のディスクの場合の例を示します。

- ディスクを指定するときの *fx* コマンド行は、次のとおりです。

  ```
  # fx "dksc(0,1)"
  ```

- *prtvtoc* コマンド行でディスク（実際には、ディスクのボリューム・ヘッダ）を指定するには、次の 2 つのコマンドのどちらかを使用します。

  ```
  # prtvtoc /dev/rdsk/dks0d1vh
  ```

  ```
  # prtvtoc dks0d1vh
  ```

- *dvhtool* コマンド行でディスク 1（実際には、ディスク 1 のボリューム・ヘッダ）を指定するときのコマンドは、次のとおりです。

  ```
  # dvhtool /dev/rdsk/dks0d1vh
  ```

- XFS に対して、*mkfs* コマンド行で上記の 2 番目のディスク・パーティション 7 を指定するときのコマンドは、次のとおりです。

  ```
  # mkfs /dev/rdsk/dks0d1s7
  ```

アドバイス：「ツールチェスト (Toolchest)」の「システム (System)」->「システム・マネージャ (System Manager)」->「ハードウェアとデバイス (Hardware and Devices)」->「ディスク・マネージャ (Disk Manager)」を使用すると、システム上のディスク情報を取得できます。詳細については、『Personal System Administration Guide』の 第 3 章「ディスク・マネージャ」を参照してください。

fx によるディスクのフォーマットと初期化

ディスクをフォーマットするとき、タイミング・マークを付け、ディスク・コントローラでアドレスを指定できるように、ディスクをトラックとセクタに分割します。出荷時に SCSI ディスクはすでにフォーマットされているので、フォーマットする必要はありません。フォーマットは *fx* で行われます。詳細については、fx(1M) マン・ページを参照してください。

注意：ディスクをフォーマットすると、ディスクのデータがすべて消去されます。したがって、ディスクのフォーマットは、IRIX システム管理者が行うことをお薦めします。

ディスクのフォーマットによって、ディスクの不良個所（不良ブロック）に関する情報が破壊されます。*fx* を使用して不良ブロックの識別や処理もできます。詳細については、fx(1M) マン・ページを参照してください。

注意：不良ブロックの処理に *fx* を使用すると、そのブロックに格納されているデータがすべて消去されます。したがって、この処理は IRIX システム管理者が行うことをお薦めします。

ディスクの初期化では、ディスクにボリューム・ヘッダを作成します。Silicon Graphics 社から提供されているディスクには、すでにボリューム・ヘッダが作成されているので、初期化する必要はありません。それ以外の販売元から提供されるディスク、またはボリューム・ヘッダが破壊されているディスクを使用する場合は、ディスクの初期化を行ってボリューム・ヘッダを作成してください。*fx* によってディスクが初期化されます。コマンドを明示的に実行する必要はありません。*fx* は自動的にボリューム・ヘッダがないことを検出し、ボリューム・ヘッダを作成します。*fx* コマンド呼出しの詳細については、この章の「fx によるディスク・パーティションの再分割」を参照してください。*fx* によってボリューム・ヘッダが作成されるとき、ボリューム・ヘッダを作成するかどうか確認を求めるメッセージが表示されるので、yes と入力してください。

アドバイス : 「ツールチェスト (Toolchest)」の「システム (System)」->「システム・マネージャ (System Manager)」->「ハードウェアとデバイス (Hardware and Devices)」->「ディスク・マネージャ (Disk Manager)」ウィンドウを使用すると、ディスクの初期化とほかのタスクを実行できます。詳細については、『Personal System Administration Guide』の 第 3 章「ディスク・ドライブの管理」を参照してください。

dvhtool によるボリューム・ヘッダへのファイルの追加

第 1 章の「ボリューム・ヘッダ」で説明されたように、システム・ディスクのボリューム・ヘッダには、*sash* プログラムのコピーが必要です。ここでの手順では、*sash* などのプログラムをボリューム・ヘッダに追加する方法について説明します。この手順を実行する前に、第 1 章の「ボリューム・ヘッダ」の *dvhtool* の説明を再度確認してください。

ディスクのボリューム・ヘッダにプログラムを追加するとき、追加するプログラムには、システムの */stand* ディレクトリと IRIX ソフトウェア・リリース CD の */stand* ディレクトリの 2 つのソースがあります。CD の */stand* ディレクトリ (CD のマウント後は、通常 */CDROM/stand*) に、プロセッサ固有の *sash*、*fx*、*ide* のコピーがあります。

特権ユーザとして、次の手順を実行し、ボリューム・ヘッダにプログラムを追加します。

1. たとえば、ディスクのボリューム・ヘッダのロー・デバイス名を引数として、*dvhtool* を起動します。

   ```
   # dvhtool /dev/rdsk/dks0d2vh
   ```

   デバイス名の設定方法の詳細については、第 1 章の「デバイス名」を参照してください。

2. *vd* (ボリューム・ディレクトリ) コマンドと *l* (リスト) コマンドを使用して、ボリューム・ヘッダのボリューム・ディレクトリ部分を表示します。

   ```
   Command? (read, vd, pt, dp, write, bootfile, or quit): vd
   (d FILE, a UNIX_FILE FILE, c UNIX_FILE FILE, g FILE UNIX_FILE or l)?
           l

   Current contents:
           File name         Length     Block #
           sgilabel             512           2
           sash              159232           3
   ```

3. *a* (追加) コマンドを使用して、各プログラムにボリューム・ヘッダをコピーします。たとえば、*/stand* ディレクトリから *sash* をボリューム・ヘッダの *sash* にコピーするには、次のコマンドを使用します。

```
(d FILE, a UNIX_FILE FILE, c UNIX_FILE FILE, g FILE UNIX_FILE or l)?
        a /stand/sash sash
```

また、CD から IP20 システムまたは IP22 システム (Indy™) に *sash* をコピーするには、次のコマンドを使用します。

```
(d FILE, a UNIX_FILE FILE, c UNIX_FILE FILE, g FILE UNIX_FILE or l)?
        a /CDROM/stand/sashARCS sash
```

CD には *sash* のプロセッサ固有のバージョンが複数あります。表 1-3 に、各プロセッサ・タイプの *sash* のバージョンを示します。

4. l（リスト）コマンドでボリュームの内容を表示し、変更内容を確認します。

```
(d FILE, a UNIX_FILE FILE, c UNIX_FILE FILE, g FILE UNIX_FILE or l)?
        l

Current contents:
        File name        Length     Block #
        sgilabel            512           2
        sash             159232           3
```

5. *quit* コマンドでサブメニューを終了し、さらに *write* コマンドを使用して、ボリューム・ヘッダに変更内容を書込み、変更内容を有効にします。

```
(d FILE, a UNIX_FILE FILE, c UNIX_FILE FILE, g FILE UNIX_FILE or l)?
        quit

Command? (read, vd, pt, dp, write, bootfile, or quit): write
Quit dvhtool by giving the quit command:
Command? (read, vd, pt, dp, write, bootfile, or quit): quit
```

dvhtool によるボリューム・ヘッダのファイルの削除

注意：ここでの手順を正しく行わないとデータが消去されてしまうことがあります。したがって、この操作は IRIX システム管理者が行うことをお薦めします。

次の手順を使用すると、XLV 論理ボリューム・ラベル（*xlvlab* など）とファイル（*sash* など）をディスクのボリューム・ヘッダから削除できます。この手順を実行する前に、第 1 章の「ボリューム・ヘッダ」の *dvhtool* の説明を再度確認してください。

1. *hinv* を使用して、変更するボリューム・ヘッダを持つディスクのコントローラとドライブ・アドレスを調べます。この手順の例で示すコマンドと出力では、ディスクがコントローラ 0、ドライブ・アドレス 2 にあります。各自のディスクのコントローラとドライブ・アドレスに置換えてください。
2. 特権ユーザとして、ディスクのボリューム・ヘッダのロー・デバイス名を使用して、*dvhtool* を起動します。

   ```
   # dvhtool /dev/rdsk/dks0d2vh
   ```

   デバイス名の設定方法については、第 1 章の「デバイス名」を参照してください。
3. 以下の 2 つのメッセージに応答して、ボリューム・ヘッダのボリューム・ディレクトリ部分を表示します。

   ```
   Command? (read, vd, pt, dp, write, bootfile, or quit): vd
   (d FILE, a UNIX_FILE FILE, c UNIX_FILE FILE, g FILE UNIX_FILE or l)?
           l

   Current contents:
           File name        Length     Block #
           sgilabel            512           2
           xlvlab            10752           3
           lvlab2              512          26
   ```
4. *d* コマンドを使用して、ファイル（*xlvlab* ファイルなど）を削除します。

   ```
   (d FILE, a UNIX_FILE FILE, c UNIX_FILE FILE, g FILE UNIX_FILE or l)?
           d xlvlab
   ```
5. ほかに削除するファイルがあれば、*d* コマンドを使用してそのファイルを削除します。

   ```
   (d FILE, a UNIX_FILE FILE, c UNIX_FILE FILE, g FILE UNIX_FILE or l)?
           d lvlab2
   ```
6. 再度ボリューム・ディレクトリを表示して、ファイルが削除されたことを確認します。

   ```
   (d FILE, a UNIX_FILE FILE, c UNIX_FILE FILE, g FILE UNIX_FILE or l)?
           l

   Current contents:
           File name        Length     Block #
           sgilabel            512           2
   ```
7. このメニューを終了して、変更内容をボリューム・ヘッダに書込みます。

   ```
   (d FILE, a UNIX_FILE FILE, c UNIX_FILE FILE, g FILE UNIX_FILE or l)?
           q

   Command? (read, vd, pt, dp, write, bootfile, or quit): write
   ```
8. *dvhtool* を終了します。

   ```
   Command? (read, vd, pt, dp, write, bootfile, or quit): quit
   ```

prtvtoc によるディスク・パーティションの表示

prtvtoc コマンドを使用すると、ディスクのサイズとパーティションに関する情報を表示できます。このコマンドを使用できるのは、特権ユーザだけです。次にコマンドを示します。

```
# prtvtoc device
```

device はオプションです。*device* を指定しないで *prtvtoc* を使用すると、システム・ディスクに関する情報が表示されます。*device* は、ディスクのボリューム・ヘッダのロー・デバイスの名前です。デバイス名の */dev/rdsk* は、省略できます。たとえば、コントローラ 0、ドライブ・アドレス 1 の SCSI ディスクの場合は、*device* は dks0d1vh になります。デバイス名の詳細については、第 1 章の「デバイス名」を参照してください。

次に *prtvtoc* の出力例を示します。

```
Printing label for root disk

* /dev/root (bootfile "/unix")
*     512 bytes/sector
Partition  Type  Fs   Start: sec    Size: sec   Mount Directory
 0          xfs  yes        4096      4138249
 1          raw           4142345       262144
 8       volhdr                0         4096
10       volume                0      4404489
```

出力の最後にあるパーティション・テーブルでは、パーティション、パーティションのタイプ（名前またはファイルシステムのタイプ）、ファイルシステムがあるかどうか、ディスク・パーティションの位置（ブロックとシリンダの開始位置とサイズ）、ファイルシステムのマウント・ディレクトリが表示されます。図 1-6 に、この出力のパーティションを示します。

xdkm によるディスク・パーティションの再分割

xdkm コマンドのグラフィック・ユーザ・インタフェースを使用して、ディスクを再分割できます。*xdkm* の詳細については、オンライン・ヘルプを参照してください。

fx によるディスク・パーティションの再分割

注意：ここでの手順は、正しく行わないとデータが消去されてしまう可能性があります。したがって、この操作は IRIX システム管理者が行うことをお薦めします。

ディスク・パーティションの再分割は、*fx* によりコマンド行から実行されます。このプログラムには、スタンドアロンのバージョンと IRIX バージョンの 2 つがあります。スタンドアロン・バージョンは Command Monitor から起動し、システム・ディスク・パーティションを再分割できます。オプション・ディスクは、IRIX バージョンを使用してパーティションを再分割します。

続いて、ディスクを再分割する手順について説明します。まず、第 2 章の「パーティションの再分割の前に」を読んでください。その後、*fx* の起動に関する説明に進んでください。

- 「Command Monitor からの fx の起動」
- 「IRIX からの fx の起動」

第 1 章の「システム・ディスク、オプション・ディスク、およびパーティション・レイアウト」で説明した標準パーティション・レイアウトは、*fx* に組込まれています。標準レイアウトを使用して、ディスクをパーティションに分割することも、カスタム・パーティション・レイアウトを作成することもできます。ここでは、標準パーティション・レイアウトとカスタム・パーティション・レイアウトの作成方法について説明します。

- 「標準パーティション・レイアウトの作成」
- 「カスタム・パーティション・レイアウトの作成」

「パーティションの再分割の後に」（34 ページ）では、パーティションを再分割した後に必要な作業について説明します。

パーティションの再分割の前に

注意 : ディスク・パーティションを再分割すると、ディスクのデータにアクセスできなくなります。ディスクのデータにアクセスするには、元のパーティションに戻してください。

ディスク・パーティションを再分割する前に、重要なデータを含むファイルのバックアップを作成してください。パーティションを再分割した後にバックアップからファイルをシステム・ディスクにコピーする場合は、「システム・マネージャ (System Manager)」または *backup* コマンドを使用してください。*backup* コマンドまたは「システム・マネージャ (System Manager)」で作成されたバックアップのみ「System Maintenance」メニューの「System Recovery」メニューから使用できます。これら 2 つの方法のうち「システム・マネージャ (System Manager)」の使用をお薦めします。詳細については、『Personal System Administration Guide』で説明します。その他のコマンドを使用する場合は、正しく機能するために完全にシステムをインストールする必要があります。

Command Monitor からの fx の起動

ここでは、Command Monitor から *fx* のスタンドアロン・バージョンを起動する方法について説明します。これは、システム・ディスクの場合のみ必要な手順です。ほかのディスクには、*fx* の IRIX バージョンを使用できます。28 ページの「IRIX からの fx の起動」を参照してください。

1. システムを停止して「System Maintenance」メニューにアクセスします。
2. 「System Maintenance」メニューを使って、Command Monitor を起動します。
3. 起動する *fx* のコピーを確認します。コピーの位置は、システム・ディスクの */stand* ディレクトリの *fx*、またはローカル・システム、リモート・システムの CD-ROM ドライブにある IRIX ソフトウェアのディストリビューション CD の *fx* です。

   fx のコピーは */stand* ディレクトリには 1 つしかありませんが、IRIX の提供ソフトウェアの CD には *fx* のプロセッサ固有のバージョンが複数あります。ローカルな CD-ROM ドライブの CD から *fx* を起動するには、その CD にプロセッサ固有の *sash* のコピーが必要です。

   表 2-1 に、プロセッサ・タイプと対応する *sash* バージョンおよび *fx* バージョンを示します。

表 2-1 *sash* と *fx* バージョン

プロセッサ・タイプ	sash バージョン	fx バージョン
IP17	sashIP17	fx.IP17
IP19、IP20、IP22	sashARCS	fx.ARCS
IP21、IP25、IP26、IP27	sash64	fx.64

4. Command Monitor から *fx* を起動します。起動するコマンドは、起動する *fx* のコピーの位置によって異なります。
   - 次のコマンドは、システム・ディスクの */stand* ディレクトリから *fx* を起動します。

     >> **boot stand/fx --x**

   - 次のコマンドは、ローカルな CD-ROM ドライブの IRIX ソフトウェア・リリース CD から *fx* を起動します。このとき、システムの CPU タイプは IP19、IP20、または IP22 で、CD-ROM ドライブはドライブ・アドレス 4、コントローラ 0 です。

     >> **boot -f dksc(0,4,8)sashARCS dksc(0,4,7)stand/fx.ARCS --x**

   - 次のコマンドは、リモート・システム dist の */CDROM* にマウントされている CD-ROM ドライブの IRIX ソフトウェア・リリース CD から *fx* を起動します。このとき、ローカル・システムの CPU タイプは IP21、IP25、IP26、または IP27 です。

     >> **boot -f bootp()dist:/CDROM/stand/fx.64 --x**

5. *fx* は、ディスク名を決定するのに必要な情報を要求します。デフォルトはかっこ内に示されており、システム・ディスクを示す値です。メッセージは、次のとおりです。

```
fx: "device-name" = (dksc)
fx: ctlr# = (0)
fx: drive# = (1)
fx: lun# = (0)
```

デフォルトのデバイス名は *dksc* です。これは SCSI コントローラの SCSI ディスクを示します。その他のデバイス名については、fx(1M) マン・ページを参照してください。次に、ディスクのコントローラ番号およびドライブ・アドレス（ユニット）の指定を求めるメッセージが表示されます。最後に、lun（論理ユニット）番号の指定を求めるメッセージが表示されます。通常、論理ユニット番号は、デバイス内でディスクをアドレス指定するために、RAID（ディスクアレイ）など一部の SCSI デバイスで使用されます。標準ディスクには、論理ユニット番号 0 を使用します。

各メッセージに対して値を入力した後、**<Enter>** キーを押します。ただし、デフォルト値を使用する場合は、**<Enter>** キーだけを押します。

メッセージに対して指定を行うと、*fx* はディスク・ドライブをテストし、*fx* メイン・メニューを表示します。

```
---- please choose one (? for help. .. to quit this menu)----
[exi]t                [d]ebug/                [l]abel/
[b]adblock/           [exe]rcise/             [r]epartition/
fx>
```

exit オプションを指定すると *fx* は終了しますが、ほかのコマンドを指定するとそれぞれのサブメニューが表示されます。メニュー・オプションの後にスラッシュ (/) 文字が付いている場合は、そのオプションを選択すると、サブメニューが表示されることを示しています。すべての *fx* のオプションについては、fx(1M) マン・ページを参照してください。

IRIX からの fx の起動

ここでは、IRIX から *fx* を起動する方法について説明します。

1. まずパーティションに分割するディスク・ドライブが現在使用されていないことを確認します。つまり、ファイルシステムがマウントされていないこと、およびそのドライブにアクセスしているプログラムがないことを確認します。
2. 特権ユーザとして、*fx* コマンドを実行します。

```
# fx "controller_type(controller,address,logical_unit)"
```

使用される変数は、次のとおりです。

controller_type コントローラのタイプ。SCSI コントローラの場合は、dksc です。ほかのコントローラ・タイプについては、fx(1M) マン・ページを参照してください。

controller ディスクのコントローラ番号。

address ディスクのドライブ・アドレス。

logical_unit デバイスの論理ユニット番号。通常、論理ユニット番号は、デバイス内でディスクをアドレス指定するために、RAID（ディスクアレイ）など一部の SCSI デバイスで使用されます。通常、*logical_unit* の値は 0 です。

引数を使用しないで *q* コマンドを指定すると、これらの値の入力を求めるメッセージが表示されます。

fx はまずドライブをテストし、その後、次に示すメニューを表示します。

```
---- please choose one (? for help. .. to quit this menu)----
[exi]t              [d]ebug/              [l]abel/
[b]adblock/         [exe]rcise/           [r]epartition/
fx>
```

exit オプションを指定すると、*fx* は終了しますが、ほかのコマンドを指定すると、それぞれのサブメニューが表示されます。メニュー・オプションの後にスラッシュ (/) 文字が付いている場合は、そのオプションを選択すると、サブメニューが表示されることを示しています。すべての *fx* のオプションについては、fx(1M) マン・ページを参照してください。

標準パーティション・レイアウトの作成

ここでは、標準パーティション・レイアウトを作成するために、ディスク・パーティションを再分割する方法を示します。別々の root パーティションと usr パーティションを持つディスクを 1 つのパーティションに組合わせる例を示します。

1. *fx* メイン・メニューから、*repartition* オプションを選択します。

```
---- please choose one (? for help. .. to quit this menu)----
[exi]t              [d]ebug/              [l]abel/
[b]adblock/         [exe]rcise/           [r]epartition/
fx> repartition

----- partitions-----
part  type        blocks          Megabytes    (base+size)
  0: efs        3024 + 50652         1 + 25
  1: raw       53676 + 81648        26 + 40
  6: efs      135324 + 1925532      66 + 940
  8: volhdr        0 + 3024          0 + 1
 10: volume        0 + 2060856       0 + 1006
```

```
capacity is 2061108 blocks

----- please choose one (? for help, .. to quit this menu)-----
[ro]otdrive           [o]ptiondrive         [e]xpert
[u]srrootdrive        [re]size
```

fx の起動時に指定されたディスクのパーティション・レイアウトと *repartition* メニューが表示されます。*rootdrive*、*usrrootdrive*、*optiondrive* の各オプションは標準パーティション・レイアウトに使用し、*resize* オプションはカスタム・パーティション・レイアウトに使用します。また、*expert* オプションは *fx* の起動時に **-x** オプションを指定した場合にのみ表示され、カスタム・パーティション機能を実行可能にします。カスタム・パーティション機能は、正しく実行しないとディスクが破損する可能性があるため、**-x** オプションで明示的に要求しないかぎり使用できません。

2. root と usr を組合わせたパーティションを作成するには、*rootdrive* オプションを選択します。

   ```
   fx/repartition> rootdrive
   ```

3. パーティションのタイプを指定するメッセージが表示されます。表 1-2 に、指定可能なタイプを示します。この例では、efs を選択します。

   ```
   fx/repartition/rootdrive: type of data partition = (xfs) efs
   ```

4. 次のような警告が表示されます。この警告の後のメッセージに対して yes と入力してください。

```
Warning: you will need to re-install all software and restore user data
from backups after changing the partition layout.  Changing partitions
will cause all data on the drive to be lost.  Be sure you have the drive
backed up if it contains any user data.  Continue? yes

----- partitions-----
part  type        blocks          Megabytes   (base+size)
  0: efs        3024 + 1976184       1 + 965
  1: raw     1979208 + 81648       966 + 40
  8: volhdr        0 + 3024           0 + 1
 10: volume        0 + 2060856        0 + 1006

capacity is 2061108 blocks

----- please choose one (? for help, .. to quit this menu)-----
[ro]otdrive        [u]srrootdrive     [o]ptiondrive      [re]size
```

パーティションを再分割後のパーティション・レイアウトが表示され、*repartition* サブメニューが再度表示されます。

5. *fx* メイン・メニューに戻るには、メッセージに対して **..** と入力します。

   ```
   fx/repartition> ..
   ```

```
----- please choose one (? for help, .. to quit this menu)-----
[exi]t                [d]ebug/              [l]abel/
[b]adblock/           [exe]rcise/           [r]epartition/
fx>
```

カスタム・パーティション・レイアウトの作成

ここでは、カスタム・パーティション・レイアウトを作成するために、ディスク・パーティションを再分割する方法を示します。ここでの手順では 380 MB の SCSI ドライブのパーティションを再分割して、root パーティションのサイズを増やします。

1. *fx* メイン・メニューで、*repartition* オプションを選択します。

```
---- please choose one (? for help. .. to quit this menu)----
[exi]t              [d]ebug/              [l]abel/
[b]adblock/         [exe]rcise/           [r]epartition/
fx> repartition
----- partitions-----
part  type        blocks          Megabytes    (base+size)
  0: efs        2835 + 32400         1 + 16
  1: rawdata   35235 + 81810        17 + 40
  6: efs      117045 + 513945       57 + 251
  7: efs        2835 + 628155        1 + 307
  8: volhdr        0 + 2835          0 + 1
 10: entire        0 + 630990        0 + 308

capacity is 631017 blocks

----- please choose one (? for help, .. to quit this menu)-----
[ro]otdrive        [u]srrootdrive     [o]ptiondrive      [re]size
```

fx の起動時に指定したディスクのパーティション・レイアウトが表示され、*repartition* メニューが表示されます。パーティション 0、1、6 のサイズ・カラムを見てください。この例では、32,400 + 81,810 + 513,945 = 628,155 ブロックを使用できます。開始ブロック番号を見ると、パーティション 7 がパーティション 0、1、6 とオーバラップしていることがわかります。パーティション 0 は、root ファイルシステムで、システムのルート・ディレクトリ (/) にマウントされています。パーティション 1 は、システムのスワップ領域です。パーティション 6 は、usr ファイルシステムで、*/usr* ディレクトリにマウントされています。この例では、usr ファイルシステムの領域を除き、root ファイルシステムの領域を拡張しています。

2. *resize* オプションを選択して、ディスク・パーティションのサイズを変更します。表示された警告メッセージに対して、**y** と入力します。

```
fx/repartition> resize

Warning: you will need to re-install all software and restore user data
```

```
from backups after changing the partition layout.  Changing partitions
will cause all data on the drive to be lost.  Be sure you have the drive
backed up if it contains any user data.  Continue? y

After changing the partition, the other partitions will
be adjusted around it to fit the change.  The result will be
displayed and you will be asked whether it is OK, before the
change is committed to disk.  Only the standard partitions may
be changed with this function.  Type ? at prompts for a list
of possible choices
```

3. 警告メッセージの後のメッセージでは、変更するデフォルトのパーティションとして、スワップ領域のパーティションが表示されますが、この例では、root パーティションを指定します。メッセージに対して **root** と入力します。

```
fx/repartition/resize: partition to change = (swap) root
current:  type efs       base:     2835 blks,    1 Mb
             len:    32400 blks,  16 Mb
```

4. 次に、パーティションに分割する方法 (パーティションのサイズの単位) を指定するようメッセージが表示されます。デフォルトの単位としてメガバイトが表示されます。その他、ディスク領域の比率やディスク・ブロック数も指定できます。ディスクをパーティションに分割する場合、メガバイトとディスク領域の比率を単位として使用する方法が最も簡単です。**<Enter>** キーを押して、パーティションを再分割する方法としてメガバイトを指定します。

```
fx/repartition/resize: partitioning method = (megabytes (2^20 bytes)) Enter
```

5. 次に、root パーティションのサイズをメガバイトで指定するように求めるメッセージが表示されます。デフォルトは、パーティションの現在のサイズです。この例では、サイズを 20 MB に増やします。

```
fx/repartition/resize: size in megabytes (max 307) = (16) 20
----- partitions-----
part  type        blocks           Megabytes   (base+size)
  0: efs        2835 + 40960         1 + 20
  1: rawdata   43795 + 73250        21 + 36
  6: efs      117045 + 513945       57 + 251
  8: volhdr        0 + 2835          0 + 1
 10: entire        0 + 630990        0 + 308
```

新しいパーティション・マップが表示されます。スワップ・パーティションから 4 メガバイトが root パーティションに追加されました。最終的には、usr パーティションからこの 4 メガバイトを追加しますが、ここでは新しいパーティション・レイアウトを使用します。

6. 新しいパーティション・レイアウトを使用するには、メッセージに対して **yes** と入力します。

```
Use the new partition layout? (no) yes
```

新しいパーティション・テーブルが、総ディスク容量とともに再度出力されます。その後「repartition」メニューに戻ります。

7. *resize* を再度選択して、usr パーティションからスワップ領域にディスク領域を転送します。

```
fx/repartition> resize
```

同じ警告メッセージが再度表示されます。

8. パーティションの変更を求めるメッセージで、**<Enter>** キーを押して、スワップ・パーティションのサイズを変更します。

```
fx/repartition/resize: partition to change = (swap) Enter
current:  type raw       base:   43795 blks,   21 Mb
                          len:   73250 blks,   36 Mb
```

9. パーティションを再分割する方法としてメガバイトを使用するには、再度 **<Enter>** キーを押します。

```
fx/repartition/resize: partitioning method = (megabytes (2^20 bytes)) Enter
```

10. 次に、スワップ・パーティションの新しいサイズを求めるメッセージが表示されます。root ファイルシステムに 4 メガバイトを追加して、root ファイルシステムを 16 メガバイトから 20 メガバイトに拡張したので、このメッセージに対して **40** と入力します。その後、**<Enter>** キーを押して、スワップ領域を元のサイズに拡張します。システムのスワップ領域が不足している場合は、もう少し大きな数字を入力して、領域を増やすこともできます。

```
fx/repartition/resize: size in megabytes (max 307) = (36) 40
----- partitions-----
part  type        blocks          Megabytes    (base+size)
  0: efs        2835 + 40960         1 + 20
  1: rawdata   43795 + 81920        21 + 40
  6: efs      125715 + 505275       61 + 247
  8: volhdr        0 + 2835          0 + 1
 10: entire        0 + 630990        0 + 308
```

新しいパーティション・テーブルが表示されます。このパーティション・テーブルでは、パーティション 6 (usr) から 4 メガバイトが取られ、スワップ・パーティションに転送されています。

11. メッセージに対して、新しいパーティション・レイアウトを使用するには、**yes** と入力します。

```
Use the new partition layout? (no) yes
```

新しいパーティション・テーブルと repartition サブメニューが再度表示されます。

12. メッセージに対して **..** と入力して、*fx* メイン・メニューに戻ります。

```
fx/repartition> ..

----- please choose one (? for help, .. to quit this menu)-----
[exi]t              [d]ebug/            [l]abel/
[b]adblock/         [exe]rcise/         [r]epartition/
fx>
```

パーティションの再分割の後に

1. *fx* メイン・メニューから、**exit** と入力して *fx* を終了します。

   ```
   fx> exit
   ```

2. システム・ディスクのパーティションを再分割した場合は、次のどちらかの方法でシステム・ディスクにソフトウェアをインストールしてください。

   - ミニルートを起動し（「System Maintenance」メニューの「Install System Software」を選択）、「Administrative Commands」メニューで *mkfs* コマンドを使用してディスク・パーティションにファイルシステムを作成し、IRIX リリースとオプションのソフトウェアをインストールします。
   - 「System Maintenance」メニューの「System Recovery」を選択し、以前に作成したバックアップまたはシステム・マネージャのバックアップ・テープを使用して、元のファイルをディスクに戻します。

   オプション・ディスクのパーティションを再分割した場合は、*mkfs* コマンドを使用して、ディスク・パーティションに新しいファイルシステムを作成します。

3. 必要に応じて、バックアップ・テープからユーザ・ファイルを復元します。

ln によるデバイス・ファイルのニーモニック名の作成

/dev/dsk/dks0d1s0 または */dev/rdsk/dks0d2s7* のようなデバイス・ファイル名は、覚えにくく入力も面倒です。この問題を解決するのがニーモニック・デバイス名です。これは、*/dev* ディレクトリ内のファイル名で、実際のデバイス・ファイルへのシンボリック・リンクです。デフォルトで、IRIX にはいくつかのニーモニック・デバイス・ファイル名があります。たとえば、*/dev/root* は */dev/dsk/dks0d1s0*（または root ファイルシステムを持つパーティション）のニーモニック・デバイス・ファイル名であり、*/dev/rswap* は */dev/rdsk/dks0d1s1*（またはスワップ・パーティション）のニーモニック・デバイス・ファイル名です。*ln* コマンドを使用すると、ニーモニック・デバイス・ファイル名を新たに作成できます。

```
# ln device_file mnemonic_name
```

ln コマンドの詳細については、ln(1) マン・ページを参照してください。

PROM モニタからのシステム・ディスクの作成

ここでは、現在システム・ディスクが動作していないシステムにシステム・ディスクをインストールする方法について説明します。この操作は、次のような場合に行います。

- 新しいディスクにフォーマット情報やパーティション情報がいっさいない場合、またはパーティションの設定が間違っている場合。
- オプション・ディスクをシステム・ディスクに変換する必要がある場合。

システムに動作中のディスクがある場合は、次の手順ではなく、第 2 章の「IRIX からの新しいシステム・ディスクの作成」で示す手順に従ってください。

ディスクをシステム・ディスクに変換するには、提供されている IRIX システム・ソフトウェア・リリース CD と、システムに接続しているか、またはネットワーク上で使用可能な CD-ROM ドライブが必要です。ネットワーク上のシステムに接続している CD-ROM ドライブを使用する場合は、そのシステムをインストレーション・サーバとして設定してください。手順については、『IRIX Admin: Software Installation and Licensing』を参照してください。

次の手順では、システム・ディスクをコントローラ 0、ドライブ・アドレス 1 にインストールするものとします。この位置はワークステーションの標準的な位置ですが、コントローラ番号はサーバ上のシステムによって異なります。次の手順に従ってください。

1. システムを起動して、「System Maintenance」メニューを表示します。
2. 「System Maintenance」メニューの 5 番目の項目を選択して、Command Monitor を起動します。
3. *hinv* コマンドを実行し、CPU タイプと 表 2-1 を使用して、起動する必要があるスタンドアロン *fx* のバージョンを調べます。たとえば、IP19 プロセッサのシステムは ARCS プロセッサです。したがって、必要となるスタンドアロンの *fx* のバージョンは *stand/fx.ARCS* です。
4. 使用する *fx* のコピーが格納されているデバイスのコントローラとドライブ・アドレスを調べます。このデバイスは、システムに接続している CD-ROM ドライブ、またはネットワーク上のワークステーションに接続している CD-ROM ドライブです。たとえばローカル CD-ROM ドライブの場合、*hinv* によってシステムの CD-ROM ドライブが scsi(0)、cdrom(4) と報告された場合、コントローラは 0 でドライブ・アドレスは 4 です。使用しているデバイスがこれとは異なっていたり、異なるワークステーションに配置されているかもしれませんが、この例ではこのデバイスを使用します。
5. ネットワーク接続を介してインストールしている場合は、CD-ROM ドライブがあるワークステーションの IP アドレスを取得します。
6. IRIX システム・ソフトウェア・リリースが格納されている CD を CD-ROM ドライブに挿入します。
7. Command Monitor のコマンドを実行して、*fx* を起動します。この例の場合、コマンドは次のとおりです。

```
>> boot -f dksc(0,4,8)sashARCS dksc(0,4,7)stand/fx.ARCS --x
72912+9440+3024+331696+23768d+3644+5808 entry: 0x89f9a950
```

```
112784+28720+19296+2817088+59600d+7076+10944 entry: 0x89cd74d0
SGI Version 5.3 ARCS   Oct 18, 1994
```

このワークステーションまたは別のワークステーションの CD-ROM から *fx* を起動するコマンドのリストについては、『IRIX Admin: Software Installation and Licensing』の 付録 A「Inst のまとめ」を参照してください。

8. プロンプトに対して **<Enter>** キーを押します。システム・ディスクが選択されます。

```
fx: “device-name” = (dksc)
fx: ctlr# = (0) Enter
fx: drive# = (1) Enter
fx: lun# = (0)
...opening dksc(0,1,)
...drive selftest...OK
Scsi drive type == SGI     SEAGATE ST31200N8640

----- please choose one (? for help, .. to quit this menu)-----
[exi]t             [d]ebug/           [l]abel/           [a]uto
[b]adblock/        [exe]rcise/        [r]epartition/     [f]ormat
```

9. *repartition* コマンドを実行して、ディスク・パーティションの設定を表示します。

```
fx> repartition

----- partitions-----
part  type        blocks          Megabytes   (base+size)
  7: efs        3048 + 2074164       1 + 1013
  8: volhdr        0 + 3048          0 + 1
 10: volume        0 + 2077212       0 + 1014

capacity is 2077833 blocks
```

パーティション・レイアウトを調べて、ディスク・パーティションを再分割する必要があるかどうかを確認します。標準パーティション・レイアウトについては、第 1 章の「システム・ディスク、オプション・ディスク、およびパーティション・レイアウト」を参照してください。

10. ディスク・パーティションの再分割が不要な場合は、手順 13 に進んでください。
11. ディスクのパーティション・レイアウトを選択します。システム・ディスクの標準パーティション・レイアウト（第 1 章の「システム・ディスク、オプション・ディスク、およびパーティション・レイアウト」を参照）またはカスタム・パーティション・レイアウトを選択できます。
12. システム・ディスクの標準パーティション・レイアウトを選択する場合は、第 2 章の「標準パーティション・レイアウトの作成」の手順に従います。カスタム・パーティション・レイアウトを選択する場合は、第 2 章の「カスタム・パーティション・レイアウトの作成」の手順に従います。
13. 今後の手順で必要となる場合があるので、次のコマンドを実行して、ボリューム・ヘッダの内容を確認します。

```
----- please choose one (? for help, .. to quit this menu)-----
[ro]otdrive           [o]ptiondrive         [e]xpert
[u]srrootdrive        [re]size
fx/repartition> label/show/directory

 0: sgilabel   block    3 size     512  2: sash       block 1914 size  159232
 1: ide        block    4 size  977920
```

必要な *sash* ファイルがボリューム・ヘッダにあるかどうかを確認します。この例では、項目 2 に表示されています。

14. *fx* と Command Monitor を終了して、「System Maintenance」メニューに戻ります。

```
----- please choose one (? for help, .. to quit this menu)-----
[para]meters       [part]itions       [b]ootinfo         [a]ll
[g]eometry         [s]giinfo          [d]irectory
fx/label/show> ../../exit
>> exit
```

15. 「System Maintenance」メニューから「Install System Software」を選択します。

root パーティションにはファイルシステムがないので、次のようなエラー・メッセージが表示されることがあります。

```
Mounting file systems:

/dev/dsk/dks0d1s0: Invalid argument
No valid file system found on: /dev/dsk/dks0d1s0
This is your system disk: without it we have nothing
on which to install software.
```

また、次のようなメッセージが表示されることもあります。ただし、root パーティションがマウントされ、*inst* が起動します。

```
Mounting file systems:

mount: /root/dev/usr on /root/usr: No such file or directory
mount: giving up on:
   /root/usr

Unable to mount all local efs, xfs file systems under /root
Copy of above errors left in /root/etc/fscklogs/miniroot

    /dev/miniroot            on  /
    /dev/dsk/dks0d1s0        on  /root
```

16. ファイルシステムの作成の確認を求めるメッセージが表示されたら、**yes** と入力します。

```
Make new file system on /dev/dsk/dks0d1s0 [yes/no/sh/help]: yes

About to remake (mkfs) file system on: /dev/dsk/dks0d1s0
This will destroy all data on disk partition: /dev/dsk/dks0d1s0.
```

```
        Are you sure? [y/n] (n): yes

        Block size of filesystem 512 or 4096 bytes? 4096

Doing: mkfs -b size=512 /dev/dsk/dks0d1s0
meta-data=/dev/rdsk/dks0d1s0     isize=256    agcount=8, agsize=248166 blks
data     =                       bsize=4096   blocks=248165
log      =internal log           bsize=512    blocks=1000
realtime =none                   bsize=4096   blocks=0, rtextents=0
Mounting file systems:

NOTICE: Start mounting filesystem: /root
NOTICE: Ending clean XFS mount for filesystem: /root
    /dev/miniroot            on  /
    /dev/dsk/dks0d1s0        on  /root
```

17. シェルの起動の確認を求めるメッセージが表示されたら、次に示すように、シェルを起動して、手作業で root ファイルシステムを作成します。必要に応じて usr ファイルシステムも作成します。

```
Please manually correct your configuration and try again.

        Press Enter to invoke C Shell csh: Enter

# mkfs /dev/dsk/dks0d1s0
meta-data=/dev/dsk/dks0d1s0      isize=256    agcount=8, agsize=31021 blks
data     =                       bsize=4096   blocks=248165
log      =internal log           bsize=4096   blocks=1000
realtime =none                   bsize=4096   blocks=0, rtextents=0
# exit
```

18. *inst* メイン・メニューが表示され、手順 16 または手順 17 で root ファイルシステムを作成していない場合は、次に示すように、root ファイルシステムを作成し、マウントします。また、usr ファイルシステムを使用している場合は、usr ファイルシステムも作成してマウントします。

```
Inst> admin
...
Admin> mkfs /dev/dsk/dks0d1s0

Make new file system on /dev/dsk/dks0d1s0 [yes/no/sh/help]: yes

About to remake (mkfs) file system on: /dev/dsk/dks0d1s0
This will destroy all data on disk partition: /dev/dsk/dks0d1s0.

        Are you sure? [y/n] (n): yes

        Block size of filesystem 512 or 4096 bytes? 4096
```

```
Doing: mkfs -b size=512 /dev/dsk/dks0d1s0
meta-data=/dev/rdsk/dks0d1s0     isize=256    agcount=8, agsize=248166 blks
data     =                       bsize=4096   blocks=248165
log      =internal log           bsize=512    blocks=1000
realtime =none                   bsize=4096   blocks=0, rtextents=0
Mounting file systems:

NOTICE: Start mounting filesystem: /root
NOTICE: Ending clean XFS mount for filesystem: /root
    /dev/miniroot            on  /
    /dev/dsk/dks0d1s0        on  /root


Re-initializing installation history database
Reading installation history .. 100% Done.
Checking dependencies .. 100% Done.

Admin> Enter
```

19. CD から IRIX ソフトウェアをインストールします。
20. 必要に応じて、ほかの CD からオプションのソフトウェアおよびパッチをインストールします。
21. *sash* を追加するために、ボリューム・ヘッダを変更する必要がない場合（手順 13 を参照）、新しいシステム・ディスクの作成はこれで終わりです。残りの手順を行う必要はありません。
22. ディスクのボリューム・ヘッダにプログラムを追加するには、まずシェルを起動します。

    ```
    Inst> sh
    ```

23. 必要に応じて、第 2 章の「dvhtool によるボリューム・ヘッダへのファイルの追加」で示した手順に従って、*sash* をシステム・ディスクのボリューム・ヘッダに追加します。*/stand* ディレクトリは、*/root/stand* にマウントされます。
24. シェルを終了します。

    ```
    # exit
    ```

25. *inst* を終了して、システムを起動します。

    ```
    Inst> quit
    ```

IRIX からの新しいシステム・ディスクの作成

この手順では、オプション・ディスクをシステム・ディスクに変換する方法について説明します。オプション・ディスクにはファイルシステムは不要です。また、この手順を開始する前にオプション・ディスクをマウントする必要もありません。

注意 : ここで示す手順を実行すると、オプション・ディスクのデータはすべて破壊されます。保存しておくファイルがオプション・ディスクにある場合は、オプション・ディスクのすべてのファイルをテープまたは別のディスクにバックアップしてから、この手順を行ってください。

この手順は、1 GB のディスクを 2 GB のディスクにする操作などシステム・ディスクの容量を増やす場合、または別のシステムに移動可能なシステム・ディスクを作成する場合に使用できます。この手順を使用して、IRIX システム・ソフトウェア CD からソフトウェアをインストールして、新しいディスクを作成します。システム・ディスクのコピーを作成するには、43 ページの「クローン化による新しいシステム・ディスクの作成」を参照してください。別のシステムにシステム・ディスクを作成する場合、IRIX ではハードウェアに依存しているので、同じハードウェア構成である必要があります。

この手順は、特権ユーザとして実行してください。また、この手順では、何度かシステムを再起動する必要があるので、ほかのユーザがシステムを使用していないことを確認してください。

オプション・ディスクをシステム・ディスクに変換するには、次の手順に従ってください。

1. *hinv* を使用して、システム・ディスクに変換するディスクのコントローラおよびドライブ・アドレスを確認します。この手順のコマンド例および出力例では、ディスクがコントローラ 0、ドライブ・アドレス 2 にあるものとします。状況に応じて、各自のコントローラおよびドライブ・アドレスに置換えてください。
2. システム・ディスクとして使用できるように、ディスク・パーティションを再分割するには、まず *fx* を起動します。

```
# fx
fx version 6.4, Sep 29, 1996
```

3. メッセージに対して、変換しているディスクのコントローラ番号とドライブ・アドレス、および lun 番号として 0 を入力します。

```
fx: “device-name” = (dksc) Enter
fx: ctlr# = (0) Enter
fx: drive# = (1) 2
fx: lun# = (0) Enter
...opening dksc(0,2,0)
...drive selftest...OK
Scsi drive type == SGI     SEAGATE ST31200N8640

----- please choose one (? for help, .. to quit this menu)-----
[exi]t               [d]ebug/             [l]abel/
[b]adblock/          [exe]rcise/          [r]epartition/
```

4. *repartition* コマンドを実行します。

```
fx> repartition
```

```
----- partitions-----
part  type        blocks          Megabytes   (base+size)
  7: efs        3024 + 2057832       1 + 1005
  8: volhdr        0 + 3024          0 + 1
 10: volume        0 + 2060856       0 + 1006

capacity is 2061108 blocks
```

5. root パーティションと usr パーティションを結合するか、または別々にするかによって、*rootdrive* または *usrrootdrive* を選択します。それぞれの長所と短所については、第 1 章の「システム・ディスク、オプション・ディスク、およびパーティション・レイアウト」を参照してください。この例では、XFS 用に設定した、root パーティションと usr パーティションを結合したディスクを選択します。

```
----- please choose one (? for help, .. to quit this menu)-----
[ro]otdrive        [u]srrootdrive     [o]ptiondrive      [re]size

fx/repartition> rootdrive

fx/repartition/rootdrive: type of data partition = (xfs) Enter
Warning: you will need to re-install all software and restore user data
from backups after changing the partition layout.  Changing partitions
will cause all data on the drive to be lost.  Be sure you have the drive
backed up if it contains any user data.  Continue? y

----- partitions-----
part  type        blocks          Megabytes   (base+size)
  0: xfs        3024 + 1976184       1 + 965
  1: raw     1979208 + 81648        966 + 40
  8: volhdr        0 + 3024          0 + 1
 10: volume        0 + 2060856       0 + 1006

capacity is 2061108 blocks
```

6. *fx* を終了します。

```
----- please choose one (? for help, .. to quit this menu)-----
[ro]otdrive        [u]srrootdrive     [o]ptiondrive      [re]size
fx/repartition> ../exit
```

7. 第 2 章の「dvhtool によるボリューム・ヘッダへのファイルの追加」で示す手順に従って、変換するディスクのボリューム・ヘッダの内容を調べ、ボリューム・ヘッダに *sash* がなければ追加します。

8. 変換しているディスクの root パーティションに root ファイルシステムを作成します。たとえば、ブロック・サイズが 4 KB で、1,000 ブロック内部ログ（デフォルト値）の XFS を作成する場合は、次のコマンドを実行します。

```
# mkfs /dev/dsk/dks0d2s0
```

XFS ファイルシステムの作成方法の詳細については、第 6 章の「XFS ファイルシステムの使用計画」と「XFS ファイルシステムの作成」を参照してください。ファイルシステムは、作成後にマウントする必要はありません。

9. ディスクに個別の usr パーティションがある場合は、そのパーティションにもファイルシステムを作成します。たとえば、次のコマンドを実行します。

```
# mkfs /dev/dsk/dks0d2s6
```

10. システムの CD-ROM ドライブまたはリモートシステムの CD-ROM ドライブに、インストールする IRIX リリースが格納されている CD を挿入します。

11. システムを停止して、CD からミニルートを起動します。起動方法については、『IRIX Admin: Software Installation and Licensing』を参照してください。

12. 「Administrative Commands」メニューに切替えて、古いシステム・ディスクから root パーティションと usr パーティション（使用されている場合）をアンマウントし、代わりに新しいディスクの root パーティションと usr パーティション（使用されている場合）をマウントします。たとえば、古いシステム・ディスクに root パーティションと usr パーティションがあり、新しいシステム・ディスクに root パーティションしかない場合のコマンドは、次のとおりです。

```
Inst> admin
Admin> umount /root
Admin> umount /root/usr
Admin> mount /dev/dsk/dks0d2s0 /root
Admin> Enter
```

13. 新しいシステム・ディスクの root パーティションと usr パーティション（使用されている場合）が、*/root* および */root/usr*（使用されている場合）としてマウントされているか確認します。この例では、手順 12 の出力を表示します。

```
Inst> sh df

Filesystem             Type  blocks     use     avail  %use Mounted on
/dev/miniroot           xfs    49000    32812    16188  67  /
/dev/dsk/dks0d1s0       xfs  1984325      251  1984074   0  /root
```

注意：間違ったパーティションがマウントされている場合は、システム・ソフトウェアが *inst* によって間違ったパーティションにインストールされ、そのパーティションのデータが破壊されます。

---

14. 通常、この CD からシステム・ソフトウェアをインストールし、ほかの CD からオプションとパッチをインストールします。これらのインストール方法については、『IRIX Admin: Software Installation and Licensing』を参照してください。

15. *inst* を終了して、システムを再度 IRIX に戻します。古いシステム・ディスクが起動されます。

16. 古いシステム・ディスクを交換する前に、またはディスクを別のシステムに移動する前に、新しいシステム・ディスクをテストするには、まずシステムを停止して、PROM モニタを起動します。
17.「System Maintenance」メニューを使って、Command Monitor を起動します。
18. 次のコマンドを実行して、新しいシステム・ディスクからシングル・ユーザ・モードでシステムを起動します。この例ではコントローラ 0 とドライブ・アドレス 2 を使用しています。つまり、この例では、数字が 3 つ並んだ箇所の 1 番目と 2 番目の位置に新しいシステム・ディスクの値を指定します。

```
>> setenv initstate s
>> boot -f dksc(0,2,8)sash dksc(0,2,0)unix root=dks0d2s0
```

19. *MAKEDEV* と *autoconfig* を実行します。

```
# cd /dev
# ./MAKEDEV
# /etc/autoconfig -f
```

20. *reboot* コマンドを使用して、マルチユーザ・モードでシステムを再起動します。

これで新しいシステム・ディスクが、現在のシステム、または同じハードウェア構成の別のシステムのシステム・ディスクと交換する準備が整いました。

クローン化による新しいシステム・ディスクの作成

この手順では、オプション・ディスクをシステム・ディスクに変換する方法について説明します。この手順は、同一のシステム・ディスクを持つ複数のシステムを設定する場合に使用します。これらのシステムのプロセッサとグラフィック・タイプは同じである必要があります。

注意：この手順を実行すると、オプション・ディスクのデータがすべて破壊されます。保存しておくファイルがオプション・ディスクにある場合は、オプション・ディスクのすべてのファイルをテープまたは別のディスクにバックアップしてから、この手順を行ってください。

この手順は、特権ユーザとして実行してください。作成するシステム・ディスクを元のシステム・ディスクと同じにするために、システムをシングル・ユーザ・モードにします。

1. 次に、パラメータなしで prtvtoc を実行し、システム (root)・ディスクのディスク・パーティションをリスト表示します。

```
# prtvtoc
Printing label for root disk
```

```
* /dev/root (bootfile "/unix")
*     512 bytes/sector
Partition  Type  Fs   Start: sec    Size: sec   Mount Directory
 0          xfs  yes        4096      4138249
 1          raw           4142345       262144
 8       volhdr                 0         4096
10       volume                 0      4404489
```

2. 次に、クローンであるオプション・ディスクのディスク・パーティションをリスト表示します。

```
# prtvtoc /dev/rdsk/dks0d2vh
...
Partition  Type  Fs   Start: sec    Size: sec   Mount Directory
 0          efs             3024        50652
 1          raw            53676        81648
 6          efs           135324      1925532
 8       volhdr                 0         3024
10       volume                 0      2060856
```

3. 2 つのディスク・パーティションを比較します。この 2 つのディスクでは、root および usr (使用されている場合) のパーティションレイアウトが同じである必要があります。パーティション・レイアウトが異なる場合は、第 2 章の「fx によるディスク・パーティションの再分割」に示す手順を使用して、システム・ディスクのパーティションと同じになるように、オプション・ディスクのパーティションを再分割してください。

4. 第 2 章の「dvhtool によるボリューム・ヘッダへのファイルの追加」に示す手順を使用して、オプション・ディスクのボリューム・ヘッダの内容を確認して、必要に応じてシステム・ディスクからプログラムをコピーし、追加します。

5. オプション・ディスクの root パーティションに新しいファイルシステムを作成します。たとえば、ブロック・サイズが 4 KB で、1,000 ブロック内部ログ（デフォルト値）を持つ XFS の root ファイルシステムを作成するには、次のコマンドを実行します。

```
# mkfs /dev/dsk/dks0d2s0
```

XFS ファイルシステムの作成方法の詳細については、第 6 章の「XFS ファイルシステムの使用計画」と「XFS ファイルシステムの作成」を参照してください。ファイルシステムは、作成後にマウントする必要はありません。

6. 個別の usr パーティションがある場合は、オプション・ディスクの usr パーティションにファイルシステムを作成します。たとえば、次のコマンドを実行します。

```
# mkfs /dev/dsk/dks0d2s6
```

7. オプション・ディスクのファイルシステムに対して一時的なマウント・ポイントを作成します。

```
# mkdir /clone
```

8. オプション・ディスクの root ファイルシステムをマウントし、マウント・ポイントにディレクトリを変更します。

```
# mount /dev/dsk/dks0d2s0 /clone
# cd /clone
```

9. *dump*（EFS の場合）または *xfsdump*（XFS の場合）を使用して、システム・ディスクの root ファイルシステムをオプション・ディスクの root ファイルシステムにコピーします。*dump* コマンドは、次のとおりです。

```
# dump 0f - / | restore xf -
```

xfsdump コマンドは、次のとおりです。

```
# xfsdump -l 0 - / | xfsrestore - .
```

10. ディスクに usr パーティションがない場合は、手順 13 に進んでください。

11. usr ファイルシステムをコピーするには、root ファイルシステムでなく、usr ファイルシステムをマウントします。

```
# cd ..
# umount /clone
# mount /dev/dsk/dks0d2s6 /clone
# cd /clone
```

12. *dump*（EFS の場合）または *xfsdump*（XFS の場合）を使用して、システム・ディスクの usr ファイルシステムをオプション・ディスクの usr ファイルシステムにコピーします。*dump* コマンドは、次のとおりです。

```
# dump 0f - /usr | restore xf -
```

xfsdupm コマンドは、次のとおりです。

```
# xfsdump -l 0 - /usr | xfsrestore - .
```

13. 一時的なマウント・ポイントにマウントされているファイルシステムをアンマウントして、マウント・ポイントを削除します。

```
# cd ..
# umount /clone
# rmdir /clone
```

これで、オプション・ディスクはシステム・ディスクのコピーになりました。同じハードウェア構成を持つシステムに移動できます。

新しいオプション・ディスクの追加

アドバイス : 「ツールチェスト (Toolchest)」の「システム (System)」->「システム・マネージャ (System Manager)」->「ハードウェアとデバイス (Hardware and Devices)」->「ディスク・マネージャ (Disk Manager)」を使用して、新しいオプション・ディスクを追加できます。詳細については、『Personal System Administration Guide』の 第 3 章「新しいハードディスクの設定」を参照してください。第 6 章「2 番目のディスクの活用」には、オプション・ディスクを効率的に使用する方法が示されています。

シェル・コマンドを使用してシステムに新しいオプション・ディスクを追加するには、次の手順に従ってください。

1. ハードウェアをインストールします。詳細については、システムのオーナーズ・ガイドを参照してください。
2. 必要に応じて、ボリューム・ヘッダを初期化します。21 ページの「fx によるディスクのフォーマットと初期化」を参照してください。
3. 必要に応じて、新しいディスクをパーティションに分割します。この場合、オプション・ディスクとしてパーティションに分割します。パーティションの分割方法については、25 ページの「fx によるディスク・パーティションの再分割」を参照してください。
4. 次の手順に進むために、新しいディスクに接続されているコントローラのタイプ（内蔵 SCSI コントローラ、または非内蔵コントローラ）を確認します。システムのディスクを表示する方法については、19 ページの「hinv によるシステム上のディスクのリスト表示」を参照してください。
5. 内蔵 SCSI コントローラのオプション・ディスクをシステムに追加するには、*Add_disk* コマンドを使用して残りの手順を実行し、ディスクを構成します。

   # **Add_disk** *controller_number drive_address lun_number*

   コントローラ 0 にある 2 番目のディスクをシステムに追加している場合は、ディスク、コントローラ番号、論理ユニット番号を指定する必要はありません。デフォルトで、コントローラ 0 のディスク 2 が追加されます。3 番目（または、それ以上）のディスクを追加している場合、またはコントローラ 0 以外のコントローラにディスクを追加している場合は、ディスクとコントローラを指定する必要があります。また、ディスク・デバイスの論理ユニット番号が 0 でない場合は、論理ユニット番号を指定する必要があります。

   Add_disk ではディスクに有効なファイルシステムがあるかどうかを確認します。ファイルシステムがある場合は、*Add_disk* は警告メッセージを表示し、確認してから、既存のファイルシステムを破壊して、新しいファイルシステムを作成します。

   Add_disk コマンドでは、次の動作を実行します。

- 新しいディスクに対して、キャラクタ・デバイス・ファイルおよびロー・デバイス・ファイルを作成します。
- ディスクに XFS ファイルシステムを作成します。
- マウント・ディレクトリを作成します。
- ファイルシステムをマウントします。
- */etc/fstab* ファイルにマウント順序を追加します。

6. 非内蔵コントローラのオプション・ディスクの場合は、ファイルシステムを作成することで新しいオプション・ディスクの構成が完了します。手順については、第 6 章の「XFS ファイルシステムの作成」または 「inst からのファイルシステムの作成」 を参照してください。

XLV 論理ボリュームの概念

この章では、XLV 論理ボリュームの概念について説明します。論理ボリュームを使用すると、1 つのファイルシステムを複数のディスク・パーティションにわたって設定できます。IRIX は、XLV 論理ボリュームをサポートしています。これは、SGI 社が開発した論理ボリューム設計です。IRIX の旧リリースでは、*lv* 論理ボリュームという旧式の論理ボリューム設計をサポートしていました。*lv* 論理ボリュームから XLV 論理ボリュームへの変換手順については、第 4 章の「lv 論理ボリュームの XLV 論理ボリュームへの変換」で説明されています。

この章では、次の項目について説明します。

XLV 論理ボリュームの管理手順については、第 4 章「XLV 論理ボリュームの作成と管理」を参照してください。

メモ：XVM 論理ボリュームの概念については、『XVM Volume Manager Administrator's Guide』を参照してください。

XLV 論理ボリュームについて

論理ボリュームを使用すると、複数のディスク・パーティションにまたがるファイルシステム、ロー・デバイス、またはブロック・デバイスを作成できます。論理ボリュームは、標準のディスク・パーティションと同じように機能し、*/dev* ディレクトリにブロック・デバイスおよびキャラクタ・デバイスとして表示されます。また、ディスク・デバイスを指定できる位置に引数として使用できます。

ファイルシステムを作成、マウントして、論理ボリューム上で通常どおりに使用できます。また、論理ボリュームをブロック・デバイスまたは ロー・デバイスとして使用することもできます。XLV 論理ボリュームでは、ボリュームにアクセスするアプリケーションに対して、ディスクのプレックス化（ミラーリング）やストライプ化などを分かりやすく行うことができます。次の場合に論理ボリュームを作成します。

- ファイルシステムまたはディスク・デバイスを物理ディスクのサイズよりも大きくする場合
- ディスクの I/O パフォーマンスを向上させる場合

論理ボリュームの欠点は、論理ボリュームで使用するすべてのディスクが正常に動作する必要があることです。論理ボリュームを 3 つのディスクに設定している場合、そのうちの 1 つのディスクが破損すると、ほかの 2 つのディスクに記録された情報も使用できなくなるので、バックアップから復元する必要があります。ただし、オプションの Disk Plexing Option ソフトウェアを使用すると、XLV 論理ボリュームの内容のコピー（プレックス）を複数作成でき、ディスクが 1 つ破損しても、XLV 論理ボリュームに記録された情報をすべて使用できます。

XLV 論理ボリュームが ロー・デバイスとして使用される場合、および XLV 論理ボリュームに XFS ファイルシステムが作成される場合、次のような利点があります。

- 大きな論理ボリュームをサポートします。32 ビット・システムの場合は最大 1 テラバイト、62 ビット・システムの場合は無制限にサポートします。
- ディスク・ストライプ化をサポートして、I/O パフォーマンスを向上します。
- プレックス化（ミラーリング）をサポートして、システムとデータの信頼性を向上します。
- ボリューム・サイズの増加など、オンラインでボリュームを再設定することで、システムの停止時間を抑制します。

XFS ファイルシステムを使用する場合、XLV にはさらに次の利点があります。

- ファイルシステム・ジャーナル・レコードを別のパーティションに持つことで、最大のパフォーマンスを実現します。このパーティションは、別のディスクにあってもかまいません。
- リアルタイム・データにアクセスします。

XLV 論理ボリュームに複数の物理ディスク・ドライブのパーティションを含めることができます。デフォルトでは、まず、データは先頭のディスク・パーティションに書込まれ、次に、2 番目のディスク・パーティションというように順次に書込まれます。図 3-1 に、ストライプ化されていない論理ボリュームのパーティションにデータが書込まれる順序を示します。

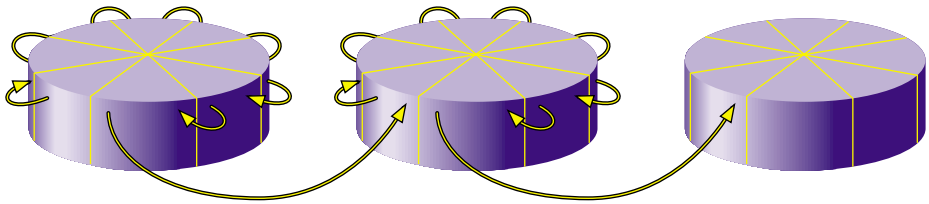


図 3-1　ストライプ化されていない論理ボリュームへのデータの書込み

ストライプ化されている論理ボリュームでは、複数のディスク・パーティションは同じサイズである必要があります。論理ボリュームがストライプ化されると、ストライプ・ユニットと呼ばれるデータが最初のディスクに書込まれ、次のストライプ・ユニット分のデータが 2 番目のディスクにというように順番に書込まれます。各ディスクに書込まれると、次のストライプ・ユニット分のデータが最初のディスクに書込まれ、これを繰返してストライプが形成されます。図 3-2 に、ストライプ化された論理ボリュームにデータが書込まれる順序を示します。

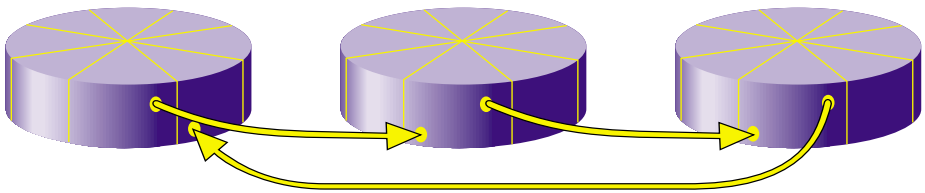


図 3-2　論理ボリュームへのデータの書込み

ストライプ内の各ストライプ・ユニットは同時に読込みと書込みが可能です。このため、I/O パフォーマンスは向上します。ストライプ化の最高のパフォーマンスを得るためには、別々のコントローラにストライプ化されているディスクを接続してください。この構成では、各ディスクとシステム間のデータ・パスは独立しています。ただし、同じコントローラでストライプ化された SCSI ディスクを使用すると、多少パフォーマンスを向上できます。

XLV ボリュームで XFS ファイルシステムが使用される場合、各論理ボリュームには、最大 3 つのサブボリュームを含めることができます。3 つのサブボリュームとは、データ・サブボリューム（必須）、ログ・サブボリューム、リアルタイム・サブボリュームです。データ・サブボリュームには、通常ユーザ・ファイルと *metadata* ファイルシステム（i ノード、インダイレクト・ブロッ

ク、ディレクトリ、空き領域ブロック）が含まれます。ログ・サブボリュームは、ファイルシステムのジャーナル・レコード用に使用されます。これは、外部ログと呼ばれます。ログ・サブボリュームがない場合、ジャーナル・レコードはデータ・サブボリューム（内部ログ）に格納されます。ビデオなどのように特殊な I/O 帯域幅の条件を持つデータは、オプションのリアルタイム・サブボリュームに格納できます。第 8 章の「リアルタイム・サブボリュームの使用」で、この手順を説明しています。

XLV では、ボリュームを停止することなく、ボリューム内のデータ（プレックス）のコピーの追加や削除、ボリュームのサイズの増加（拡張）、プレックス化されたボリュームの破損したエレメントの交換を行うことができます。これによって、システムの信頼性と有効性が高まります。

XLV 論理ボリュームを作成するには、2 つの方法があります。どちらの方法を使用するかは、空のディスクを使用するかディスク・パーティションのファイルシステムを使用するかによって異なります。ディスクを空にした状態で論理ボリュームを作成する場合は、次の手順に従ってください。

1. 必要に応じて、ディスク・パーティションを作成します。第 2 章の「fx　によるディスク・パーティションの再分割」を参照してください。
2. XLV 論理ボリュームを作成します。第 4 章の「xlv_make によるボリューム・オブジェクトの作成」と「例 3 : 外部ログを持つ XFS ファイルシステムのプレックス XLV 論理ボリュームの作成」を参照してください。
3. XLV 論理ボリューム上にファイルシステムを作成します。第 6 章の「XFS ファイルシステムの作成」または 「inst からのファイルシステムの作成」を参照してください。

XLV 論理ボリュームを作成する 2 番目の方法では、ディスク・パーティションにファイルシステムがある場合、既存のディスク・パーティションおよび新しいディスク・パーティションがある論理ボリュームを作成することによってファイルシステムのサイズを増やし、ファイルシステムを拡張します。この手順については、第 6 章の「別のディスクへの XFS ファイルシステムの拡張」で説明します。

lv 論理ボリュームから XLV 論理ボリュームへの変換は簡単です。*lv_to_xlv* コマンドと *xlv_make* コマンドを使用すると、データをダンプしてリストアすることなく、*lv* 論理ボリュームを XLV 論理ボリュームに変換できます。

ディスクが 1 台しかないシステムの場合、XLV 論理ボリュームの使用はお薦めできません。

XLV 論理ボリュームの構成

XLV 論理ボリュームは、論理記憶オブジェクトの階層構造から構成されます。つまり、ボリュームはサブボリュームから構成され、サブボリュームはプレックスから構成され、プレックスはボリューム・エレメントから構成されます。さらに、ボリューム・エレメントはディスク・パーティションから構成されます。図 3-3 に、この記憶単位の階層構造を示します。これは、比較的複雑な論理ボリュームの例です。

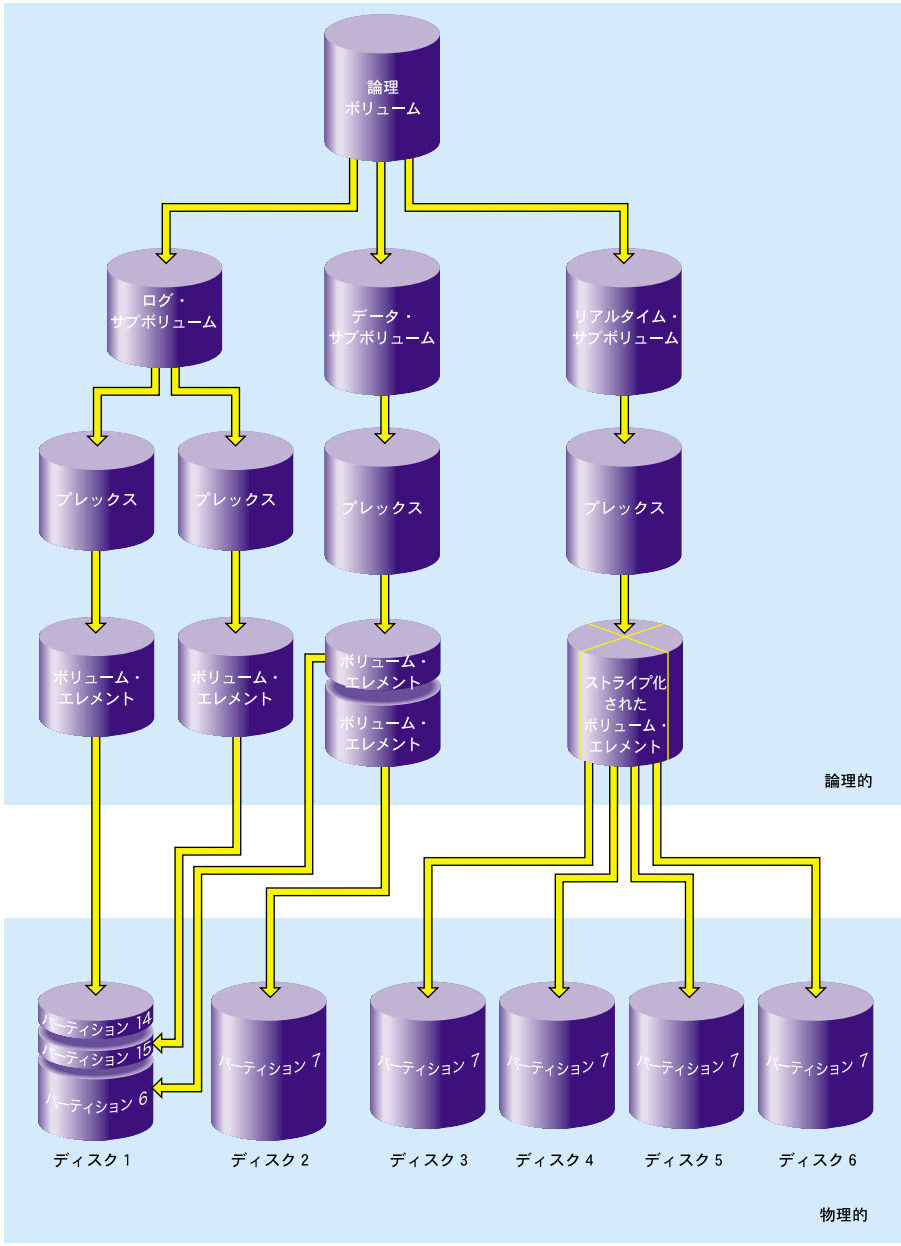


図 3-3 XLV 論理ボリュームの例

図 3-3 に、XLV 論理ボリューム内のボリューム、サブボリューム、プレックス、ボリューム・エレメントの関係を示します。この例では、6 つの物理ディスク・ドライブに 8 つのディスク・パーティションがあります。この論理ボリュームには、ログ・サブボリューム、データ・サブボリューム、リアルタイム・サブボリュームがあります。ログ・サブボリュームには、信頼性を向上させるための 2 つのプレックス（データのコピー）があります。ただし、データ・サブボリュームとリアルタイム・サブボリュームはプレックス化されていません。つまり、この 2 つのサブボリュームはそれぞれ 1 つのプレックスで構成されています。各ログ・プレックスは、ディスク 1 のディスク・パーティションであるボリューム・エレメントで構成されています。データ・サブボリュームのプレックスは、ディスク 1 の残りのパーティションとディスク 2 のすべてのパーティションという 2 つのボリューム・エレメントで構成されています。リアルタイム・サブボリュームに使用されるプレックスは、パフォーマンスを向上させるためにストライプ化されています。ストライプ化されたボリューム・エレメントは、4 つのディスク・パーティションで構成され、各ディスク・パーティションはディスク全体を表しています。

次に、これらの論理記憶オブジェクトの詳細を説明します。

ボリューム

ボリュームはサブボリュームで構成されます。どのボリュームにもかならずデータ・サブボリュームがあります。このほかにログ・サブボリュームとリアルタイム・サブボリュームという 2 種類のサブボリュームがあり、この 2 種類はオプションです。XFS ファイルシステムの場合は、データ・サブボリューム、オプションのログ・サブボリューム、オプションのリアルタイム・サブボリュームで構成されます。EFS ファイルシステムの場合、ボリュームは一つのサブボリュームであるデータ・サブボリュームだけで構成されます（EFS ファイルシステムは以前の IRIX のリリースでサポートされていたファイルシステムで、付録 A「EFS ファイルシステム」で説明されています）。図 3-4 に、ボリュームとサブボリュームの関係を示します。

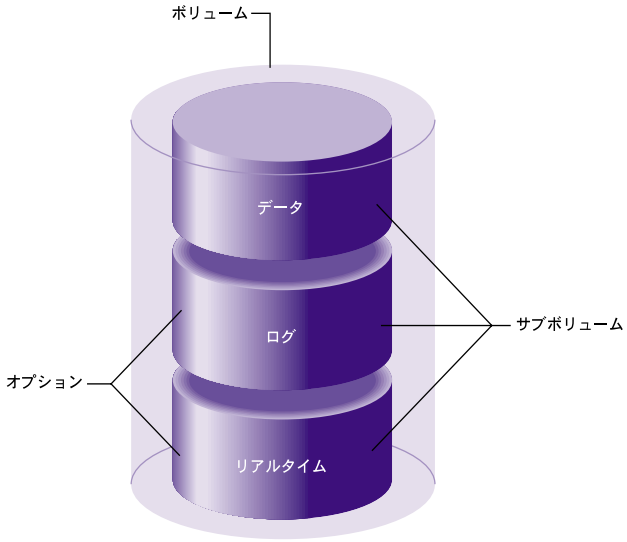


図 3-4 ボリュームの構成

各ボリュームは、単一のファイルシステムまたはロー・パーティションとして使用できます。システムの起動時に使用されるボリューム情報は、そのボリュームで使用される各ディスクのボリューム・ヘッダの論理ボリューム・ラベルに格納されます（第 1 章の「ボリューム・ヘッダ」を参照）。システムの起動時にサブボリュームのいずれかがオンライン状態にならない場合は、ボリュームもオンラインで動作しません。ボリュームの作成、削除、別のシステムへの移動を行うことができます。

サブボリューム

第 3 章の「ボリューム」で説明したように、各論理ボリュームは 1 ～ 3 つのサブボリュームで構成されます。図 3-5 で示すようにサブボリュームは、1 ～ 4 つのプレックスで構成されています。

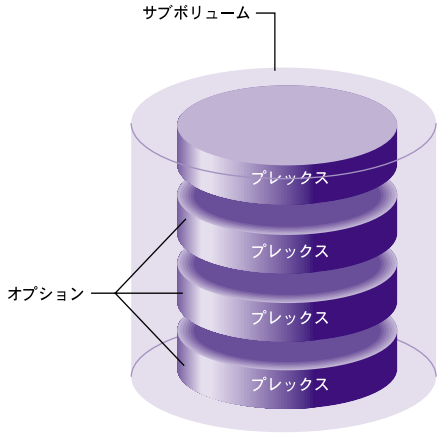


図 3-5　　サブボリュームの構成

メモ: オプションのプレックスを使用するための XLV プレックス化機能は、Disk Plexing Option ソフトウェア・オプションを購入した場合のみ使用できます。

サブボリュームのアドレス空間とサブボリュームのタイプはそれぞれ異なります。次に、サブボリュームのタイプを示します。

データ・サブボリューム
: すべての XLV 論理ボリュームで必要になります。EFS ファイルシステムで唯一のサブボリュームです（EFS ファイルシステムは以前の IRIX のリリースでサポートされていたファイルシステムで、付録 A「EFS ファイルシステム」で説明されています）。

ログ・サブボリューム
: ログ・サブボリュームには、XFS ジャーナル情報があります。この情報は、ファイルシステムのトランザクション・ログで、クラッシュした場合にシステムを即座に回復するために使用されます。ログ情報は、ログ・サブボリュームではなく、データ・サブボリュームに格納される場合もあります。詳細については、第 6 章の「ログ・タイプとログ・サイズの選択」と mkfs_xfs(1M) マン・ページおよび **-l** オプションの説明を参照してください。

リアルタイム・サブボリューム
: リアルタイム・サブボリュームは、データの整合性よりも応答時間の保証を重視するビデオなどのデータ・アプリケーションで通常使用されます。第 8 章「帯域保証 I/O のシステム管理」では、アプリケーションがリアルタイム・サブボリュームのデータにアクセスする方法を説明します。

サブボリュームでは、データのタイプが区別されます。たとえば、ユーザ・データはファイルシステムのログ・データを上書きできません。また、サブボリュームでは、パフォーマンスと信頼性を向上させるために、ファイルシステムのデータとユーザ・データを設定できます。たとえば、異なるディスク・ドライブにサブボリュームを設定すると、パフォーマンスを向上できます。

各サブボリュームは独立させて編成できます。たとえば、ログ・サブボリュームはフォルト・トレランスのためにプレックス化でき、リアルタイム・サブボリュームはビデオ再生用に最高のスループットを実現するために多数のディスクにストライプ化できます。

リアルタイム・サブボリュームを構成するボリューム・エレメントは、データ・サブボリュームまたはログ・サブボリュームに使用されるボリューム・エレメントと同じディスクに設定しないでください。これは、リアルタイム・サブボリュームのすべてのファイルで推奨する設定で、ハードウェアで有効な帯域保証 I/O に使用されるファイルでは必須の設定です。詳細については、第 8 章の「GRIO 用のハードウェア構成の必要条件」を参照してください。

サブボリュームの作成後、サブボリュームをボリュームから切離したり、ボリュームを削除せずにサブボリュームを削除したりすることはできません。サブボリュームは、ボリュームが削除されると、自動的に削除されます。

プレックス

サブボリュームに 1 ～ 4 つのプレックス（ミラー）を入れることができます。各プレックスは、すべてのサブボリュームのデータまたは一部の正確なコピーです。複数のプレックスから構成されるサブボリュームを作成すると、データの予備コピーができるので、システムの信頼性が向上します。

サブボリュームにプレックスが 1 つしかない場合、そのプレックスはサブボリュームのアドレス空間全体を占めます。これに対して、複数のプレックスがある場合、すべてのプレックスの集合がアドレス空間全体を占めているかぎり、各プレックスのアドレス空間に穴があってもかまいません。図 3-6 にこの構成例を示します。この例のサブボリュームには、3 つのプレックスがあります。完全なプレックスには、それぞれ 3 つのボリューム・エレメントがありますが、この例の 2 つのプレックスにはボリューム・エレメントが 1 つ足りません。この場合、各アドレス範囲に少なくとも 1 つのボリューム・エレメントがあるのでこの構成でもかまいません。プレックス 1

が切離されても（サブボリュームから削除されても）、各アドレス範囲に少なくとも 1 つのボリューム・エレメントがあるので、このサブボリュームは機能します。

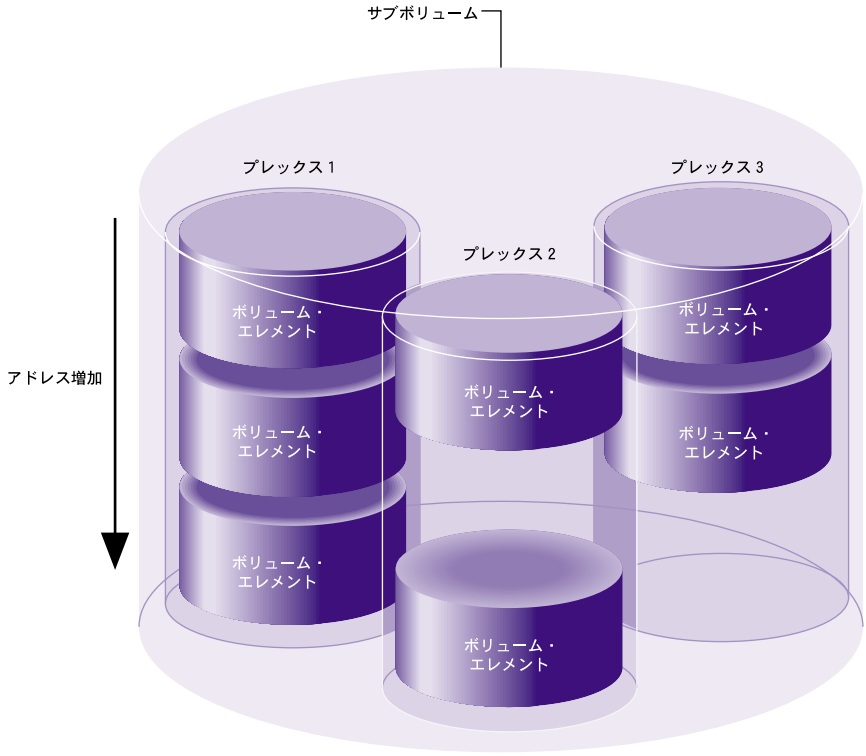


図 3-6　　プレックス化されたサブボリュームの例

データはすべてのプレックスに書込まれます。サブボリュームにプレックスが追加されると、プレックス全体が自動的にコピーされます（プレックス再生）。詳細については、xlv_assemble(1M) および xlv_plexd(1M) マン・ページを参照してください。

図 3-7 に示すように、プレックスは、ボリューム・エレメントで構成され、最大 128 のボリューム・エレメントが可能です。各ボリューム・エレメントは、サブボリュームでのアドレスの範囲を示します。

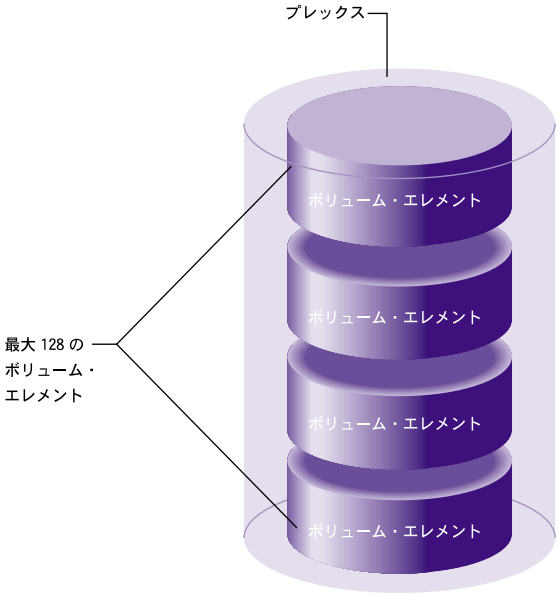


図 3-7　　プレックスの構成

プレックスが複数のボリューム・エレメントで構成されている状態をボリューム・エレメントが連結されているといいます。ボリューム・エレメントが連結されると、プレックスに順次書込まれるデータは、ボリューム・エレメントにも書込まれます。最初のボリューム・エレメントが一杯になると、次のボリューム・エレメントへと順次書込まれます。連結は、単一のディスクのサイズよりも大きなファイルシステムを作成する場合に便利です。

プレックスをサブボリュームに追加したり、複数のプレックスで構成されるサブボリュームから切離して別のサブボリュームに追加したり、複数のプレックスで構成されるサブボリュームから削除できます。

メモ：複数のプレックスを持つには、Disk Plexing Option ソフトウェア・オプションを購入し、FLEX*lm* ライセンスを取得およびインストールする必要があります。

ボリューム・エレメント

ボリューム・エレメントは、論理記憶オブジェクトの階層構造の最下位レベルです。つまり、ボリュームはサブボリュームで、サブボリュームはプレックスで、プレックスはボリューム・エレメントで構成されます。ボリューム・エレメントは、ディスク・パーティション（物理的な記憶エレメント）で構成されます。ボリューム・エレメントは、ストライプ化の有無にかかわらず、1 つまたは複数のディスク・パーティションで構成されています。なお、ストライプ化には、少なくとも 2 つのディスク・パーティションが必要です。プレックスには、シングルパーティション、ストライプ化、マルチパーティションという 3 種類のボリューム・エレメントを組合わせることができます。

シングルパーティション・ボリューム・エレメント

最も単純なボリューム・エレメントは、シングル・ディスク・パーティションです。ストライプ化されたボリューム・エレメントとマルチパーティション・ボリューム・エレメントの 2 つは、複数のディスク・パーティションで構成されます。図 3-8 に、シングル・パーティション・ボリューム・エレメントを示します。

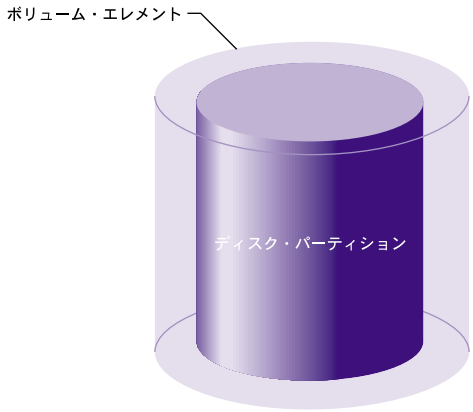


図 3-8　シングル・パーティション・ボリューム・エレメントの構成

ストライプ化されたボリューム・エレメント

図 3-9 に、ストライプ化されたボリューム・エレメントを示します。ストライプ化されたボリューム・エレメントは、複数のディスク・パーティションで構成されます。したがって、ストライプ・ユニットと呼ばれるデータ量が各ディスク・パーティションに書込まれてから、次のストライプ・ユニット分のデータが次のパーティションに書込まれます。

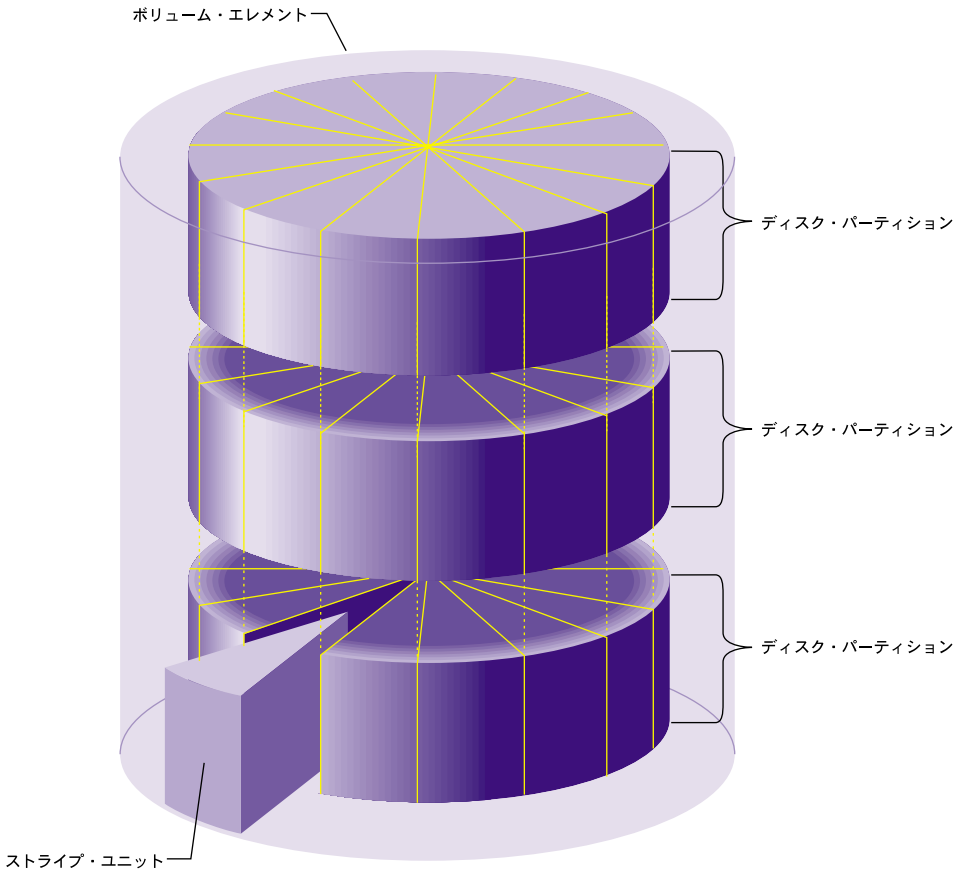


図 3-9 ストライプ化されたボリューム・エレメントの構成

ストライプ化を行うと、複数のディスク間でデータ・セクションを交互に使用できます。この並列 I/O 処理を実現することによって、パフォーマンスが高まります。次の経験則を元に、ストライプ・ユニット・サイズを選択できます。

- ストライプ・ユニット・サイズは、ストライプ化されたボリュームを使用するアプリケーションの I/O サイズと、ストライプ内のパーティション数によって決定されます。具体的には、ストライプ・ユニット・サイズは、アプリケーションの I/O サイズをパーティション数で割った値です。この値の場合、すべてのディスクが常にビジー状態という理想的な状態になります。

- デフォルトのストライプ・ユニットはデバイスのトラック・サイズであり、適切な値です。ディスクへの書込みより読取りを多く行う場合に特に適しています。
- 64K バイト以下のストライプ・ユニット・サイズはお薦めしません。
- 書込みのパフォーマンスを最大限に高めるには、ストライプ・ユニット・サイズが複数のトラックでなければなりません。しかし、ストライプ・ユニット・サイズが大きいと I/O バッファ・サイズが増大するため問題です。
- 最適なストライプ・ユニット・サイズを選択するには、並列 I/O 処理の利点、シングル・ディスク・ドライブに対する I/O の効率（読取りと書込みの単位が大きいとオーバーヘッドが減少します）、I/O バッファ・サイズの制約をそれぞれ考慮してバランスを取る必要があります。

マルチパーティション・ボリューム・エレメント

図 3-10 に、複数のディスク・パーティションからボリューム・エレメントが構成されるマルチパーティション・ボリューム・エレメントを示します。この構成では、ディスク・パーティションが順次アドレス指定されます。

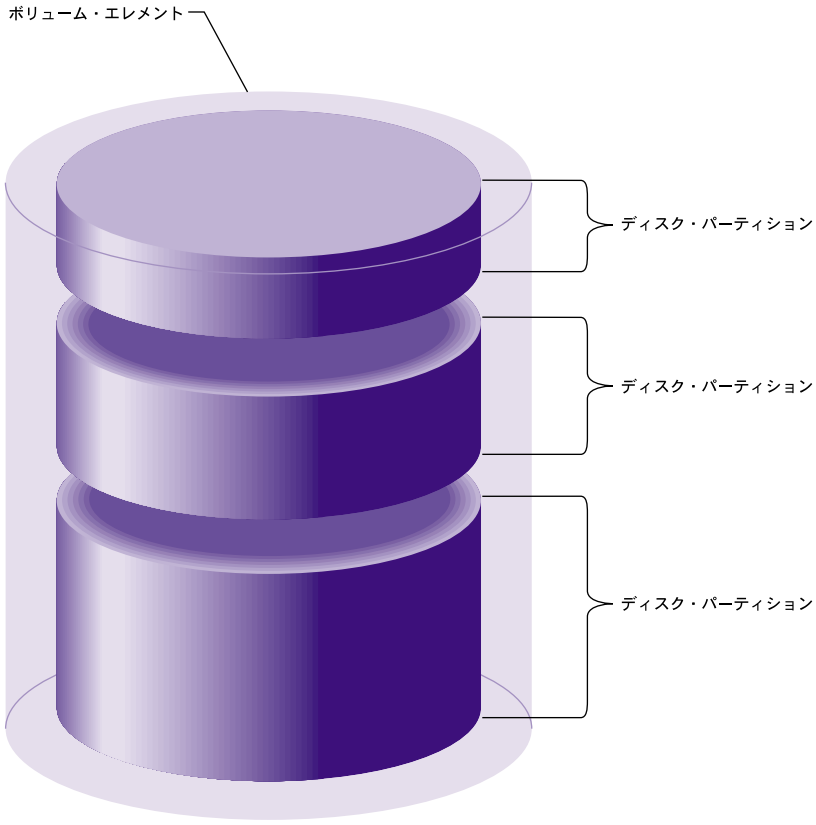


図 3-10　マルチパーティション・ボリューム・エレメントの構成

XLV 論理ボリューム名

ボリュームは、*/dev* ディレクトリにブロック・デバイスおよびキャラクタ・デバイスとして表示されます。XLV 論理ボリュームのデバイス名は、*/dev/xlv/volume_name* と */dev/rxlv/volume_name* です。*volume_name* は、*xlv_make* コマンドを使用してボリュームが作成される際に指定されるボリューム名です。*xlv_make* コマンドを使って指定するボリューム名、プレックス名、サブボリューム名、ボリューム・エレメント名には、ピリオド (.) を含むことができません。

メモ：XFS を使用している IRIX 6.2 と IRIX 5.3 では、XLV 論理ボリューム・デバイス・ファイル名はそれぞれ */dev/dsk/xlv/volume* および */dev/rdsk/xlv/volume* です。

あるシステムでボリュームが作成され、ディスクの移動によってそのボリュームが別のシステムに移動すると、新しいボリューム名として、元のボリューム名の前に元のシステムのホスト名が付けられます。たとえば、xlv0 というボリュームが engrlab1 というシステムから engrlab2 というシステムに移動された場合は、新しいシステムのボリュームのデバイス名は */dev/xlv/engrlab1.xlv0* になります。つまり、古いシステム名 engrlab1 が、ボリューム名 xlv0 の前に付けられます。

XLV デーモン

次に、XLV デーモンを示します。

xlv_labd — *xlv_labd* XLV 論理ボリューム・ラベルを更新します。インストールされていて、アクティブな XLV 論理ボリュームがある場合は、システムの立上げ時に自動的に起動します。

xlvd — プレックス・エラーの修復時に、プレックスに対して I/O 操作を完行します。プレックス・ソフトウェアがインストールされ、アクティブな XLV 論理ボリュームがある場合は、システムの立上げ時に自動的に作成されます。

xlv_plexd — *xlv_plexd* サブボリュームにあるプレックスすべてに同じデータを格納します。アクティブな XLV 論理ボリュームがある場合は、システムの起動時に自動的に起動します。

XLV では、明示的な設定ファイルは不要です。また、*chkconfig* コマンドでオン / オフの切替えはできません。XLV は、論理ボリューム・ラベルに書込まれた情報のみに基づいて、論理ボリュームを組立てることができます。初期化時に、システムはハードウェアの一覧を作成し、すべての論理ボリューム・ラベルを読取り、すでに定義されているボリュームに使用可能なディスクを自動的に集めます。

ディスクが一部不足している場合、XLV は、使用可能なプレックスにアドレス空間全体をマップするためのボリューム・エレメントがあるかどうかを確認します。アドレス空間全体が使用可能な場合は、プレックスの一部が完全でなくても、XLV はそのボリュームをオンライン状態にします。

XLV エラーに関する方針

ログ・サブボリュームとデータ・サブボリュームで読取りエラーが生じた場合、XLV は別のプレックス（使用可能な場合）を再度読取り、その結果を再度書込むことによってエラーのあるプレックスを修復します。XLV では、リアルタイム・データのエラー時に再試行は行いません。

ログ・サブボリュームとデータ・サブボリュームで書込みエラーが生じた場合、XLV はこの書込みエラーをハード・エラーとみなします。なお、ディスク・ドライバとコントローラは、ソフト・エラーを処理します。ハード・エラーのあるボリューム・エレメントがプレックス化された場合は、XLV はそのボリューム・エレメントをオフライン状態にし、それ以降そのボリューム・エレメントを無視します。ボリューム・エレメントがプレックス化されていない場合は、ボリューム・エレメントはボリュームと関連付けられたままエラーが返されます。

XLV は、リアルタイム・サブボリュームでの書込みエラーは処理しません。それ以降の読取りでは、エラー・メッセージは返されずに間違ったデータが返されます。

XLV 論理ボリュームの使用計画

次に、XLV 論理ボリュームの使用を計画する際に考慮すべき点について説明します。

XLV を使用しない場合

XLV 論理ボリュームを使用できない、または使用しない方が良い場合があります。

- ロー・スワップ・デバイスは XLV 論理ボリュームにできません。ただし、ファイルシステムにスワップ領域を標準のファイルとして追加でき、そのファイルシステムを XLV 論理ボリュームに設定することは可能です。詳細については、『IRIX Admin:System Configuration and Operation』の 第 6 章 「ディスクおよびスワップ領域の設定」を参照してください。
- ディスクが 1 つしかないシステムでは、XLV 論理ボリュームの使用はお薦めしません。
- ストライプ化または連結された XLV ボリュームは、root ファイルシステムでは使用できません。

サブボリュームの選択

XFS ファイルシステムで使用するサブボリュームを選択する際には、次の基本ガイドラインに従ってください。

- データ・サブボリュームが必要です。
- ログ・サブボリュームはオプションです。ログ・サブボリュームを使用しないと、ログ情報はデータ・サブボリュームの内部ログに格納されます。ほとんどの場合、外部ログを使用する必要はありません
- リアルタイム・サブボリュームはオプションです。

ファイルシステムを持たない大きな ロー・パーティションが必要な場合は、データ・サブボリュームだけを使用します。

リアルタイム・サブボリュームを持つ論理ボリュームを作成する場合は、データ・サブボリュームも含む必要があります。

EFS ファイルシステムで使用するサブボリュームを選択する際には、次の基本ガイドラインに従ってください（EFS ファイルシステムは以前の IRIX のリリースでサポートされていたファイルシステムで、付録 A「EFS ファイルシステム」で説明されています）。

- 使用できるのはデータ・サブボリュームのみです。
- EFS ファイルシステムの最大サイズは 8 GB です。したがって、データ・サブボリュームを 8 GB より大きくしないでください。それより大きい領域は無視されます。

サブボリュームのサイズの選択

サブボリュームのサイズを選択する際には、次の基本ガイドラインに従ってください。

- サブボリュームの最大サイズは、32 ビット・システム（IP17、IP20、IP22、IP32）では 1 テラバイトです。64 ビット・システム（IP19、IP21、IP25、IP26、IP27）では無制限です。
- ログ・サイズの選択とログ・サブボリュームのサイズの選択については、第 6 章の「ログ・タイプとログ・サイズの選択」で説明しています。最適なサイズのログ・パーティションを作成するためにディスク・パーティションを再分割しない場合は、使用可能なディスク・パーティションの選択によってログ・サイズが決まります。

プレックス化を行う場合

次に、プレックス化の際の基本ガイドラインを示します。

- データの信頼性と有効性を向上させる場合にプレックス化を使用します。
- root ファイルシステムをプレックス化できます。各プレックスは、シングル・パーティション・ボリューム・エレメントです。

- デュアルホスト、XLV 論理ボリューム（2 つのシステムに接続されたディスクの論理ボリューム）は、プレックス化できません。
- RAID ディスクはプレックス化しないでください。
- プレックスには穴がある場合、つまり、あるアドレス範囲のボリューム・エレメントが欠けている場合があります。ただし、サブボリューム内のプレックスのうち、少くとも 1 つがすべてのアドレス範囲のボリューム・エレメントを含んでいる場合にかぎります。
- サブボリュームの各プレックスのボリューム・エレメントは、ほかのプレックスの対応するボリューム・エレメント（同じアドレス範囲を持つボリューム・エレメント）とサイズが同じである必要があります。ボリューム・エレメント内の構造（シングル・パーティション、ストライプ化されたボリューム・エレメント、マルチパーティション）は、その対応するボリューム・エレメント内の構造と同じである必要はありません。
- ボリューム・エレメントのサイズを同じにするには、`fx` コマンドをエキスパート・モード (`fx -x`) で使用します。最初の `fx` メニューで、***repartition/expert -b*** コマンドを実行します。この結果、ブロック単位でパーティションを再分割でき、ボリューム・エレメントのサイズを自由に設定できます。

ストライプ化する場合

次に、ストライプ化の際の基本ガイドラインを示します。

- root ファイルシステムはストライプ化できません。
- ストライプ化された I/O は、ダイレクト I/O およびバッファ I/O で使用できます。ストライプ化するかどうかは、データのアクセス・パターンによって決まります。通常、ストライプ化した場合の方が、ストライプ化しない場合よりパフォーマンスが向上します。
- ストライプ化されたディスクは、アプリケーションが、ファイルシステムにあるストライプ化されたすべてのディスクにアクセスするような大きなデータ転送を行う場合にのみ、パフォーマンスが向上します。
- ストライプ化されたボリューム・エレメントは、同じサイズのディスク・パーティションから作成します。ディスク・パーティションのサイズが異なる場合は、最も小さいサイズが使用されます。それより大きなパーティションの空間は無視されます。
- 最高のパフォーマンスを実現するためには、ストライプ化されたボリューム・エレメントを含むディスクをそれぞれ独立したコントローラに設定します。ディスク・タイプによっては、コントローラあたり最高 4 つのディスクの場合にパフォーマンスが向上するものがあります。ディスクを 3 つ以上にしてもパフォーマンスが向上しないものもあります。
- ログ・サブボリュームをストライプ化できるのは、外部ログの場合のみです。ログをストライプ化しても、パフォーマンスは向上しません。

ディスク・パーティションを連結する場合

次に、ディスク・パーティションを連結する際の基本ガイドラインを示します。

- root ファイルシステムでは、ディスク・パーティションを連結できません。
- マルチパーティション・ボリューム・エレメントを 1 つ作成するより、シングル・パーティション・ボリューム・エレメントをプレックスに連結することをお薦めします。これはパフォーマンスのためではなく、信頼性を向上させるためです。マルチパーティション・ボリューム・エレメントにおいてディスク・パーティションが 1 つ破損すると、ボリューム・エレメント全体がオフライン状態になります。

XLV 論理ボリュームの作成と管理

この章では、コマンド行ユーティリティを使用して、XLV 論理ボリュームを作成および管理するための手順について説明します。これらの多数の手順を実行するためのグラフィカル・ユーザ・インタフェースが *xlvm* コマンドから提供されています。*xlvm* コマンドの詳細については、オンライン・ヘルプを参照してください。

メモ： XVM 論理ボリュームの管理については、『XVM Volume Manager Administrator's Guide』を参照してください。

この章では、次の項目について説明します。

プレックス化のサポートの確認

第 3 章「XLV 論理ボリュームの概念」で説明したように、複数のプレックスを使用できる XLV プレックス機能は、Disk Plexing Option ソフトウェア・オプションを購入し、FLEX*lm* ライセンスをインストールしたときのみ使用できます。

xlv_mgr コマンドを使用すると、プレックス・ソフトウェアと有効なライセンスがインストールされていることを確認できます。次の手順に従ってください。

1. *xlv_mgr* を起動します。

   ```
   # xlv_mgr
   ```

2. *showconfig* コマンドを実行します。

   ```
   xlv_mgr> show config
   Allocated subvol locks: 30      locks in use: 6
   Plexing license: present
   Plexing support: present
   Maximum subvol block number: 0x7fffffffffffffff
   ```

   出力の第 2 行目と第 3 行目にある “Plexing license: present” と “Plexing support: present” は、有効なライセンスを持つプレックス・ソフトウェアがインストールされていることを示しています。

3. *xlv_mgr* を終了します。

   ```
   xlv_mgr> quit
   ```

xlv_make によるボリューム・オブジェクトの作成

xlv_make コマンドは、未使用のディスク・パーティションから、ボリューム、サブボリューム、プレックス、およびボリューム・エレメントを作成します。このコマンドは、ディスクのボリューム・ヘッダのみに XLV 論理ボリューム・ラベルを書込みます。したがって、ディスク・パーティションのデータに対して何も操作は行われません。

ボリュームの作成後は、必要に応じてボリュームにファイルシステムを作成し、そのファイルシステムをマウントして XLV 論理ボリュームを使用できるようにします。

注意 : mkfs を使ってファイルシステムを作成すると、ディスク・パーティションにある既存のデータはすべて破壊されます。

---

xlv_make は対話式で実行するか、または、入力ファイルからコマンドを実行できます。次に、*xlv_make* を使用する例を示します。最初の例は対話式、次の例は非対話式の例です。

例 1：簡単な XLV 論理ボリュームの作成

この例は、2 つのオプション・ディスク全体を使用して作成したデータ・サブボリュームで構成する簡単な XLV 論理ボリュームの作成方法です。ディスクは、コントローラ 0、ドライブ・アドレス 2 と 3 にあります。XFS ファイルシステムが作成され、*/vol 1* にマウントされます。

1. ボリューム内に使用するディスクがマウントされている場合は、そのディスクをアンマウントします。

```
# df
Filesystem                Type  blocks     use   avail %use  Mounted on
/dev/root                  efs 1939714  430115 1509599  22%  /
/dev/dsk/dks0d2s7          efs 2004550      22 2004528   0%  /d2
/dev/dsk/dks0d3s7          efs 3826812      22 3826790   0%  /d3
# umount /d2
# umount /d3
```

2. *xlv_make* を起動します。

```
# xlv_make
xlv_make>
```

3. ボリューム名を指定して、ボリュームの作成を開始します。たとえば、xlv0 を指定します。

```
xlv_make> vol xlv0
xlv0
```

4. データ・サブボリュームの作成を開始します。

```
xlv_make> data
xlv0.data
```

xlv_make は、作成する各オブジェクト（ボリューム、サブボリューム、プレックス、またはボリューム・エレメント）の名前を表示します。

5. プレックスを指定して、ボリューム階層の下に移動します。

```
xlv_make> plex
xlv0.data.0
```

6. ボリュームに含めるボリューム・エレメント（ディスク・パーティション）を指定します。たとえば、*/dev/dsk/dks0d2s7* と */dev/dsk/dks0d3s7* を指定します。

```
xlv_make> ve dks0d2s7
xlv0.data.0.0
xlv_make> ve dks0d3s7
xlv0.data.0.1
```

ディスク・パーティションのパス名の最後の部分（上記参照）、または完全なパス名を指定できます。*xlv_make* で使用できるディスク・パーティションのタイプは xlv、sfx、および efs です。**-force** オプションを指定すれば、その他のタイプ（lvol など）も使用できます。たとえば、*ve -force dks0d2s7* と指定します。*xlv_make* は、パーティション・タイプを自動的に xlv に変更します。

7. オブジェクトの指定が完了したことを示します。

```
xlv_make> end
Object specification completed
```

8. 指定したオブジェクトを確認します。

```
xlv_make> show

        Completed Objects
(1)  VOL xlv0
VE xlv0.data.0.0 [empty]
        start=0, end=2004549, (cat)grp_size=1
        /dev/dsk/dks0d2s7 (2004550 blks)
VE xlv0.data.0.1 [empty]
        start=2004550, end=5831361, (cat)grp_size=1
        /dev/dsk/dks0d3s7 (3826812 blks)
```

この出力は、1 つのボリュームに 2 つのボリューム・エレメントがあることを示しています。使用されている各パーティションのサイズは示されている通りで、一方は 2,004,550 ブロックです。このブロックはディスク・ブロックで、512 バイトです。

9. *xlv_make* を終了して、論理ボリューム・ラベルにボリューム情報を書込みます。

```
xlv_make> exit
Newly created objects will be written to disk.
Is this what you want?(yes)  yes
Invoking xlv_assemble
```

10. *mkfs* を使用して XFS を作成します。次に例を示します。

```
# mkfs /dev/xlv/xlv0
meta-data=/dev/xlv/xlv0          isize=256    agcount=8, agsize=16094 blks
data     =                       bsize=4096   blocks=2482901
log      =internal log           bsize=4096   blocks=1000
realtime =none                   bsize=4096   blocks=0, rtextents=0
```

11. ファイルシステムをマウントします。

```
# mkdir /vol1
# mount /dev/xlv/xlv0 /vol1
```

12. システムの起動時に論理ボリュームを自動的にマウントするには、該当するボリュームのエントリを */etc/fstab* に追加します。次に例を示します。

```
/dev/xlv/xlv0 /vol1 xfs rw,raw=/dev/rxlv/xlv0 0 0
```

例 2 : ストライプ化されたプレックス XLV 論理ボリュームの作成

次に示す例は、サイズの等しい 4 つのオプション・ディスク（コントローラ 0、ユニット 2 ～ 5）から XLV 論理ボリュームを非対話式で作成する方法を示しています。2 つのプレックスは、各プレックスの 2 つのディスク間でストライプ化されたデータから作成されます。ストライプ・ユニットは、128 KB です。XFS ファイルシステムを作成し、*/vol1* にマウントします。

1. 前の例で示したように、必要に応じて使用するディスクのファイルシステムをアンマウントします。

2. *xlv0.specs* という *xlv_make* の入力を含むファイルを作成します。次に、この例のボリューム xlv0 のファイルの内容を示します。

```
vol xlv0
data
plex
ve -stripe -stripe_unit 256 dks0d2s7 dks0d3s7
plex
ve -stripe -stripe_unit 256 dks0d4s7 dks0d5s7
end
show
exit
```

   このスクリプトによって、ボリュームが階層的に指定されます。つまり、ボリューム、サブボリューム（データ）、ストライプ化されたボリューム・エレメントを持つ最初のプレックス、ストライプ化されたボリューム・エレメントを持つ 2 番目のプレックスという階層になります。*ve* コマンドには、256 のストライプ・ユニット引数があります。この引数は、512 バイトのディスク・ブロック（セクタ）の数を示します。したがって、128 KB/512 = 256 となります。*end* コマンドは、指定が完了したことを示します。また、オプションの *show* コマンドで指定内容を表示できます。この論理ボリューム・ラベルは、*exit* コマンドによって作成されます。

3. *xlv_make* を実行してボリュームを作成します。

```
# xlv_make xlv0.specs
```

4. 10 MB の内部ログと 1 KB のブロック・サイズを持つ XFS ファイルシステムを作成します。

```
# mkfs -b size=1k -l size=10m /dev/xlv/xlv0
```

5. ファイルシステムをマウントします。

```
# mkdir /vol1
# mount /dev/xlv/xlv0 /vol1
```

6. システムの起動時に論理ボリュームを自動的にマウントするには、該当するボリュームのエントリを */etc/fstab* に追加します。

```
/dev/xlv/xlv0 /vol1 xfs rw,raw=/dev/rxlv/xlv0 0 0
```

例 3 : 外部ログを持つ XFS ファイルシステムのプレックス XLV 論理ボリュームの作成

次に示す例では、プレックス化されたログ・サブボリューム、および連結され、プレックス化されたデータ・サブボリュームを持つ XLV 論理ボリュームの作成方法を示します。このボリュームを使用して、外部ログを持つ XFS ファイルシステムを保持します。

この例では、コントローラ 1 のドライブ・アドレス 2 ～ 5 の 4 つのディスクを使用します。ドライブ・アドレス 2 と 3 のディスクは、xfslog パーティションを持つオプション・ドライブとしてパーティションに分割されています。ドライブ・アドレス 4 と 5 のディスクは、xfslog パーティションを持たないオプション・ドライブとしてパーティションが分割されています。

1. *xlv_make* を起動し、2 つのプレックスを持つログ・サブボリュームを作成して、*xfs-mp5* というボリュームの作成を開始します。

```
# xlv_make
xlv_make> vol xfs-mp5
xfs-mp5
xlv_make> log
xfs-mp5.log
xlv_make> plex
xfs-mp5.log.0
xlv_make> ve dks1d2s15
xfs-mp5.log.0.0
xlv_make> plex
xfs-mp5.log.1
xlv_make> ve dks1d3s15
xfs-mp5.log.1.0
```

2. 2 つのプレックスを持つデータ・サブボリュームを作成します。各プレックスには、それぞれ 2 つのボリューム・エレメントがあります。

```
xlv_make> data
xfs-mp5.data
xlv_make> plex
xfs-mp5.data.0
xlv_make> ve dks1d2s7
xfs-mp5.data.0.0
xlv_make> ve dks1d4s7
xfs-mp5.data.0.1
xlv_make> plex
xfs-mp5.data.1
xlv_make> ve dks1d3s7
xfs-mp5.data.1.0
xlv_make> ve dks1d5s7
xfs-mp5.data.1.1
```

3. オブジェクトの指定が完了したことを通知し、指定したオブジェクトを確認し、*xlv_make* を終了します。

```
xlv_make> end
Object specification completed
xlv_make> show

        Completed Objects
(1)  VOL xfs-mp5
VE xfs-mp5.log.0.0 [empty]
        start=0, end=8255, (cat)grp_size=1
        /dev/dsk/dks1d2s15 (8256 blks)
VE xfs-mp5.log.1.0 [empty]
        start=0, end=8255, (cat)grp_size=1
        /dev/dsk/dks1d3s15 (8256 blks)
VE xfs-mp5.data.0.0 [empty]
        start=0, end=3920223, (cat)grp_size=1
        /dev/dsk/dks1d2s7 (3920224 blks)
VE xfs-mp5.data.0.1 [empty]
        start=3920224, end=7848703, (cat)grp_size=1
        /dev/dsk/dks1d4s7 (3928480 blks)
VE xfs-mp5.data.1.0 [empty]
        start=0, end=3920223, (cat)grp_size=1
        /dev/dsk/dks1d3s7 (3920224 blks)
VE xfs-mp5.data.1.1 [empty]
        start=3920224, end=7848703, (cat)grp_size=1
        /dev/dsk/dks1d5s7 (3928480 blks)

xlv_make> exit
Newly created objects will be written to disk.
Is this what you want?(yes)  y
Invoking xlv_assemble
```

4. *mkfs* を実行して、XFS ファイルシステムを作成します。外部ログが提供されている場合は、*mkfs* によってその外部ログが自動的に使用されます。

```
# mkfs /dev/xlv/xfs-mp5
meta-data=/dev/xlv/xfs-mp5   isize=256    agcount=8, agsize=122636 blks
data     =                       bsize=4096   blocks=981088
log      =volume log             bsize=4096   blocks=1032
realtime =none                   bsize=65536  blocks=0, rtextents=0
```

5. ファイルシステムをマウントします。

```
# mkdir /v1
# mount /dev/xlv/xfs-mp5 /v1
```

6. システムの起動時に論理ボリュームを自動的にマウントするには、該当するボリュームのエントリを */etc/fstab* に追加します。

```
/dev/xlv/xfs-mp5 /v1 xfs rw,raw=/dev/rxlv/xfs-mp5 0 0
```

XLV 論理ボリューム・オブジェクトの表示

システムのトップ・レベルの XLV オブジェクトのリスト（ボリューム、アタッチされていないプレックス、およびアタッチされていないボリューム・エレメント）を取得するには、*xlv_mgr* を起動し、*show all* コマンドを実行します。

```
# xlv_mgr
xlv_mgr> show all
Volume Element: SPARE_VE
Volume:         BIG_VOLUME (complete)
```

この例では、トップ・レベルのオブジェクトが 2 つあります。1 つは SPARE_VE というボリューム・エレメントで、もう 1 つは BIG_VOLUME という XLV 論理ボリュームです。ボリューム・エレメントは、プレックスの一部ではないので（プレックスにアタッチされていないので）、トップ・レベルのオブジェクトです。ボリューム・エレメントは、後でプレックスにアタッチできます。

トップ・レベルのオブジェクトの完全な階層構造を表示するには、*xlv_mgr* コマンドを実行し、*show object* とオブジェクト名を入力します。

```
xlv_mgr> show object BIG_VOLUME
VOL BIG_VOLUME (complete)
VE BIG_VOLUME.log.0.0   [active]
        start=0, end=8255, (cat)grp_size=1
        /dev/dsk/dks1d2s15 (8256 blks)
VE BIG_VOLUME.log.1.0   [active]
        start=0, end=8255, (cat)grp_size=1
        /dev/dsk/dks1d3s15 (8256 blks)
VE BIG_VOLUME.log.2.0   [active]
        start=0, end=8255, (cat)grp_size=1
        /dev/dsk/dks1d4s15 (8256 blks)
VE BIG_VOLUME.data.0.0  [active]
        start=0, end=3920223, (cat)grp_size=1
        /dev/dsk/dks1d2s7 (3920224 blks)
VE BIG_VOLUME.data.1.0  [active]
        start=0, end=3920223, (cat)grp_size=1
        /dev/dsk/dks1d3s7 (3920224 blks)
VE BIG_VOLUME.data.2.0  [active]
        start=0, end=3920223, (cat)grp_size=1
        /dev/dsk/dks1d4s7 (3920224 blks)
```

この出力では、BIG_VOLUME にログ・サブボリュームとデータ・サブボリュームがあることを示しています。各サブボリュームには 3 つのプレックスがあり、各プレックスにはボリューム・エレメントが 1 つあります。

プレックスへのボリューム・エレメントの追加（XLV 論理ボリュームの拡張）

プレックスの最後にボリューム・エレメントを追加すると、XLV 論理ボリュームを拡張（サイズを増加）できます。すべてのプレックスにボリューム・エレメントを追加しない場合、追加されたボリューム・エレメントに格納されているデータはコピーされません。

次の手順を行う場合は、XLV 論理ボリュームから開始してください。単一のディスク・パーティションにあるファイルシステムから開始し、それを論理ボリュームに変更し、その論理ボリュームを別のディスク・パーティションへ拡張する場合は、第 6 章の「別のディスクへの XFS ファイルシステムの拡張」を参照してください。

1. ボリュームに追加するボリューム・エレメントがない場合は、*xlv_make* を使用して作成します。たとえば、この手順に従って、新しいディスク */dev/dsk/dks0d4s7* からボリューム・エレメントを作成します。

   ```
   # xlv_make
   xlv_make> ve spare_ve dks0d4s7
   new_ve
   xlv_make> end
   Object specification completed
   xlv_make> exit
   Newly created objects will be written to disk.
   Is this what you want?(yes)  yes
   Invoking xlv_assemble
   ```

   ve コマンドを使用して、ボリューム・エレメント名である spare_ve を作成します。ボリューム・エレメントは大規模階層の一部ではないので、名前を指定する必要があります。つまり、このボリューム・エレメントは、トップ・レベルのオブジェクトです。

2. *xlv_mgr* コマンドの **attach** コマンドを使用して、各ボリューム・エレメントを追加します。たとえば、ボリューム xlv0 のデータ・サブボリュームのプレックス 0 に、手順 1 で作成したボリューム・エレメントを追加するには、次のコマンドを実行します。

   ```
   # xlv_mgr
   xlv_mgr> attach ve spare_ve xlv0.data.0
   ```

3. *xlv_mgr* を終了します。

   ```
   xlv_mgr> quit
   ```

4. XFS ファイルシステムを拡張する場合、XFS ファイルシステムがまだマウントされていなければ、マウントします。

   # **mount** *volume mountpoint*

   volume は、論理ボリュームのデバイス名です。たとえば、*/dev/xlv/xlv0* です。また、*mountpoint* は、論理ボリュームのマウント・ポイント・ディレクトリです。

5. XFS ファイルシステムを拡張する場合は、*xfs_growfs* を使用します。

```
# xfs_growfs -d mountpoint
```

mountpoint は、論理ボリュームのマウント・ポイント・ディレクトリです。

6. EFS ファイルシステムを拡張する場合、EFS ファイルシステムがマウントされていればアンマウントし、*growfs* を使用して拡張します。

```
# umount mountpoint
# growfs volume
```

mountpoint は、ファイルシステムのマウント・ポイント・ディレクトリです。また、*volume* は、論理ボリュームのデバイス名です。たとえば、*/dev/xlv/xlv0* です。

XLV 論理ボリュームへのプレックスの追加

Disk Plexing Option ソフトウェア・オプションを購入し、そのソフトウェアの FLEX*lm* ライセンスもインストールしてある場合、プレックスを既存のサブボリュームに追加すれば、ディスクが破損した場合の信頼性を向上できます。次に、サブボリュームにプレックスを追加する手順を示します。複数のプレックスをサブボリュームに追加したり、プレックスをボリュームの各サブボリュームに追加するには、必要に応じてこの手順を繰返し行ってください。

1. サブボリュームに追加するプレックスがない場合は、*xlv_make* を使用して作成します。たとえば、plex1 というプレックスを作成し、root_vol というボリュームのデータ・サブボリュームに追加するには、次のコマンドを入力します。

```
# xlv_make
xlv_make> plex plex1
plex1
xlv_make> ve /dev/dsk/dks0d3s7
plex1.0
xlv_make> end
Object specification completed
xlv_make> exit
Newly created objects will be written to disk.
Is this what you want?(yes)  yes
Invoking xlv_assemble
```

2. *xlv_mgr* コマンドを使用して、ボリュームにプレックスを追加します。たとえば、root_vol にスタンドアロン・プレックス plex1 を追加するには、次のコマンドを実行します。

```
# xlv_mgr
xlv_mgr> attach plex plex1 root_vol.data
```

xlv_mgr は、自動的にプレックスの回復処理を初期化し、元のプレックス (root_vol.data.0) の内容を新しく追加されたプレックスにコピーします。

3. オブジェクト階層を表示して、root_vol に 2 つのプレックスがあることを確認します。

```
xlv_mgr> show object root_vol
```

```
VOL root_vol (complete)
VE root_vol.data.0.0    [active]
        start=0, end=988091, (cat)grp_size=1
        /dev/dsk/dks0d2s7 (988092 blks)
VE root_vol.data.1.0    [empty]
        start=0, end=988091, (cat)grp_size=1
        /dev/dsk/dks0d3s7 (988092 blks)
```

新しく追加したプレックス root_vol.data.1 は、新規に作成されたので、初期状態は空です。

4. *xlv_mgr* を終了します。

```
xlv_mgr> quit
```

プレックスの回復が完了すると、新しいプレックスは自動的にアクティブ状態に切替わります。その過程をチェックし、プレックスがアクティブ状態であることを確認するには、次の手順に従ってください。

1. 実行中の XLV デーモンをリスト表示します。

```
# ps -ef | grep xlv
    root    27     1  0 10:49:27 ?        0:00 /sbin/xlv_plexd -m 4
    root    35     1  0 10:49:28 ?        0:00 /sbin/xlv_labd
    root    31     1  0 10:49:27 ?        0:00 xlvd
    root   407    27  1 11:01:01 ?        0:00 xlv_plexd -v 2 -n root_vol.data
-d 50331648 -b 128 -w 0 0 1992629
    root   410   397  2 11:01:11 pts/0    0:00 grep xlv
```

xlv_plexd デーモンの例として、現在回復している root_vol.data があります。プレックスが完全に回復されると、このデーモンが終了します。

2. 後で XLV デーモンを再度チェックします。

```
# ps -ef | grep xlv
    root    27     1  0 10:49:27 ?        0:00 /sbin/xlv_plexd -m 4
    root    35     1  0 10:49:28 ?        0:00 /sbin/xlv_labd
    root    31     1  0 10:49:27 ?        0:03 xlvd
    root   459   397  2 11:21:10 pts/0    0:00 grep xlv
```

root_vol/data を回復していた *xlv_plexd* は、もう動作していません。したがって、プレックスの回復は完了しました。

3. *xlv_mgr* を使用して、プレックスの状態をチェックします。

```
# xlv_mgr
xlv_mgr> show object root_vol
VOL root_vol (complete)
VE root_vol.data.0.0    [active]
        start=0, end=988091, (cat)grp_size=1
        /dev/dsk/dks0d2s7 (988092 blks)
VE root_vol.data.1.0    [active]
        start=0, end=988091, (cat)grp_size=1
```

```
        /dev/dsk/dks0d2s0 (988092 blks)
xlv_mgr> quit
```

プレックスは両方とも、アクティブ状態になっています。

XLV 論理ボリュームからのプレックスの切離し

ディスク・ドライブをスワップする場合など、ボリュームからプレックスを切離すことがあります。これは、ボリュームがアクティブ状態でなければ実行できません。ただし、これは、残りのプレックスにあるアクティブなボリューム・エレメントが、サブボリュームのアドレス範囲全体を含んでいる場合にかぎり可能です。*xlv_mgr* コマンドでは、ボリューム内のほかのプレックスがアクティブではない場合に、アクティブなプレックスだけを切離すことはできません。プレックスの切離しを行うには、次の手順に従ってください。

1. *xlv_mgr* を開始し、切離すプレックスがあるボリュームを表示します。次に、root_vol の例を示します。

```
# xlv_mgr
xlv_mgr> show object root
VOL root (complete)
VE root.data.0.0        [active]
        start=0, end=1843199, (cat)grp_size=1
        /dev/dsk/dks1d3s0 (1843200 blks)
VE root.data.1.0        [active]
        start=0, end=1843199, (cat)grp_size=1
        /dev/dsk/dks1d4s0 (1843200 blks)
```

2. plex1 を切離して rplex1 という名前を与えるには、次のコマンドを実行します。

```
xlv_mgr> detach plex root.data.1 rplex1
```

3. ボリュームおよび切離したプレックスを調べるには、次のコマンドを実行します。

```
xlv_mgr> show -long all
PLEX rplex1
VE rplex1.0     [stale]
        start=0, end=1843199, (cat)grp_size=1
        /dev/dsk/dks1d4s0 (1843200 blks)

VOL root (complete)
VE root.data.0.0        [active]
        start=0, end=1843199, (cat)grp_size=1
        /dev/dsk/dks1d3s0 (1843200 blks)
```

4. *xlv_mgr* を終了します。

```
xlv_mgr> quit
```

XLV オブジェクトの削除

注意：ここでの手順は、正しく行わないとデータが消去されてしまうことがあります。したがって、この操作は IRIX システム管理者が行うことをお薦めします。

ボリュームまたはほかの XLV オブジェクトを削除するには、次に手順に従ってください。

1. ボリュームを削除する場合は、まず、ボリュームをアンマウントする必要があります。

   ```
   # umount /vol1
   ```

2. *xlv_mgr* を開始し、システムの各オブジェクトを表示します。

   ```
   # xlv_mgr
   xlv_mgr> show -long all
   VOL root_vol (complete)
   VE root_vol.data.0.0    [active]
           start=0, end=988091, (cat)grp_size=1
           /dev/dsk/dks0d2s0 (988092 blks)
   VE root_vol.data.1.0    [active]
           start=0, end=988091, (cat)grp_size=1
           /dev/dsk/dks0d2s7 (988092 blks)
   ```

   この例では、データ・サブボリュームに 2 つのプレックス（root_vol.data.0 と root_vol.data.1）を持つボリュームを示します。各プレックスにはボリューム・エレメントが 1 つあります。

3. 削除するエレメントが、高レベルのオブジェクトではない場合、まず高レベルのオブジェクトからボリューム・エレメントを切離します。たとえば、プレックスの 1 つを削除するには、最初にそのプレックスを切離す必要があります。

   ```
   xlv_mgr> detach plex root_vol.data.1 plex_to_be_deleted
   ```

   切離すオブジェクトの名前を指定してください。この例では、plex_to_be_deleted と指定します。

4. オブジェクト plex_to_be_deleted を削除します。

   ```
   xlv_mgr> delete object plex_to_be_deleted
   ```

5. オブジェクトが削除されたことを確認します。

   ```
   xlv_mgr> show -long all
   VOL root_vol (complete)
   VE root_vol.data.0.0    [active]
           start=0, end=988091, (cat)grp_size=1
           /dev/dsk/dks0d2s0 (988092 blks)
   ```

6. *xlv_mgr* を終了します。

```
xlv_mgr> quit
```

プレックスの削除とマウント

注意 : ここでの手順は、正しく行わないとデータが消去されてしまうことがあります。したがって、この操作は IRIX システム管理者が行うことをお薦めします。

プレックス化されたボリュームからプレックスを削除して、そのプレックスを別々にマウントすることによって、ファイルシステムのスナップショットを取得できます。ボリュームしかマウントできないので、プレックスはボリュームに変換しておく必要があります。次の手順では、元のボリュームからプレックスを削除し、それを別のボリュームに追加する方法を説明します。

1. 現在ボリュームが回復されていないことを確認します。回復している場合は、プレックスの回復が終了するまでプレックスのデータは異なるので、回復が終了するまで待ってください。

   ```
   # ps -ef | grep xlv_plexd
       root    35     1  0   Dec 13 ?        0:00 /sbin/xlv_plexd -m 4
   ```

   出力から、マスター・プロセスである *xlv_plexd* のコピーだけが動作していることがわかります。複数のコピーが動作している場合は、プレックスの回復中です。

2. 論理ボリュームにマウントされているファイルシステムをアンマウントします。この例では、*/projvol5* をアンマウントします。

   ```
   # umount /projvol5
   ```

   ファイルシステムをアンマウントすると、ファイルシステムが clean 状態になります。

3. *xlv_mgr* を開始し、論理ボリュームを表示します。この例では、xfs-mp5 を表示します。

   ```
   # xlv_mgr
   xlv_mgr> show object xfs-mp5
   VOL xfs-mp5 (complete)
   VE xfs-mp5.log.0.0 [active]
           start=0, end=8255, (cat)grp_size=1
           /dev/dsk/dks1d2s15 (8256 blks)
   VE xfs-mp5.log.1.0 [active]
           start=0, end=8255, (cat)grp_size=1
           /dev/dsk/dks1d3s15 (8256 blks)
   VE xfs-mp5.data.0.0 [active]
           start=0, end=3920223, (cat)grp_size=1
           /dev/dsk/dks1d2s7 (3920224 blks)
   VE xfs-mp5.data.0.1 [active]
           start=3920224, end=7848703, (cat)grp_size=1
           /dev/dsk/dks1d4s7 (3928480 blks)
   VE xfs-mp5.data.1.0 [active]
   ```

```
        start=0, end=3920223, (cat)grp_size=1
        /dev/dsk/dks1d3s7 (3920224 blks)
VE xfs-mp5.data.1.1 [active]
        start=3920224, end=7848703, (cat)grp_size=1
        /dev/dsk/dks1d5s7 (3928480 blks)
```

4. ログ・サブボリュームから 2 番目のプレックスを切離し、それを log_copy とします。

   ```
   xlv_mgr> detach plex xfs-mp5.log.1 log_copy
   ```

   ボリュームには必ずデータ・サブボリュームとそれに対応するログ・サブボリュームがあります。このため、データ・プレックスを切離すときはログ・サブボリュームにあるプレックスも切離す必要があります。

5. データ・サブボリュームから 2 番目のプレックスを切離し、それを data_copy とします。

   ```
   xlv_mgr> detach plex xfs-mp5.data.1 data_copy
   ```

6. 高レベルのオブジェクトをすべて表示して、ボリュームが 1 つとプレックスが 2 つになったことを確認します。

   ```
   xlv_mgr> show all
   Volume:         xfs-mp5 (complete)
   Plex:           log_copy
   Plex:           data_copy
   ```

7. 切離された各プレックスに対して、*delete* コマンドを実行します。

   ```
   xlv_mgr> delete object log_copy
   Object log_copy deleted.

   xlv_mgr> delete object data_copy
   Object data_copy deleted.
   ```

   delete コマンドは、ボリューム・ヘッダにある XLV 論理ボリューム情報を変更します。ただし、パーティションのデータに対しては、何の操作も行われません。

8. *xlv_mgr* を終了します。

   ```
   xlv_mgr> quit
   ```

9. 切離したプレックスから、ボリュームにパーティションを作成します。

   ```
   # xlv_make
   xlv_make> vol copy
   copy
   xlv_make> log
   copy.log
   xlv_make> ve dks1d3s15
   copy.log.0.0
   xlv_make> data
   copy.data
   xlv_make> ve dks1d3s7
   copy.data.0.0
   ```

```
xlv_make> ve dks1d5s7
copy.data.0.1
xlv_make> end
Object specification completed
xlv_make> exit
Newly created objects will be written to disk.
Is this what you want?(yes)  yes
Invoking xlv_assemble
```

10. 新しいボリュームをマウントします。ファイルシステムに対して何も操作が行われていないので、*mkfs* は使用されません。*mkfs* を使用するとデータが消去されます。

```
# mkdir /copy
# mount /dev/xlv/copy /copy
```

11. 元のファイルシステムを再度マウントします。

```
# mount /dev/xlv/xfs-mp5 /projvol5
```

12. *ls* コマンドを使用して、元のボリュームにあるファイルが、削除されたプレックスから作成した新しいボリュームにもあることを確認します。

```
# ls /copy
autoconfig  chroot      config      cron.d
chkconfig   clri        cron        fstab
# ls /projvol5
autoconfig  chroot      config      cron.d
chkconfig   clri        cron        fstab
```

ディスクのプレックス化したボリュームへの置換

ここでは、プレックス化されたボリューム・エレメントの一部を含むディスクを置換する場合の手順の概要について説明します。例として、プレックス化されたボリューム・エレメントに使用される Origin Vault エンクロージャ・ディスクを使用します。他のディスク構成を使う場合は、XLV コマンドは同じですが、ディスクを物理的に置換するための手順が異なります。

ディスクをプレックス化したボリュームへ置換するには、次の手順に従います。

1. 破損したディスクを使用して XLV からボリューム・エレメントを削除します。
2. ディスク・ドライブを物理的に置換します。
3. 新しいドライブを使用して XLV ボリューム・エレメントを再度作成します。

XLV からのボリューム・エレメントの削除

ここでは、Origin Vault エンクロージャを例として説明します。この例では、破損ディスクは Origin Vault1 (dks2d6s7) のドライブ ID 6 であり、vol 2 (plex 0) にあります。また、2 つのプレックスがあり、各プレックスが単一のボリューム・エレメントのみを所有していることを前提とします。ここで示すコマンドは、特定のディスク障害用のサンプルです。

1. 破損ディスクを含むプレックス（またはボリューム・エレメント）をボリューム（この例では vol 2）から削除します。このコマンド・シーケンスでは、プレックスを切離し、badplex という名前に変更します。

   ```
   # xlv_mgr
   xlv_mgr> detach plex vol2.data.0 badplex
   ```

   正しく削除できた場合には、第 4 章の「物理的なディスク・ドライブの置換」へ進み、そこで説明されている手順に従ってください。破損ディスクが応答せず、プレックスの切離しに失敗した場合は、手順 2 に進んでください。

2. 次のコマンドを実行します。**-force** オプションは、親オブジェクトに欠落しているものがある場合に切離しを実行します。

   ```
   xlv_mgr> detach -force plex vol2.data.0 badplex
   xlv_mgr> delete object badplex
   ```

   この時点で正しく削除できた場合には、第 4 章の「物理的なディスク・ドライブの置換」へ進み、そこで説明されている手順に従ってください。破損ディスクが応答せず、プレックスの切離しに失敗した場合は、手順 3 に進んでください。

3. ファイルシステムをアンマウントし、ファイルを開いているプロセスを削除 (kill) します。

   ```
   # unmount -k /fs2
   ```

4. *xlv_mgr script* コマンドに **-write** オプションを使用して、ボリュームの構成内容を保存します。ボリュームの構成内容は、手順 6 で説明するボリュームの再作成で必要になります。

   xlv_mgr script コマンドを実行すると、ボリュームを作成するときに必要な xlv_make(1M) コマンドが表示されます。詳細については、xlv_mgr(1M) マン・ページを参照してください。**-write** オプションを使用すると、指定したファイルにコマンドを保存できます。コマンドの一覧を別途保存する場合は、このオプションは不要です。

   xlv_mgr がディスクの XLV ラベルを完全に読取れない場合には、*script* コマンドが機能しない可能性があります。この場合、通常のバックアップやメンテナンス操作で保存しておいたボリュームの構成情報を使用します。

5. ボリューム・オブジェクトを削除します。

   ```
   xlv_mgr> delete object vol2
   ```

6. 破損ディスクを使用しないでボリュームを再作成します。

この例では、次のコマンド・シーケンスを使用してボリューム v2 が作成されました。

```
# xlv_make
xlv_make> vol2
xlv_make> data
xlv_make> plex
xlv_make> ve dks2d6s7
xlv_make> plex
xlv_make> ve dks3d6s7
xlv_make> end
xlv_make> exit
```

破損ディスクを使用しないでボリュームを再作成するには、次のコマンド・シーケンスを実行します。

```
# xlv_make
xlv_make> vol2
xlv_make> data
xlv_make> plex
xlv_make> ve dks3d6s7
xlv_make> end
xlv_make> exit
```

物理的なディスク・ドライブの置換

次の手順に従って、OriginVault エンクロージャのディスク・ドライブを置換してください。このとき、電源を切り、ディスクを置換する間、バスが動作しないことを確認する必要があります。バスが動作中にディスクを挿入するとデータの破損の原因になります。

1. 破損ドライブでエンクロージャを確認してください（この例では Origin Vault 1）。
2. 電源を切り、Origin Vault 1 にします。
3. 10 秒間待ちます。破損ドライブがさらに損傷を受けないためにも、この待ち時間は重要です。
4. 物理的に破損ディスクを取除き、代りのディスクをインストールします。
5. Origin Vault 1 を起動します。

Origin Vault 1 の電源が切れている間に Origin Vault 2 で vol1 への I/O 書込みが行われた場合には、vol1 をアップデートする必要があります。*xlv_mgr* を使用して、vol 1 をアップデートする必要があるかどうかを確認してください。または、次のコマンドを使用して、[*offline*] にしてください。

```
xlv_mgr> show kernel
```

出力で [*offline*] が表示された場合には、Origin Vault 1 のディスク ID 5 に古いデータが存在しています。vol1 の一部が [*offline*] の場合には、*xlv_mgr* を使用して、影響を受けたボリューム・エレメントをオンラインに戻してください。

この例では、停電によりドライブ dks2d5s7 が [*offline*] になっています。このドライブは、volume 1 の plex 0 です。次のコマンドを実行してください。

```
xlv_mgr> change online vol1.data.0.0
```

ディスク・ドライブを置換するには、warm-plug 機能を使用することもできます。warm-plug 機能は、ここで例として使用した Origin Vault エンクロージャ以外のシステム・キャビネットに設置されたディスクでも使用可能です。この機能の詳細については、scsiadminswap(1M)、scsihotswap(1M)、および scsiquiesce(1M) マン・ページを参照してください。

新しいドライブを使用した XLV ボリューム・エレメントの再作成

次の手順に従って、復元する XLV ボリューム・エレメントを使用して置換ドライブを作成してください。

1. fx(1M) コマンドを使用して、新しいドライブのパーティションを分割します。新しいドライブのパーティションは、破損ドライブとまったく同じように再分割してください。
2. 新しいディスク・ドライブにプレックス（ボリューム・エレメント）を作成します。

   ```
   # xlv_make
   xlv_make> plex newplexname
   xlv_make> ve dks2d6s7
   ```

3. プレックスをボリュームにアタッチします。あるいは、ve をボリュームにアタッチまたは挿入します。

   ```
   # xlv_mgr
   xlv_mgr> attach plex newplexname vol2.data
   ```

Root 用のプレックス XLV 論理ボリュームの作成

注意：ここでの手順は、正しく行わないとデータが消去されてしまうことがあります。したがって、この操作は IRIX システム管理者が行うことをお薦めします。

プレックス化されたボリュームに root ファイルシステムを格納すると、信頼性を向上できます。プレックス化された root ボリュームを使用すると、ルート・ディスクの 1 つが破損しても、シス

テムの動作を保持できます。独立した usr ファイルシステムがシステム・ディスクにある場合は、usr ファイルシステムもプレックス化する必要があります。ルート・ディスクが破損すると、スワップ・パーティションが使用不能になることがあるので、別のディスクにスペアのスワップ・パーティションを使用可能にしておくことが必要です。ボリュームで使用されている各ディスクが同じで、パーティションが同じように分割されている場合は、root ボリュームおよび、usr ボリューム（ある場合）とスワップ・パーティションのプレックスの管理が一番簡単です。

データ・サブボリュームのみ root ボリュームにあります。また、単一のボリューム・エレメントのみデータ・サブボリュームの各プレックスにあります。ボリューム・エレメントには、必ず単一のディスク・パーティションがあります。

root ファイルシステムは、内部ログを持つ EFS ファイルシステムか XFS ファイルシステムのどちらかになります。

次の手順に従って、プレックス化された root ボリュームを作成します。この手順では、システム・ディスクを空のシステムではなく、作業システムから起動してください。

1. XLV ボリュームに root パーティションを追加します。この例では、XLV ボリュームは xlv_root と呼ばれます。

   ```
   # xlv_make
   xlv_make> vol xlv_root
   xlv_root
   xlv_make> data
   xlv_root.data
   xlv_make> ve -force /dev/dsk/dks0d1s0
   xlv_root.data.0.0
   xlv_make> end
   Object specification completed
   xlv_make> exit
   Newly created objects will be written to disk.
   Is this what you want?(yes)  yes
   Invoking xlv_assemble
   ```

   root パーティションを含む xlv_root という XLV ボリュームが作成されます。XLV は、パーティションのデータをそのまま保持するので、root パーティションの内容も保持されます。マウントされたパーティションがボリュームに含まれていたため、**-force** オプションを指定した *ve* コマンドが使用されています。

2. システムを再起動して、*/dev/dsk/dks0d1s0* の root パーティションから論理ボリューム */dev/xlv/xlv_root* に切替えます。

   ```
   # reboot
   ```

3. */dev/root* と */dev/xlv/xlv_root* のメジャー・デバイス番号とマイナー・デバイス番号を比較して、root ボリュームが使用されていることを確認できます。

```
# ls -l /dev/root /dev/xlv/xlv_root
brw-------    2 root     sys      192,  0 10月 31日 17時58分 /dev/root
brw-------    2 root     sys      192,  0 12月 12日 17時58分 /dev/xlv/xlv_root
```

4. たとえば、*/dev/dsk/dks0d2s0* から 2 番目のプレックスを作成し、そのプレックスを root_plex1 と呼びます。

```
# xlv_make
xlv_make> plex root_plex1
root_plex1
xlv_make> ve /dev/dsk/dks0d2s0
root_plex1.0
xlv_make> end
Object specification completed
xlv_make> exit
Newly created objects will be written to disk.
Is this what you want?(yes)  yes
Invoking xlv_assemble
```

5. 代替プレックスに使用するディスクのボリューム・ヘッダに *sash* を追加します。この結果、主要プレックスが破損しても、代替プレックスから起動できるようになります。

```
# dvhtool -v get sash /tmp/sash /dev/rdsk/dks0d1vh
# dvhtool -v add /tmp/sash sash /dev/rdsk/dks0d2vh
```

6. *xlv_mgr* を使用してボリュームに 2 番目のプレックスをアタッチし、*xlv_mgr* を終了します。

```
# xlv_mgr
xlv_mgr> attach plex root_plex1 xlv_root.data
xlv_mgr> quit
```

シェル・プロンプトが返されると、システムは自動的にプレックスの回復を開始します。この結果、2 つのプレックスが同じデータを持つようになります。

代替プレックスからのシステムの起動

root ボリュームをプレックス化すると、主要プレックスが使用不能になった場合に代替プレックスから起動できます。ブート PROM は XLV 論理ボリュームを認識しないので、手作業でシステムの再設定を行い、代替プレックスがあるディスクからシステムを起動する必要があります。代替プレックスからシステムを起動させるための手順は、ワークステーションまたはサーバのモデルによって異なります。次に、第 4 章の「CHALLENGE L、CHALLENGE XL、および CHALLENGE DM」のシステムの場合の手順を示します。Origin2000 サーバーを含むほかのすべてのワークステーションとサーバーについては、第 4 章の「その他のモデル」の手順に従ってください。

CHALLENGE L、CHALLENGE XL、および CHALLENGE DM

CHALLENGE L、CHALLENGE XL、CHALLENGE DM システムでは、ダイアルまたはスイッチを使用して、ディスクのドライブ・アドレスを変更できます。システム・ディスクと代替ディスクが同じチャネルの内部ディスクで、パーティションが同じように分割されている場合は、2 つのディスクのドライブ・アドレスをスワップできます。システムがこの条件を満たさない場合は、第 4 章の「その他のモデル」で示す手順に従ってください。システム・ディスクと代替ディスクのドライブ・アドレスを切替えると、システムが自動的に代替ディスクから起動し、代替ディスクが新しいシステム・ディスクになります。次の手順に従ってください。

1. 次のコマンドを使用して、システムを停止します。

   ```
   # shutdown
   ```

2. システムの電源を切ります。
3. システム・ディスクと代替ディスクにあるスイッチまたはダイアルを操作して、各ディスクのドライブ・アドレスを別のドライブ・アドレスに変更します。
4. システムの電源を入れます。

その他のモデル

次の手順では、代替 root プレックスからシステムを起動する方法を説明します。この手順はすべてのシステムで使用できます。ここでは、システムをパーティション */dev/dsk/dks0d2s0* から起動するように再設定し、パーティション */dev/dsk/dks0d2s1* をスワップ用に使用します。実際にこの操作を行うときは使用しているシステム用に実際のパーティションを指定してください。

1. 「System Maintenance」メニューの「Enter Command Monitor」を選択します。

   ```
   ...
   5) Enter Command Monitor

   Option? 5
   Command Monitor.  Type "exit" to return to the menu.
   ```

2. PROM 環境変数を表示します。

   ```
   >> printenv
   SystemPartition=dksc(0,1,8)
   OSLoadPartition=dksc(0,1,0)
   root=dks0d1s0
   ...
   ```

   これ以降に表示されるべきスワップ PROM 環境変数は、NVRAM に保存されていないので表示されません。

3. SystemPartition、OSLoadPartition、および root 環境変数をリセットして、それに代替プレックスを含むディスク・パーティションの値を設定します。また、スワップ環境変数をリセットして、それに代替スワップ・パーティションの値を設定します。

```
>> setenv SystemPartition dksc(0,2,8)
>> setenv OSLoadPartition dksc(0,2,0)
>> setenv root dks0d2s0
>> setenv swap /dev/dsk/dks0d2s1
```

4. Command Monitor を終了し、システムを再起動します。

```
>> exit
...
Option? 1
                Starting up the system...
...
```

XLV 論理ボリュームを 11 個以上使用する場合のシステム設定

デフォルトでは、システムは最大 10 個の XLV 論理ボリュームをサポートしています。サポートする XLV 論理ボリューム数を増やすには、*/var/sysgen/master.d/xlv* を編集します。次に、その手順を示します。

1. エディタを使用して、*/var/sysgen/master.d/xlv* ファイルを開きます。

```
# vi /var/sysgen/master.d/xlv
```

2. ファイル内の以下の行を検索します。

```
#define XLV_MAXVOLS 10
```

3. 検索した行の 10 をより大きな値に変更します。

```
#define XLV_MAXVOLS 20
```

4. このファイルを保存し、エディタを終了します。
5. 新しいカーネルを生成します。

```
# /etc/autoconfig
```

6. システムを再起動して変更内容を有効にします。

```
# reboot
```

lv 論理ボリュームの XLV 論理ボリュームへの変換

ここでは、*lv* 論理ボリュームを XLV 論理ボリュームに変換するための手順を説明します。変換中は、論理ボリュームにあるファイルを編集したりダンプしたりすることはできません。この手順は、特権ユーザとして行ってください。

1. 必要であれば、論理ボリュームの新しい名前を選択します。*lv* とは異なり、XLV では、ピリオドを除いた有効な名前を必要とするので、意味を持つ名前を選択できます。たとえば、使用するマウント・ポイントの名前をボリューム名にすることができます。*/a*、*/b*、および */c* に論理ボリュームをマウントする場合は、XLV ボリューム名をそれぞれ a、b、および c にできます。
2. XLV 論理ボリュームに変換するすべての *lv* 論理ボリュームをアンマウントします。

   ```
   # umount /a
   ```

3. *lv_to_xlv* を使用して、*xlv_make* の入力スクリプトを作成します。

   ```
   # lv_to_xlv -o scriptfile
   ```

   scriptfile は、*lv_to_xlv* が作成するテンポラリ・ファイル名です。たとえば、*/usr/tmp/xlv.script* です。このファイルには、*/etc/lvtab* に列挙されている *lv* 論理ボリュームに相当する XLV ボリュームを作成するための一連の *xlv_make* コマンドがあります。
4. ボリューム名を変更する場合は、*scriptfile* を編集し、vol で始まる行にある名前を新しい名前にします。

   ```
   vol lv0
   ```

   この名前を次のようにします。

   ```
   vol a
   ```

   ボリューム名は任意の有効な名前にできます。
5. デフォルトでは、システムの *lv* 論理ボリュームすべてが XLV に変換されます。すべての *lv* 論理ボリュームを XLV に変換しない場合は、*scriptfile* を編集して、変更しないボリュームに対する *xlv_make* コマンドを削除します。詳細については、第 4 章の「xlv_make によるボリューム・オブジェクトの作成」、および xlv_make(1M) マン・ページを参照してください。
6. *scriptfile* を入力し、*xlv_make* コマンドを実行して XLV ボリュームを作成します。

   ```
   # xlv_make scriptfile
   ```

7. すべての *lv* 論理ボリュームを XLV に変換した場合は、*/etc/lvtab* を削除します。

   ```
   # rm /etc/lvtab
   ```

   すべての *lv* 論理ボリュームを XLV に変換しなかった場合は、*/etc/lvtab* を開いて、変換した論理ボリュームのエントリを削除します。

```
# vi /etc/lvtab
```

8. 起動時に XLV 論理ボリュームが自動的にマウントされるように */etc/fstab* を編集します。*/etc/fstab* の変更は各 XLV 論理ボリュームに必要です。

   - 最初のフィールドには、*/dev/dsk* の後にサブディレクトリ *xlv* を挿入します。
   - ボリューム名を変更した場合は、最初のフィールドを変更します。たとえば lv0 から a に変更します。
   - ロー・デバイス名にサブディレクトリ *xlv* を挿入します。
   - ボリューム名を変更する場合は、ロー・デバイスを変更します。たとえば lv0 から a に変更します。

   元の行は、次のとおりです。

   ```
   /dev/dsk/lv0   /a efs rw,raw=/dev/rdsk/lv0 0 0
   ```

   名前を変更した行は、次のとおりです。

   ```
   /dev/xlv/a /a efs rw,raw=/dev/rxlv/a 0 0
   ```

9. XLV 論理ボリュームをマウントします。

   ```
   # mount /a
   ```

XLV 論理ボリューム構成のレコードの作成

XLV オブジェクト、ボリューム、サブボリューム、プレックス、およびボリューム・エレメントに関する情報は、XLV オブジェクトを含む各ディスクのボリューム・ヘッダ内の論理ボリューム・ラベルに格納されています。詳細については、第 1 章の「ボリューム・ヘッダ」を参照してください。XLV 論理ボリューム・ラベルが削除されると、論理ボリューム・ラベルに記述されているオブジェクトにデータがまだ存在していても、システムはその論理ボリューム・ラベルを含む論理ボリュームを組立てることができません。ただし、該当する論理ボリュームの設定を覚えているかぎり *xlv_make* を使用して、論理ボリューム・ラベルを再度作成できます。*xlv_mgr* コマンドを使用すれば、システムにある各論理ボリュームの正確な設定を記録するスクリプトを作成できます。システムに XLV 論理ボリュームを再度作成するためにこのスクリプトが必要な場合は、後から *xlv_make* に対してこのスクリプトを指定できます。

システムにある各 XLV 論理ボリュームの正確な設定のレコードを作成するには、次の手順に従ってください。

1. *script* コマンドを実行して、画面上のテキストを検索し、そのテキストを */var/config/XLV.configuration* ファイルに書込みます。

   ```
   # script /var/config/XLV.configuration
   Script started, file is XLV.configuration
   ```

2. *xlv_mgr* を起動します。

```
# xlv_mgr
```

3. *xlv_mgr* に対して、設定情報を含むファイル名（たとえば */var/config/XLV.configuration*）を指定して、*script-write* コマンドを実行します。

```
xlv_mgr> script -write /var/config/XLV.configuration
```

4. *xlv_mgr* を終了します。

```
xlv_mgr> quit
```

5. 設定が記録されているファイルの内容をチェックします。

```
# cat /var/config/XLV.confguration
#
# Create Volume proj_vol
#
vol proj_vol
data
plex
ve   -force -start 0 /dev/dsk/dks1d3s11 /dev/dsk/dks1d3s12
plex
ve   -force -start 0 /dev/dsk/dks1d6s2 /dev/dsk/dks1d6s3
end
exit
```

ファイルシステムの概念

この章では、IRIX システムのファイルとディレクトリで構成されているハードディスク・ファイルシステムの重要な概念について説明します。ここでは、IRIX の主なファイルシステムである XFS (eXtent File System) ファイルシステム、およびその他のディスクのファイルシステムについて説明します。また、IRIX ディレクトリの編成、ファイルシステムの機能、ファイルシステムの種類、ファイルシステムの作成、ファイルシステムのマウントとアンマウント、ファイルシステムの整合性チェックなど、ファイルシステムを管理する上での重要な概念についても説明します。

メモ：CXFS ファイルシステムおよびサポートしているクラスタ環境については、『CXFS Software Installation and Administration Guide』を参照してください。

この章では、次の項目について説明します。

- 「ファイルシステムの再編成」(116 ページ)
- 「ミニルートからのファイルシステムの管理」(117 ページ)
- 「ファイルシステム領域の追加方法」(117 ページ)
- 「ディスクの割当て」(119 ページ)
- 「ファイルシステムの破損」(120 ページ)

UNIX ファイルシステムの基本的な概念を十分理解している場合も次の節を読んでください。IRIX の XFS ファイルシステムはほかの UNIX ファイルシステムとは内部の構造が多少異なり、管理用のコマンドや管理手順も異なります。

ファイルシステムの管理手順については、第 6 章「ファイルシステムの作成と拡張」と 第 7 章「ファイルシステムの保守」で説明します。

フロッピーと CD-ROM のファイルシステムの詳細については、『IRIX Admin: Peripheral Devices』を参照してください。

IRIX のディレクトリ編成

IRIX システム・ディスクには、必ず標準ディレクトリがあります。これらのディレクトリには、機能別に編成されたオペレーティング・システム・ファイルがあります。ファイルの編成方法は、それぞれの IRIX のバージョンにおける手法を基に自然に変化してきたので、論理的な構造にはなっていません。表 5-1 では、ほとんどのシステムで使用されている標準ディレクトリを示します。また、いくつかの標準ディレクトリについては、別名も示します。通常、別名は標準ディレクトリの古いパス名で、IRIX ディレクトリの編成が変わっても新しいパス名に移行しやすいようにシンボリック・リンクとして設けられています。

表 5-1 標準ディレクトリとその内容

ディレクトリ	別名	内容
/		IRIX カーネル (/unix)、root ログイン用のログイン・ファイル、およびその他のすべてのサブディレクトリを含む ルート・ディレクトリ
/CDROM		*mediad* デーモンで使用される CD-ROM のマウント・ポイント
/dev		端末、ディスク、テープ・ドライブ、CD-ROM ドライブなどのデバイス・ファイル

表 5-1 標準ディレクトリとその内容 **（続き）**

ディレクトリ	別名	内容
/dev/fd		ファイル記述子ファイルシステム
/etc		重要なシステム設定ファイルとメンテナンス・コマンド
/etc/config	/var/config、/usr/var/config	/etc/init.d のスクリプト用のシステム設定ファイル
/etc/init.d		システムの初期化の際に実行するスクリプト（/etc/rco.d および /etc/re2.d のディレクトリも同様）
/hosts		*autofs* でマウントした NFS ファイルシステムのデフォルトのマウント・ポイント
/hw		ハードウェア・グラフ・ファイルシステム
/lib		重要なコンパイラのバイナリとライブラリ
/lib32		重要なコンパイラのバイナリとライブラリ
/lib64		64 ビットのシステム（IP19、IP21、IP25、IP26、IP27、IP28、および IP30）用の重要なコンパイラのバイナリとライブラリ
/lost+found		*xfs_repair* コマンドおよび *fsck* コマンドを使用して回復したファイルの保管領域（/lost+found ディレクトリは、マウントされた XFS ファイルシステムおよび EFS ファイルシステムのルートにもあり）
/ns		*nsd* デーモンでサポートされた統一ネーム・サービス (UNS: Unified Name Service) の擬似ファイルシステムのインタフェースのデフォルトのマウント・ポイント
/opt		サードパーティのソフトウェア用のインストール先
/proc	/debug	プロセス（デバッグ）ファイルシステム
/sbin		システムの動作に最低限必要なコマンド
/stand		スタンドアロン・ユーティリティ（*fx*）
/tmp		テンポラリ・ファイル
/tmp_mnt		自動的にマウントされるファイルシステムのマウント・ポイント

表 5-1 標準ディレクトリとその内容 **（続き）**

ディレクトリ	別名	内容
/usr		いくつかのシステムにおけるファイルシステムのマウント・ポイント
/usr/bin	/bin	コマンド
/usr/bin/xll		最も標準的な X ウィンドウ・システムの実行権
/usr/bsd		コマンド
/usr/demos		デモ・プログラム
/usr/etc		重要なシステム設定ファイルとメンテナンス・コマンド
/usr/freeware		サポートされていないフリー・ソフトウェアの保管場所
/usr/gnu		GNU ユーティリティ
/usr/include		C ヘッダ・ファイル
/usr/lib		ライブラリとサポートされるファイル
/usr/lib32		ライブラリとサポートされるファイル
/usr/lib32/internal		SGI 社が出荷するプログラムで使用するダイナミック・シェア・オブジェクト (DSO: dynamic shared object)。コンパイル用には使用されません。
/usr/lib64		64 ビットのシステム（IP19、IP21、IP25、IP26、IP27、IP28、および IP30）用のライブラリとサポートされるファイル
/usr/lib64/internal		SGI 社が出荷するプログラムで使用するダイナミック・シェア・オブジェクト。 コンパイル用には使用されません。
/usr/Motif-1.2		Motif 1.2 に固有なバイナリ、ヘッダ、および lib
/usr/people		ホーム・ディレクトリ
/usr/relnotes		リリース・ノート
/usr/sbin		コマンド
/usr/share		さまざまなアプリケーション用の共有データ・ファイル。読込み専用で NFS を通してマウントできます。

表 5-1 標準ディレクトリとその内容 **（続き）**

ディレクトリ	別名	内容
/usr/share/Insight		InSight ブック
/usr/share/catman		リファレンス・ページ（マン・ページ）
/usr/var		/ と /usr が別のファイルシステムである場合に存在
/var		よく編集される、またはマシン固有のシステム・ファイル
/var/X11		X11 設定ファイル
/var/adm	/usr/adm	システム・ログ・ファイル
/var/inst		ソフトウェア・インストールの履歴
/var/inst/patchbase		パッチに置換えられたオリジナルのインストール済ファイル
/var/mail	/usr/mail	受信メール
/var/ns		プロトコルに固有のダイナミック・シェア・オブジェクトおよび nsd デーモンのキャッシュ・ファイル
/var/preserve	/usr/preserve	テンポラリ・エディタ・ファイル
/var/spool	/usr/spool	プリンタ・サポート・ファイル
/var/tmp	/usr/tmp	テンポラリ・ファイル
/var/yp		NIS コマンド

一般的なファイルシステムの概念

ファイルシステムとは、ディスク・パーティションのファイルやディレクトリを検索しやすいように編成されたデータ構造のことです。1 つのディスク・パーティションには、ファイルシステムは 1 つだけです。

ファイルは、ほかの構造は含まない 1 次元のバイト配列です。各ファイルの情報は、*i* ノードと呼ばれる構造体に格納されています。i ノードについては、第 5 章の「i ノード」で説明します。ファイルはファイルシステム間にまたがることはできません。

ディレクトリとは、ファイルやほかのディレクトリを格納するコンテナです。ユーザが使用できる別の種類のファイルで、書込みは不可です。ディレクトリの書込みは、オペレーティング・シ

ステムによって制御されます。また、ディレクトリは複数のファイルシステムにまたがることはできません。ファイルシステムは、ディレクトリとファイルの組合わせで構成されています。

ファイルシステムは、まず名前のないディレクトリから始まります。このディレクトリが特別なファイルシステムであるルート・ディレクトリになります。IRIX オペレーティング・システムでは、常に root ファイルシステムという名前のファイルシステムが 1 つあります。root ファイルシステムのルート・ディレクトリは、単一のスラッシュ (/) で表されます。ファイルシステムは、*mount* コマンドによって、ディレクトリ階層に結付けられます。図 5-1 に、この IRIX ディレクトリ構造を示します。

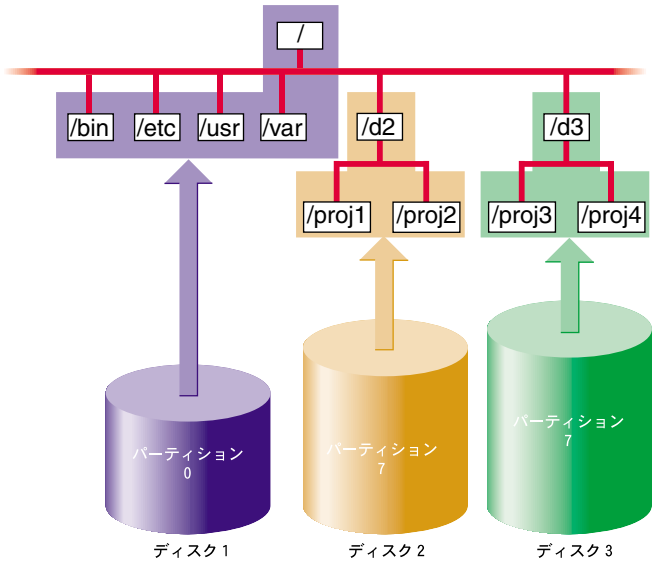


図 5-1 IRIX ファイルシステム

複数のディスク・パーティションを結合すると、論理ボリュームを作成できます。論理ボリュームは、単一のディスク・パーティションとして扱われるので、ファイルシステムは論理ボリュームに常駐できます。したがって、この方法を使用した場合にかぎり、単一のファイルシステムは複数のディスクにまたがることができます。XLV 論理ボリュームの詳細については、第 3 章「XLV 論理ボリュームの概念」を参照してください。

次にファイルシステムの重要なコンポーネントについて説明します。

i ノード

各ファイルに関する情報は、*i* ノードと呼ばれる構造体に格納されます。i ノードは、*index node*（インデックス・ノード）の略語です。i ノードはデータ構造体で、ディレクトリに格納されているファイル名を除くファイルのすべての情報を格納します。各 i ノードには、識別用の i ノード番号があります。この番号は、ファイルシステムで一意です。

次に、i ノードに含まれる情報を示します。

- ファイルのタイプ。詳細については、104 ページの「ファイルの種類」を参照してください。
- ファイルのアクセス・モード。このモードによって、読取り、書込み、および実行のアクセス権が定義されます。また、この情報にはセキュリティ・ラベルとアクセス・コントロール・リストも含まれています。
- ファイルとのハード・リンク数。詳細については、104 ページの「ハード・リンクとシンボリック・リンク」を参照してください。
- ファイルの所有者（所有者のユーザ ID 番号）とファイルが属するグループ（グループ ID 番号）。
- ファイルのバイト数。
- ファイルが最後にアクセスされ、修正されたときの日付と時刻。
- ディスク・パーティションまたは論理ボリュームにおけるファイル・データの検索についての情報。
- シンボリック・リンクのパス名。XFS ファイルシステムでシンボリック・リンクが適合する場合にかぎります。

`ls` コマンドにさまざまなオプションを指定すると、i ノードに格納されている情報を表示できます。たとえば、`ls -l` コマンドを使用すると、上記の最後の 2 つの項目を除くすべての項目を表示できます。表示される日付は、最後に修正を行った時刻です。

i ノードには、ファイルやそのディレクトリの名前はありません。

ファイルの種類

ファイルシステムには、表 5-2 に示すファイルの種類を定義できます。ファイルの種類は、*ls -l* 出力行の先頭の文字で確認できます。

表 5-2 ファイルの種類

ファイルの種類	文字	説明
標準のファイル	-	標準の一次元のバイト配列。
ディレクトリ	d	ファイルおよびそのほかのディレクトリのコンテナ。
シンボリック・リンク	l	ほかのファイル名またはディレクトリ名を示すファイル。
キャラクタ・デバイス	c	ハードウェアと IRIX の通信を可能にするデバイス。データはキャラクタ・ベースでアクセスされます。
ブロック・デバイス	b	ハードウェアと IRIX の通信を可能にするデバイス。データはシステムのバッファ・キャッシュからブロック内にアクセスされます。
名前付きパイプ (FIFO)	p	同じホストで実行されている関連付けられていない 2 つのプロセス間の通信を可能にします。これらは、*mknod* コマンドにより作成されます。*mknod* の詳細については、mknod(1M) マン・ページを参照。
UNIX ドメイン・ソケット	s	ネットワークを介して通信を可能にするプロセス間の接続。

ハード・リンクとシンボリック・リンク

第 5 章の「i ノード」で説明したとおり、ファイル名とディレクトリを除くファイルについての情報は、各ファイルの i ノードに格納されています。ファイル名はそのファイルのディレクトリに格納され、ファイル名と i ノード番号を関連付けることによってファイルへのリンクが作成されます。このリンクをハード・リンクといいます。すべてのファイルがハード・リンクを作成しますが、通常、この用語は複数のファイル名が同じ i ノード番号と関連している場合にのみ使用されます。i ノード番号は、ファイルシステムでのみ一意的です。このため、ファイルシステムの境界を越えてハードリンクは作成できません。

ファイルに対する 2 番目以降のハード・リンクは、*ln* コマンドに **-s** オプションを付けずに作成します。たとえば、カレント・ディレクトリに *origfile* というファイルがある場合、*origfile* ファイルに対して *linkfile* というハード・リンクを作成するには、次のコマンドを実行します。

```
% ln origfile linkfile
```

次に、*origfile* と *linkfile* の *ls -l* 出力を示します。サイズと修正時刻が同じです。

```
% ls -l origfile linkfile
-rw-rw-r--    2 joyce    user         4   4 月  5 日 11 時 15 分 origfile
-rw-rw-r--    2 joyce    user         4   4 月  5 日 11 時 15 分 linkfile
```

origfile と *linkfile* は、同一のファイルに付いた名前です。このため、ファイル内容の変更はどちらのファイル名からでも確認できます。片方のリンクを削除してももう一方へは影響しません。リンク・ファイルは、リンク・ファイルに対するリンクがなくなるまで削除されません。リンク数（リンク・カウント）は、ファイルの i ノードに格納されます。

別の種類のリンクとしてシンボリック・リンクがあります。この種類のリンクは、実際にはファイルです（表 5-2 を参照）。ファイルには、別のファイルまたは別のディレクトリのパス名を示すテキスト文字列が入っています。シンボリック・リンクはファイルであるため、ファイルの所有者とパーミッションが設定されています。シンボリック・リンクが示すファイルまたはディレクトリは、別のファイルシステムにある場合があります。こうしたファイルやディレクトリが削除されると、リンクするターゲットが再び作成されないかぎり、シンボリック・リンクは無効になります。これをダングリング・シンボリック・リンクといいます。

シンボリック・リンクは、*ln* コマンドに **-s** オプションを指定して作成します。たとえば、*origdir* ディレクトリに対して *linkdir* と呼ばれるシンボリック・リンクを作成するには、次のコマンドを実行します。

```
% ln -s origdir linkdir
```

次に、シンボリック・リンクの *ls -ld* 出力を示します。パーミッションの内容とそのほかの情報は一致していません。*linkdir* のリストでは、*origdir* に対するシンボリック・リンクであることを示しています。

```
% ls -ld linkdir origdir
drwxrwxrwt  13 sys      sys   2048   4 月  5 日 11 時 37 分 origdir
lrwxrwxr-x   1 joyce    user     8   4 月  5 日 11 時 52 分 linkdir -> origdir
```

シンボリック・リンクを伴うパス名に “..” を使用すると、シンボリック・リンクを含む親ディレクトリではなく、真のファイルまたはディレクトリの親ディレクトリを参照することに注意してください。

ハード・リンクとシンボリック・リンクの詳細については、ln(1) マン・ページを参照し、実際にハード・リンクとシンボリック・リンクを作成したり削除してください。

ファイルシステム名

ファイルシステムには名前がありません。したがって、ファイルシステムは次のようにディスク上の位置やディレクトリ構造内の位置から識別されます。

- ファイルシステムを含むディスク・パーティションまたは論理ボリュームのブロック・デバイス・ファイル名およびキャラクタ・デバイス・ファイル名から識別します。第 1 章の「ブロック・デバイスとキャラクタ・デバイス」を参照してください。
- ディスク・パーティションのニーモニック名、またはファイルシステムを含む論理ボリュームから識別します。第 2 章の「ln によるデバイス・ファイルのニーモニック名の作成」を参照してください。
- ファイルシステムのマウント・ポイントから識別します。114 ページの「ファイルシステムのマウントとアンマウント」を参照してください。

ファイルシステムを管理するため、コマンド（*mkfs*、*mount*、*umount*、*fsck* など）を使って上記のどの方法でファイルシステムを識別するかは、コマンドによって異なります。使用するコマンドのマン・ページを参照するか、このマニュアルの例を参照してどのファイルシステム名を使用するかを確認します。

XFS ファイルシステム

XFS は IRIX ファイルシステムで、デスクトップ・システムからスーパーコンピュータ・システムまでの、大半の SGI システム上で利用することを目的に設計されています。主な特徴を次に示します。

- 完全な 64 ビット・ファイル機能を備えています（2 GB 以上のファイル）。
- ジャーナリング技術を使用することで、システムがクラッシュしても迅速にかつ高い信頼性でシステムを回復できます。
- 大規模なスパース・ファイル（ホールのあるファイル）を効率的にサポートします。
- 統合され、完全な機能を備えたボリューム・マネージャ XLV Volume Manager を備えています。
- マルチプロセッシング・システムにおいてスケーラブルな、非常に高い I/O 性能を備えています。
- マルチメディアとデータ取得の使用に対して帯域保証 I/O を実現します。
- 既存のアプリケーションおよび NFS と互換性があります。
- 512 バイトから 64 KB までのファイルシステム・ブロック・サイズをユーザ指定できます。

- 小さなディレクトリと 156 文字以下のシンボリック・リンクは領域を使用しません。

XFS ファイルシステムを使用する場合、少なくとも 32 MB のメモリを装備することをお薦めします。

XFS ファイルシステムは、32 ビット・システム（IP17、IP20、IP22、および IP32）上で 2^{40}-1 つまり 1,099,511,627,775 バイト（1 テラバイト）のファイルとファイルシステムをサポートします。64 ビット・システム（IP19、IP21、IP25、IP26、および IP27）の場合は、2^{63}-1 バイトのファイルと、サイズ制限のないファイルシステムがサポートされます。IRIS Development Option(IDO) ソフトウェア・オプションで提供されるファイルシステム・インタフェースを使用すると、64 ビットの位置とファイル・サイズを追跡できる 32 ビット・プログラムを作成できます。2 GB より大きいファイルでも順次読取りが行われるので、多くのプログラムは修正を必要とすることなく動作します。また、NFS によって、64 ビット XFS ファイルシステムをほかのシステムにエキスポートできます。

XFS ファイルシステムは、データベース・ジャーナリング技術を使用することで、信頼性が高く迅速な回復を可能にします。システム・クラッシュが起きても、*fsck* コマンドなどのファイルシステム・チェッカを使用せずに、数秒で回復を完了します。回復に必要な時間は、ファイルシステムのサイズに左右されません。

XFS ファイルシステムは、パフォーマンスが非常に高いファイルシステムとして設計されています。特定の条件下でのスループットは毎秒 100 MB を超えます。この性能は、CHALLENGE MP アーキテクチャおよび ORIGIN 2000 アーキテクチャを補完します。従来のファイルシステムでは、サイズを大きくすることによってパフォーマンスが低下しましたが、XFS ではパフォーマンスが低下することはありません。

512 バイト～ 64 KB のブロック・サイズで、ファイルシステムを作成できます。リアルタイム・データの場合、エクステントの最大サイズは 1 GB です。ファイル中の隣接したファイルのためのファイルシステム・エクステントは、ノーマル・ファイルに対して自動的に作成され、**fcntl()** システム・コールを使用したリアルタイム・ファイルに対するファイル作成時間で構成されます。またエクステントは、ファイルシステムのブロックの倍数です。i ノードは、必要に応じて XFS ファイルシステムで作成されます。i ノードのサイズを指定するには、ファイルシステムを作成する *mkfs* コマンドに **-i** オプションを使用します。また、i ノードが専有できるファイルシステム内の領域の最大比率を指定するには、*mkfs* コマンドの **-i maxpct=** オプションを使用します。

XFS ファイルシステムの拡張属性によって、ユーザとアプリケーションは、名前と値のペアをファイル、ディレクトリ、シンボリック・リンク、および i ノードに関連付けることができます。この名前と値のペアを属性と呼び、*attr* コマンドで設定したり表示できます。詳細については、attr(1) マン・ページを参照してください。

XFS ファイルシステムの 2 つの機能を使って、アプリケーションの I/O 帯域幅の割当てを制御できます。第 8 章「帯域保証 I/O のシステム管理」で説明されている帯域保証 I/O を使用すると、プロセスは、システム上にほかのアクティビティがあるかどうかにかかわらず、予め定義された速度でシステム・リソースからデータを受信できます。prio(5) マン・ページで説明されている優先 I/O (Priority I/O) を使用すると、プロセスは、システム・リソースの一部を確保して一定の時間内だけ、排他的に使用できます。

ファイルシステムのほとんどのコマンド（*du*、*dvhtool*、*ls*、*mount*、*prtvtoc*、および *umount*）は、XFS ファイルシステムでも動作します。ユーザにとって、視覚上の違いはありません。*df*、*fx*、および *mkfs* など一部のコマンドは、XFS では追加機能があります。*clri*、*fsck*、*findblk*、および *ncheck* などのファイルシステム・コマンドは、XFS ファイルシステムでは使用されません。

バックアップおよびリストアには、標準的な IRIX コマンドである *backup*、*bru*、*cpio*、*restore*、*tar*、および IRIX 対応のオプション・ソフトウェア NetWorker を、2 GB 未満のファイルに対して使用できます。XFS ファイルシステムをダンプするには、*dump* コマンドの代わりに *xfsdump* コマンドを使用する必要があります。これらのダンプしたファイルをリストアするには、*xfsrestore* コマンドを使用します。XFS ファイルシステムにおける *xfsdump*、*xfsrestore*、*dump*、および *restore* の関係の詳細については、『IRIX Admin: Backup, Security and Accounting』の 第 2 章「バックアップと回復の手順」の「xfsdump と xfsrestore について」を参照してください。

CXFS ファイルシステム

CXFS は、クラスタ構造の XFS ファイルシステムです。論理ファイルの共有やファイルシステムのネットワーク化が可能で、パフォーマンスと機能性にすぐれています。CXFS は SAN（Storage area network）上で動作します。SAN では、クラスタ内の各コンピュータ・システムが、共有のディスク・セットにダイレクトな高速データ・チャネルを持っています。CXFS を実行するには、FLEXlm ライセンス・キーが必要です。

CXFS ファイルシステムの機能、インストール、管理については、『CXFS Software Installation and Administration Guide』を参照してください。

EFS ファイルシステム

メモ：今後の IRIX リリースでは、EFS ファイルシステムのサポートは予定されていません。EFS ファイルシステムを XFS ファイルシステムに変換する方法については、第 6 章「ファイルシステムの作成と拡張」を参照してください。

EFS ファイルシステムは、IRIX のオリジナル・ファイルシステムです。このファイルシステムには、標準の UNIX ファイルシステムを拡張したエクステント (extents) があり（この後の説明を参照）、EFS ファイルシステム (EFS: Extent File System) と呼ばれます。EFS ファイルシステムの最大サイズは、約 8 GB です。EFS は、512 バイトのファイルシステム・ブロックを使用し、ファイルの最大許容量は、2 GB から 1 バイト引いたものです。

EFS ファイルシステムとその管理方法については、付録 A「EFS ファイルシステム」を参照してください。

ネットワーク・ファイル・システム (NFS: Network File System)

オプションのネットワーク・ファイル・システム (NFS: Network File System) ソフトウェアを使用すると、NFS を利用できます。NFS は、1 つのホストからエキスポートされ、ネットワークを介してほかのホストにマウントされます。

NFS が常駐するホストでは、NFS はほかの XFS と同じように扱われます。NFS が XFS と違う唯一の点は、マウント用に、ほかのワークステーションからエキスポートされることです。NFS ファイルのエキスポートは、*exportfs* コマンドで実行します。ほかのホストでは、これらのファイルシステムは、*mount* コマンドを使用してマウントするか、または NFS のオートマウント機能を使用してマウントします。

アドバイス：NFS のマウントおよびエキスポートについては、『Personal System Administration Guide』の 第 4 章「自分のディスク領域をほかのユーザと共有」と 第 5 章「他のシステム上のディスク領域の使用」を参照してください。

NFS の詳細については、『ONC3/NFS Administrator's Guide』を参照してください。このマニュアルには、NFS ソフトウェア・オプションの情報もあります。

キャッシュ・ファイル・システム (CacheFS: Cache File System)

キャッシュ・ファイル・システム (CacheFS: Cache File System) は、新しいタイプのファイルシステムで、NFS およびそのほかのファイルシステムでクライアント側にキャッシング機能を提供します。ローカルなディスク領域を持つ NFS クライアントにおいて CacheFS を使用すると、サーバがサポートできるクライアント数が増え、読込み専用ファイルシステムによるクライアントのデータ・アクセス時間が減少します。

cfsadmin コマンドは、CacheFS を管理するために使用します。fsck コマンドの特殊バージョンである *fsck_cachefs* は、キャッシュ・ディレクトリの完全性をチェックするために使用します。このコマンドは、CacheFS がマウントされると自動的に呼出されます。CacheFS のマウントおよびアンマウントを行うときは、**-t cachefs** オプションを使用します。これらのコマンドの詳細については、cfsadmin(1M)、fsck_cachefs(1M)、および mount(1M) マン・ページを参照してください。

オプションの NFS ソフトウェアを使用すると、CacheFS を利用できます。CacheFS の詳細については、『ONC3/NFS Administrator's Guide』を参照してください。このマニュアルには、NFS ソフトウェア・オプションの情報もあります。

/proc ファイルシステム

/proc ファイルシステムは、デバッグ・ファイルシステムともいいます。このファイルシステムは、*ps* と *top* などの監視プログラム、および *dbx* などのデバッガで使用できるように、実行中の IRIX プロセスにインタフェースを提供します。デバッグ・ファイルシステムは、通常、*/debug* へのリンクにより、*/proc* にマウントされます。混乱を避けるために、*df* コマンドで空き領域を表示するときは、*/proc* は表示されません。

デバッグ・ファイルシステムのファイルは、*/proc/nnnnn* と */proc/pinfo/nnnnn* の形式で、*nnnnn* はプロセス ID に対応する 10 進数です。これらのファイルは、デバッグ・プロセス用のハンドルであるので、ディスク領域をむだに使用しません。なお、*/proc* ファイルは削除できません。

デバッグ・ファイルシステムの詳細については、proc(4) マン・ページを参照してください。

/hw ファイルシステム

hwgraph とも呼ばれるハードウェア・グラフは IRIX の機能で、大規模でトポロジ的に複雑な I/O サブシステムの管理を容易にします。*/hw* ファイルシステムは、hwgraph を視覚的に表しています。/hw ファイルシステムは、*/proc* と同様のファイルシステムの種類です。*/proc* と同じように、*df* コマンドで空き領域をリストしても、*/hw* は表示されません。

メモ：ハードウェア・グラフの内容およびリンクは、ハードウェアやソフトウェアのバージョンによって異なる場合があります。そのため管理者は 、全デバイス・ファイルへのパスの root として /dev ディレクトリを使用する必要があります。これは、/dev 下のディレクトリが /hw にリンクされている場合も同じです。たとえば、ファイルシステムをマウントしている場合や root ファイルシステムを設定している場合、/hw 下でデバイス名を使用することはできません。

---

hwgraph は方向性を示すグラフで、ハードウェア・エンティティを表す頂点（ポイント）と、頂点を結ぶエッジ（ライン）のセットで構成されています。各エッジは、ソース頂点からターゲット頂点への一方向のリンケージです。これが、方向性を示すグラフの定義です。各エッジにはラベルがあり、これはエッジの名前を表す文字列です。典型的な hwgraph の一部を、図 5-2 に示します。

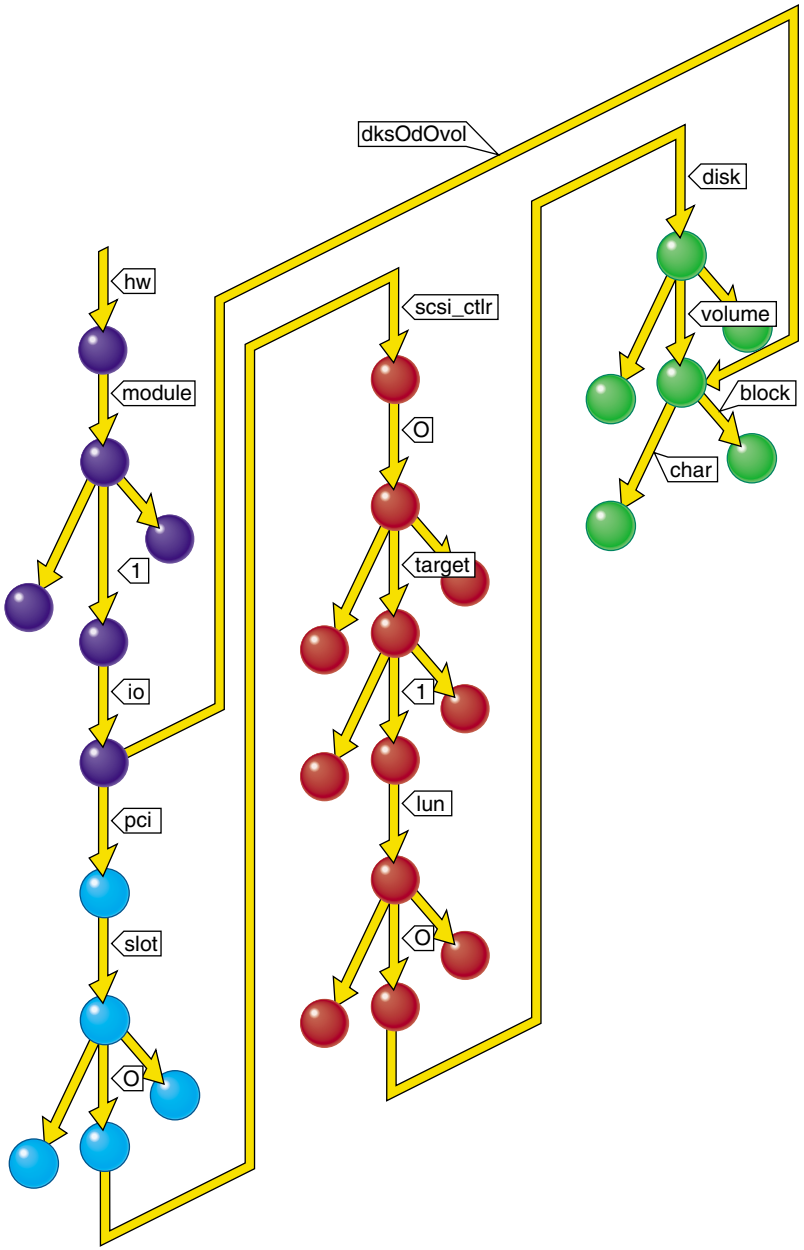


図 5-2 典型的な Hwgraph の一部

図 5-2 は、ディスクのボリューム・パーティション全体へのブロックおよびキャラクタ・アクセスを表すグラフの一部です。パス名表記を使用して、各ハードウェア・エンティティ（頂点）を識別しています。パス名は、root からハードウェア・エンティティへのパス内の、各エッジで構成されます。たとえば、ブロック・デバイスとキャラクタ・デバイスへの 2 つのパスは次のとおりです。

```
/hw/module/1/io/pci/slot/0/scsi_ctlr/0/target/1/lun/0/disk/volume/block
/hw/module/1/io/pci/slot/0/scsi_ctlr/0/target/1/lun/0/disk/volume/char
/hw/module/1/io/dks0d0vol/block
/hw/module/1/io/dks0d0vol/char
```

hwgraph は動的に構築され（ディスクの内容はありません）、ハードウェア・リストの変更を反映します。図 5-2 は色でコード分けされており、グラフがカーネル（黒）、PCI バス・アダプタ（赤）、SCSI コントローラ・ドライバ（マゼンタ）、ディスク・デバイス・ドライバ（緑）で構築されていることを示します。hwgraph では、I/O サブシステム内の各コントローラに対して、物理コントローラ番号ではなく論理コントローラ番号が使用されます。これらの論理コントローラ番号は、ファイル */etc/ioconfig.conf* で指定されます。詳細については、ioconfig(1M) マン・ページを参照してください。ioconfig(1M) マン・ページは、設定ファイル */etc/ioperms* についても説明しています。設定ファイルには、hwgraph 内のデバイスの所有者、グループ、およびパーミッションについての情報が含まれています。

cd、*ls*、および *find* などのコマンドを使用して */hw* ファイルシステムをナビゲートし、これをブラウズすることでハードウェア構成を検出できます。これまで */dev* に存在していたすべてのデバイス特殊ファイル名に対して、*/hw* パスへのシンボリック・リンクが存在します。ただし、XLV 論理ボリュームは除きます。シンボリック・リンクは、これを */dev* から */hw* に作成することによって実装します。次に典型的なリンクを示します。

```
lrwxr-xr-x    1 root     sys  13    8 月   6 日 11 時 23 分 /dev/root -> /hw/disk/root
```

/hw は削除しないでください。*/hw* を指定しないと、システムがほとんど動作しなくなります。

外部ファイルシステム

IRIX オペレーティング・システムは、ほかのオペレーティング・システムでは標準の 4 つのファイルシステム形式をサポートしています。

hfs (*mac*)　Macintosh コンピュータで使用されるファイルシステム

dos (*fat*)　IBM と互換性のあるパーソナル・コンピュータで使用されるファイルシステム

iso9660 (CD-ROM)
ISO 標準 9660 に準拠している CD-ROM ファイルシステム

cdda　コンパクト・ディスク・デジタル・オーディオ

IRIX がサポートするファイルシステムの種類についての詳細は、filesystems(4) マン・ページを参照してください。また、*hfs* ファイルシステムと *dos* ファイルシステムの管理方法については、mkfp(1M) および fpck(1M) マン・ページを参照してください。

XFS ファイルシステムの作成

ディスク・パーティションまたは論理ボリュームを XFS ファイルシステムに変換するには、*mkfs* コマンドを使用する必要があります。このコマンドは、ディスク・パーティションまたは論理ボリュームを取得し、それをデータ・ブロック、i ノード、および空きリストの各領域に分割し、該当する i ノード・テーブル、スーパーブロック、およびブロック・マップを書込みます。また、ファイルシステムのルート・ディレクトリを作成します。

次に、*mkfs* コマンドを使用して、1 MB の内部ログ・セクションを持つ XFS を作成する例を示します。

```
# mkfs -l size=1m /dev/rdsk/dks0d2s7
```

次に、*mkfs* コマンドを使用して、ログ・サブボリュームとデータ・サブボリュームを持つ論理ボリュームに XFS ファイルシステムを作成する例を示します。

```
# mkfs /dev/rxlv/a
```

XFS ファイルシステムの作成手順の詳細については、第 6 章「ファイルシステムの作成と拡張」、mkfs(1M)、および mkfs_xfs(1M) マン・ページを参照してください。

ファイルシステムのマウントとアンマウント

ファイルシステムを使用するには、マウントする必要があります。図 5-3 に、このプロセスを示します。ファイルシステムをマウントすると、ファイルシステムのデバイス・ファイル名（図 5-3 の */dev/rdsk/dks0d2s7*）とディレクトリ名（図 5-3 の */proj*）が指定されます。この */proj* ディレクトリは、マウント・ポイントと呼ばれ、マウント・ポイントがあるファイルシステムとマウントするファイルシステムを接続します。ファイルシステムをマウントすると、パス名の決定時にマウント・ポイントがそのファイルシステムのトップ・レベルのディレクトリに相当することをカーネルに通知します。図 5-3 では、*/dev/rdsk/dks0d2s7* のファイルシステムのファイル *a*、*b*、*c* が、図の下に示すように */proj/a*、*/proj/b*、*/proj/c* になります。

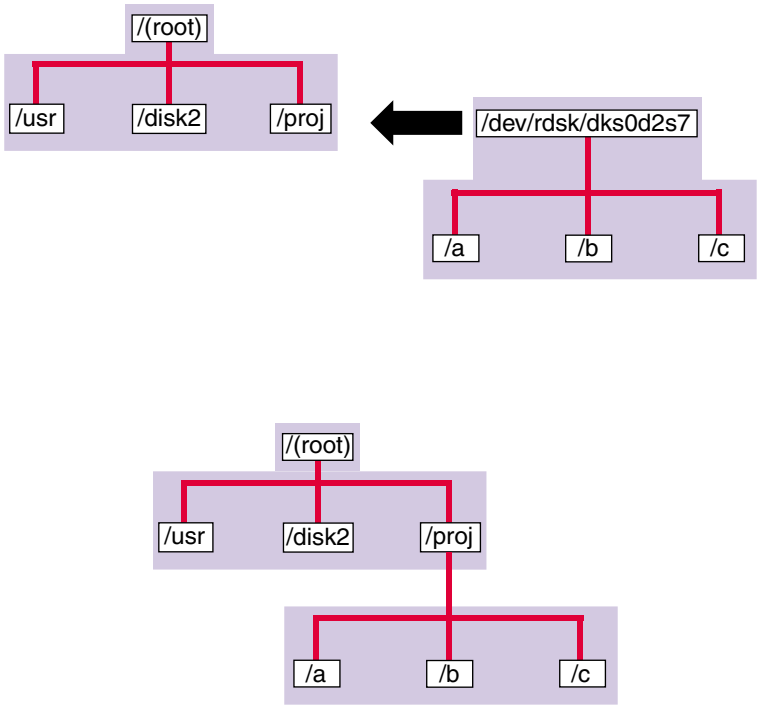


図 5-3 ファイルシステムのマウント

ファイルシステムをマウントすると、マウント・ポイント・ディレクトリの元の内容が隠され、ファイルシステムをアンマウントするまで使用できなくなります。ただし、マウント・ポイント・ディレクトリの所有者とパーミッションには影響はありません。制限されたパーミッションを設定しておけば、マウントされたファイルシステムへのアクセスを制限できます。

ほかのファイルシステムとは異なり、root ファイルシステム (/) は、カーネルが実行されるとすぐにマウントされます。ただし、システム・オペレーションには必要とされるので、アンマウントできません。usr ファイルシステムが root ファイルシステムとは別のファイルシステムである場合も、システムが正しく動作するためにマウントする必要があります。システム管理上、root および usr ファイルシステムをアンマウントする必要がある場合は、ミニルートで行います。詳細については、116 ページの「XFS ファイルシステムのチェック」を参照してください。

次に、ファイルシステムのマウント方法を示します。

- *mount* コマンドを使用して手作業で行います（第 7 章の「手作業によるファイルシステムのマウント」を参照）。
- */etc/fstab* ファイルの情報を使用して、システムの起動時に自動的に行います（第 7 章の「/etc/fstab ファイルによるファイルシステムのオートマウント」を参照）。
- ファイルシステムのアクセス時に自動的に行います（オートマウントと呼ばれ、NFS （リモート）ファイルシステムにかぎり適用されます)。第 7 章の「リモート・ファイルシステムのオートマウント」を参照）。
- リムーバブル・ディスクが挿入された際、自動的に行います（リムーバブル・メディア・デバイスを監視するデーモンについては、mediad(1M) マン・ページを参照）。

次に、ファイルシステムのアンマウント方法を 2 つ示します。

- システムを停止します（ファイルシステムは自動的にアンマウントされます）。
- *umount* コマンドを使用してファイルシステムを手作業でアンマウントします（第 7 章の「ファイルシステムのアンマウント」を参照）。

mount コマンドと *umount* コマンドの詳細については、第 7 章の「ファイルシステムのマウントおよびアンマウント」を参照してください。

XFS ファイルシステムのチェック

xfs_check コマンドは、XFS の整合性をチェックします。通常、ファイルシステムの整合性に問題があると考えられる場合のみ使用します。詳細については、xfs_check(1M) マン・ページを参照してください。

ファイルシステムの再編成

時間が経過するにつれ、ファイルシステムで断片化が起こります。断片化が発生すると、空き領域ブロックが少なくなり、ファイルが複数のエクステントにまたがります。*fsr* コマンドは、ファイルシステムを再編成して断片化したファイルのレイアウトを最適化し、その結果、全体のパフォーマンスを向上させます。

デフォルトでは、*fsr* は 1 週間に 1 度 *crontab* から自動的に実行されるように設定されています。マウントされたファイルシステムが XFS ファイルシステムであることを *fsr* コマンドが検出すると、*fsr_xfs* コマンドが呼び出されます。*fsr* コマンドについては fsr(1M) リファレン

ス・ページを参照してください。*fsr_xfs* オプションのコマンドについては、*fsr_xfs*(1M) マン・ページを参照してください。

ミニルートからのファイルシステムの管理

ファイルシステムの変更やそのほかの管理タスクにおいて root ファイルシステムがマウントされていない場合、または未使用の場合、IRIX システム・ソフトウェア・リリース CD のインストレーション・ツールで提供されるミニルート環境を使用できます。ミニルートを使用すると、*/* にマウントされたファイルシステムのスワップ・パーティションに、制限付きの IRIX バージョンがインストールされます。システムは、root および usr ファイルシステムの標準の IRIX ではなく、この制限付きの IRIX バージョンで動作します。root および usr ファイルシステムが利用可能になり、*/root* および */root/usr* にマウントされます。したがって、root および usr ファイルシステムのすべてのファイルのパス名には、接頭辞 */root* が付きます。

ファイルシステム領域の追加方法

ファイルシステム領域を追加するには、次の 3 つの方法があります。

- 新しいディスクを追加し、そのディスクにファイルシステムを作成し、既存のファイルシステムのサブディレクトリとして、そのファイルシステムをマウントします。
- 1 つのパーティションから領域を削除し、その領域を同じディスクの別のパーティションに追加して、既存のファイルシステムのサイズを変更します。
- 別のディスクを追加し、*xfs_growfs* コマンドを使用して既存の XFS ファイルシステムを追加したディスクに拡張します。

次に、上記のファイルシステム領域の追加方法を説明します。

サブディレクトリとしてのファイルシステムのマウント

ファイルシステムをサブディレクトリとしてマウントするには、独立したファイルシステムを持つ新しいディスクを追加し、ファイルシステム内に新しいマウント・ポイントを作成します。領域を追加するには、この方法が最も安全です。たとえば、usr ファイルシステムの領域が不足した場合、新しいディスクを追加して、*/usr/work* というディレクトリに新しいファイルシステムをマウントします。この手法の欠点は、既存のファイルシステムと新しいファイルシステム間でハード・リンクが作成できないことです。

ディスク・パーティションの分割方法、およびパーティションにあるファイルシステムの作成方法の詳細については、第 2 章「ディスク管理手順の実行」を参照してください。

別のファイルシステムからの領域の獲得

あるディスクのファイルシステムから、同じディスクの別のファイルシステムにディスク領域を移動するには、両方のファイルシステムの既存のデータをバックアップし、*fx* コマンドを実行してディスク・パーティションを切直し、*mkfs* コマンドを使用して 2 つのファイルシステムを再度作成する必要があります。この方法の欠点は、非常に手間がかかり、リスクを伴うことです。たとえば、あるファイルシステムのサイズを増やすには、代わりにほかのファイルシステムの領域を削除する必要があります。また、ファイルシステムのサイズの変更後、小さくなった新しいファイルシステムに、元のデータが適応することを確認する必要があります。なお、ファイルシステムのサイズ変更は、追加のディスク領域を獲得するまでの一時的な操作です。

ディスクの切直しについては、第 2 章の「fx によるディスク・パーティションの再分割」を参照してください。ファイルシステムが root ファイルシステムであるときのほかの解決方法については、第 7 章の「Root ファイルシステムでの空き領域の不足」を参照してください。

別のディスクへの XFS ファイルシステムの拡張

ファイルシステムの使用可能領域を増やすには、追加したディスクやパーティションに、既存のファイルシステムを拡張する方法もあります。既存のディスク・パーティションと新しいディスク・パーティションは、論理ボリュームになります。*xfs_growfs* コマンドは、ハードディスクの既存のデータを保護し、新しいディスク・パーティションからファイルシステムに領域を追加します。この処理は、ファイルシステムを再度作成するより簡単です。この方法の欠点は、1 つのディスクに不具合が生じた場合に、別のディスクのデータが正常に動作しても、そのデータを回復できない可能性があることです。usr ファイルシステムが論理ボリュームの場合は、システムをマルチユーザ・モードで起動できません。したがって、追加するディスクとファイルシステムは、ディレクトリとして root または usr（*/* または */usr*）にマウントすることをお薦めします。

追加ディスクへのファイルシステムの拡張の手順については、第 6 章の「別のディスクへの XFS ファイルシステムの拡張」を参照してください。

ディスクの割当て

システムのディスク領域が常に不足し、使用可能領域を増加できない場合は、ディスクの割当てを行う必要があります。ディスクの割当てによって、各ユーザが占有できる容量制限を設定できます。また、各ユーザが所有できるファイル数（i ノード）を制限することもできます。IRIX ではディスク割当てシステムを提供し、このプロセスを自動的に行います。このシステムを使用すると、システムの各ユーザにディスクの使用量を割当てることができます。また、ハード・リミットあるいはソフト・リミットのどちらかを選択することもできます。ハード・リミットはシステムが強制して行い、ソフト・リミットはディスク使用量を減らすようにユーザに通知します。ディスク使用量の制限は、*root* には適用されません。

ソフト・リミットを選択した場合、指定されたソフト・リミットを超える使用量でユーザがログインすると、ログイン・コマンドによって警告が発せられ、タイマが有効になります。割当て量がソフト・リミットよりも少ない値に落ちると、タイマが無効になります。タイマの有効期間がシステム管理者の設定した期間を超えた場合、超過したリミットは、ハード・リミットの場合と同じように扱われ、そのユーザにディスク領域が割当てられなくなります。この状態をリセットするには、使用量を割当て量より減らしてください。*root* においてのみ、この時間制限を設定できます。またこの処理はファイルシステム・ベースで行われます。

XFS ファイルシステムでは各種のオプションを使用できます。ユーザ、ファイルシステム、ユーザあたりの全ディスク使用量、ファイルの総数に対して制限を設定できます。また、XFS ファイルシステムでは、アカウントの数に制限がなく、ユーザ数が多い場合の罰則もほとんどありません。

また、XFS ファイルシステムではユーザ ID 以外にもプロジェクト ID に対する制限が設定できます。プロジェクト ID の内容と設定方法については『IRIX Admin:Backup, Security, and Accounting』を参照してください。プロジェクト ID に対するディスク割当ての設定については 162 ページの「XFS ファイルシステムでのディスク割当ての使用」を参照してください。

XFS ファイルシステムでディスク割当て機能を使用すると、ディスク使用量アカウンティングを行うことができます。ディスク使用量アカウンティングは、ディスク使用量を監視しますが、ディスク使用量制限は設定しません。詳細については、第 7 章の「ディスク領域を多く使用するアカウントの識別」を参照してください。

ディスク割当ての詳細は、quotas(4) マン・ページに詳しく説明されています。ディスク割当てを設定および監視する手順については、第 7 章の「XFS ファイルシステムでのディスク割当ての使用」を参照してください。

ファイルシステムの破損

ファイルシステムの破損は、システムに非常事態が発生した場合や、システムの停止が正しく行われなかった場合に起こります。これは、ソフトウェアやハードウェアの不具合、またはプラグを抜いてしまうなどの人為的ミスが原因です。また、パーティションの重複もファイルシステム破損の原因です。

ハードウェアの故障を予測する確実な方法はありません。ハードウェアの故障を回避するには、推奨されている方法に従って診断および保守を入念に行ってください。

人為的ミスは、ファイルシステム破損の最大の原因です。この問題を回避するには、次の決まりに従ってください。

- システムの停止は常に正しく行ってください。システムの電源を安易に切らないでください。*shutdown* コマンドなど、標準のシステムのシャットダウン・ツールを使用してください。
- 電源を切らずにハードディスクを取外すなど、ファイルシステムを物理的に移動しないでください。
- 読込み専用としてマウントされている場合を除き、マウントされたファイルシステムを物理的な書込み禁止にしないでください。
- 2 つのシステムのデュアルホスト・ディスクにファイルシステムを同時にマウントしないでください。

データの損失を回避するには、定期的に慎重なバックアップを行うことをお薦めします。システムのバックアップの詳細については、『IRIX Admin: Backup, Security, and Accounting』を参照してください。

場合によっては、XFS ファイルシステムの破損は、root ファイルシステムであってもコマンド *xfs_repair* で修復します。*xfs_repair* の詳細については、第 7 章の「xfs_check と xfs_repair による XFS ファイルシステムの整合性チェック」 を参照してください。

ファイルシステムの作成と拡張

この章では、XFS ファイルシステムを作成または拡張する（サイズを増やす）場合に実行する手順、または EFS ファイルシステムから XFS ファイルシステムに変換する場合に実行する手順について説明します。

この章では、次の項目について説明します。

XFS ファイルシステムの使用計画

次の項では、XFS ファイルシステムの作成時に準備すべき事項および選択事項について説明します。XFS ファイルシステムの作成、または EFS ファイルシステムから XFS ファイルシステムへの変換を計画する場合は、次で説明する内容を確認してから必要な準備を行ってください。

必要なソフトウェア

root および usr ファイルシステムを XFS に変換する場合は、ソフトウェアのディストリビューション CD を使用するか、IRIX システム・ソフトウェアのリモートディレクトリにアクセスする必要があります。

ファイルシステムのブロック・サイズとエクステント・サイズの選択

XFS ファイルシステムでは、各ファイルシステムに論理ブロック・サイズを選択できます。物理ディスク・ブロックは 512 バイトのままです。XLV 論理ボリュームでリアルタイム・サブボリュームを使用する場合は、エクステント・サイズも選択する必要があります。エクステント・サイズは、容量の割当ての追加が必要な場合、ファイルに割当てられる容量の合計です。

次に、ディスク・パーティションと論理ボリュームで使用する XFS ファイルシステム、および XLV ボリュームのファイルシステムにおけるデータ・サブボリュームのブロック・サイズに関するガイドラインを示します。

- ブロック・サイズの最小値は 512 バイトです。ブロック・サイズが小さいと、ファイルシステムのパフォーマンスを低下させるアロケーション・オーバーヘッドが増えます。ただし、100 MB 以下のファイルシステム、および多数の小さなファイルがあるファイルシステムでは、通常 512 バイトのブロック・サイズを推奨しています。ファイルシステムのブロック・サイズは 2 の乗数である必要があります。
- デフォルトのブロック・サイズは、4,096 バイト (4K) です。100 MB を超えるファイルシステムには、このサイズを推奨しています。
- ブロック・サイズの最大値は 65,536 バイト (64K) です。ブロック・サイズが大きいと、領域が無駄になり、断片化の原因となるので、通常はブロック・サイズを 4,096 バイト (4K) 以上にしません。
- 独立した root ファイルシステムと usr ファイルシステムがあるシステムの root ファイルシステムの場合は、512 バイトのブロック・サイズを推奨しています。root および usr が別になっていないファイルシステムには、デフォルトのブロック・サイズ 4,096 バイトを推奨しています。
- 新しいサーバの場合、バージョン 2 フォーマットの新しいファイルシステムをブロック・サイズ 512 バイト、ディレクトリ・ブロック・サイズ 4,096 バイトで使用することを推奨しています。バージョン 2 ディレクトリの使用に関しては、123 ページの「ファイルシステムのディレクトリ・フォーマットとディレクトリ・ブロック・サイズの指定」を参照してください。

ブロック・サイズは、10 進表現（デフォルト）、8 進表現（先頭が 0）、または 16 進表現（先頭が 0x か 0X）でバイトが指定されます。数値に接尾辞 “k” が付いている場合は 1,024 の倍数を示し、数値に接尾辞 “m” が付いている場合は 1,048,576 の倍数 (1,024 * 1,024) を示します。

次に、エクステント・サイズのガイドラインを示します。

- エクステント・サイズは、常にデータ・サブボリュームのブロック・サイズの倍数になります。
- 最小エクステント・サイズは、4 KB です。
- 最大エクステント・サイズは、1 GB です。

- デフォルトのエクステント・サイズは、64 KB です。
- エクステント・サイズは、リアルタイム・サブボリュームで使用されるボリューム・エレメントのアプリケーションおよびストライプ・ユニットに一致する必要があります。

ファイルシステムのエクステントは、ファイルへの割当て後に書き込みが行われていない場合、未書き込みとみなされます。これは、*fcntl*(2) システム・コールの *F_RESVSP* パラメータを使って領域をプリアロケートした場合に発生します。エクステントが未書き込みのときに領域をプリアロケートし、それからデータを読み込もうとすると、そのデータの内容は古い可能性があります。他のユーザが書き込むことも可能で、セキュリティが侵害される恐れがあります。

XFS ファイルシステムを定義するときに、未書き込みエクステントのトラッキング（Unwritten extent tracking）をオンにするかどうか指定できます。オンにすると、XFS は未書き込みエクステントを追跡し、古いデータが読み込まれないようにします。オンの場合、未書き込みエクステントが読み込まれると値 0 が返されます。IRIX 6.5 以上のバージョンでは、未書き込みエクステントのトラッキングはデフォルトでオンになっています。

ファイルシステムのディレクトリ・フォーマットとディレクトリ・ブロック・サイズの指定

XFS は、ディスク上のディレクトリについて、バージョン 1、バージョン 2 と呼ばれる 2 種類の mkfs 出力フォーマットをサポートしています。ファイルシステムの作成時に選択したバージョンは、ファイルシステムのすべてのディレクトリに適用されます。バージョン 1 は IRIX ファイルシステムのオリジナルのディレクトリ･フォーマットです。バージョン 2 は IRIX リリース 6.5.5 で追加されたフォーマットで、デフォルトで選択されます。ディレクトリ・フォーマットの指定は、*mkfs* コマンドの *-n* パラメータを使って行います。

バージョン 2 ディレクトリ・フォーマットの XFS ファイルシステムでは、ファイルシステム・ディレクトリの論理サイズとして、ファイルシステムの論理ブロック・サイズよりも大きい値を選択することができます。これにより、ディレクトリ操作のパフォーマンスを落とさずに、ファイルシステムのブロック・サイズを、配布するデータ・ファイルのサイズに合わせることができます。このオプションは、ニュースやメールのファイルシステムのように小さなファイルが多数あるファイルシステムのパフォーマンスを向上させるのにも役立ちます。この場合、ファイルシステムの論理ブロック・サイズを小さくし（512、1K、または 2K バイト）、ファイルシステム・ディレクトリの論理ブロック・サイズを大きく（4K または 8K バイト）にすることができます。これにより、インデックス情報を格納しているツリーに大きいブロックが割り当てられ、階層が浅くなるため、ディレクトリ検索のパフォーマンスが向上します。

ファイルの作成、削除を行うため、*readdir*(3C) または *getdents*(2) システム・コールでディレクトリにひんぱんにアクセスするようなアプリケーションをサポートしている場合、ファイルシステムのディレクトリの論理ブロック・サイズはファイルシステムの論理ブロック・サイズよ

り大きく設定する必要があります。ファイルシステムのブロック・サイズを小さくすると、ディスク領域が節約でき、小さなファイルの I/O スループットが向上します。

バージョン 2 ディレクトリ・フォーマットの XFS ファイルシステムでは、*readdir* を実行するのに必要なデータはインデックス情報とは別になります。バージョン 2 のディレクトリ・ブロックで *readdir* を実行すると、ディレクトリのデータ・ブロックを「あらかじめ読み込む」ことができます。これはバージョン 1 のディレクトリ・ブロックではできません。あらかじめ読み込むことで、*readdir* のパフォーマンスは劇的に向上します。

バージョン 2 のディレクトリ・ブロックでは *readdir* の実行に必要なデータとインデックス情報は別であるため、ディレクトリのオフセットは 32 ビットに限定されます。バージョン 1 のディレクトリ・ブロックでは、64 ビットのオフセットが使用されます。64 ビットのオフセットは、NFS V2、DFS といった 32 ビットのクライアントや 32 ビット（O32）アプリケーションでは、相互運用上の問題が発生する可能性があります。

SGI では、新規作成する XFS ファイルシステムでは必ずバージョン 2 ディレクトリを使用することを推奨しています。ただし、IRIX 6.5.5 よりも古いシステムではバージョン 2 ディレクトリのファイルシステムがマウントできないため、マウントを試みると次のエラー・メッセージが返されます。

```
Wrong filesystem type: xfs
```

ファイルシステムを単体で (バックアップを作成せずに) バージョン 1 とバージョン 2 の間で変換する方法はありません。*xfsdump*/*mkfs*/*xfsrestore* コマンド を使用すると、ファイルシステムの変換が可能です。

mkfs の *-n* オプションを使ったバージョン 1 ディレクトリ・フォーマットの選択方法については、*mkfs_xfs*(1M) マン・ページを参照してください。

ログ・タイプとログ・サイズの選択

各 XFS ファイルシステムには、ファイルシステム・ジャーナル・レコードを持つログがあります。このログには専用のディスク領域が必要です。このディスク領域は、*df* コマンドのリストには表示されません。また、ファイル名でアクセスすることもできません。

このディスク領域の位置は、選択するログ・タイプによります。2 つのログ・タイプを以下に示します。

外部ログ　XFS が XLV 論理ボリュームに作成され、ログ・レコードがログ・サブボリュームに書込まれるときのログを外部ログと呼びます。ログ・サブボリュームは、ログ専用のディスク・パーティションのことです。

内部ログ　　ディスク・パーティションまたは XLV 論理ボリュームに XFS が作成された場合、または、ログ・サブボリュームを持たない XLV 論理ボリュームに XFS が作成された場合は、ログ・レコードはユーザ・ファイルがあるディスク・パーティション（またはデータ・サブボリューム）の専用領域に書込まれます。このタイプのログを内部ログと呼びます。

次に、ログ・タイプを選択するときのガイドラインを示します。

- 異なるパーティションにログとデータ・サブボリュームを割当てたり、異なるサブボリューム構成を使用するには、外部ログを使用します。
- ログ・サブボリュームを、データ・サブボリュームから独立させる場合は（詳細については、第 3 章の「ボリューム・エレメント」を参照）、外部ログを使用する必要があります。
- ディスク・パーティションに、XVL 論理ボリュームではなく、XFS を作成する場合は、内部ログを使用する必要があります。
- ログ・サブボリュームを持たない XLV 論理ボリュームに XFS を作成する場合は、内部ログを使用する必要があります。
- ログ・サブボリュームを持つ XLV 論理ボリュームに XFS を作成する場合は、外部ログを使用する必要があります。

XVL およびログ・サブボリュームの詳細については、第 3 章の「XLV 論理ボリュームについて」を参照してください。

ログに必要なディスク領域は、ファイルシステムの使い方によって異なります。ログ・レコードに必要なディスク領域は、ファイルシステムのサイズに比例するのではなく、ファイルシステムでのトランザクション・レートとトランザクションのサイズに比例します。ブロック・サイズが大きくなると、トランザクションも大きくなります。ディレクトリを更新するトランザクションが行われると（*mkdir* と *rmdir* コマンド、および **create()** と **unlink()** システム・コールなど）、多くのログ・データが生成されます。

ログ専用のディスク領域の大きさ（ログ・サイズ）を指定することができます。ファイルシステムの最少ログ・サイズは、もっとも大きなトランザクションのサイズによって決定されます。トランザクション・サイズはファイルシステムとディレクトリのブロック・サイズによって決まります。最大ログ・サイズは 64k 個のブロックまたは 128MB のうち小さい方です (どちらになるかは、ブロック・サイズによります)。

高度なトランザクションを行うファイルシステムでは、ログ・サイズを大きく設定してください。ただし、過度に大きなログ・サイズを設定することは避けるべきです。ログ・サイズが大きすぎると、クラッシュ後のファイルシステムのマウントに時間がかかります。

デフォルトのログ・サイズは、ファイルシステムのサイズに応じて、最大ログ・サイズを上限として大きくなります。上限は 128 MB (1 テラバイトのファイルシステム) です。

ストライプ化されている XLV 論理ボリュームのファイルシステムの場合、デフォルトの内部ログ・サイズはストライプ・ユニット・サイズの倍数に丸められます。ユーザがサイズを指定する場合、ストライプ・ユニット・サイズの倍数である必要があります。

外部ログの場合、ログ・サイズはログ・サブボリュームのサイズと同じです。ログ・サブボリュームは、ディスク・パーティションから構成されます。ディスク・パーティションを再分割して、ログ・サブボリュームを適切なサイズにする必要がある場合があります。128 ページの「ディスク・パーティションの再分割」を参照してください。

外部ログでは、ログのサイズは *xlv_make* コマンドを使ってログ・サブボリュームを作成するときに設定されます。内部ログの場合は、*mkfs* コマンドを使用してファイルシステムを作成する際に、*-l size=* オプションでログ・サイズが指定されます。

122 ページの「ファイルシステムのブロック・サイズとエクステント・サイズの選択」で示すように、ログ・サイズはバイトで指定するか、または接尾辞 "b" を使用してファイルシステムのブロック・サイズの倍数で指定します。

アロケーション・グループとストライプ・ユニットの選択

XFS ファイルシステムのデータ・セクションは複数のアロケーション・グループに分割されます。アロケーション・グループの数は、XFS ファイルシステムを作成するときに指定できます。または、アロケーション・グループのサイズを指定することもできます。アロケーション・グループ数が多いほど、ブロックと i ノードのアロケート処理の並列性が向上します。ただし、アロケーション・グループをあまり多くすると (またはアロケーション・グループのサイズを、アロケーション・グループの数があまりにも多くなるように設定すると)、ファイルシステムの利用率の限界に近づいたときに CPU 時間が過度に増加する恐れがあります。

アロケーション・グループの最少サイズは 16 MB で、最大サイズは 4 GB 弱です。

ファイルシステムが 128 MB より小さいか、8 GB より大きくない限り、デフォルトのアロケーション・グループ数は 8 になります。ファイルシステムが 128 MB より小さい場合は最少アロケーション・グループ・サイズが 16 MB になるため、デフォルトのアロケーション・グループ数は 8 より少なくなります。この場合、デフォルトのデータ・セクションは、サイズが少なくとも 16 MB である、複数のアロケーション・グループにできるだけ多く分割されます。ファイルシステムが 8 GB より大きく 64GB より小さい場合、デフォルトのアロケーション・グループ数は 8 より多くなり、各アロケーション・グループのサイズは約 1 GB になります。ファイルシステム

が 64GB を越える場合、デフォルトのアロケーション・グループ数は 8 より多く、アロケーション・グループのサイズは 4GB になります。

XFS では、RAID デバイスまたは XLV ストライプ・ボリュームのストライプ・ユニットを選択できます。これにより、ファイルの終端が延長された場合やファイル・サイズが 512 KB より大きい場合に、データ・アロケーション、i ノード・アロケーション、および内部ログを、ストライプ・ユニットに合わせることができます。ストライプ・ユニット数は 512 バイト・ブロック単位またはバイト単位で指定します。値はファイルシステム・ブロック・サイズの倍数である必要があります。ストライプ・ユニット数の指定方法については *mkfs_xfs*(1M) マン・ページを参照してください。

ストライプ・ユニットを指定する際、ストライプの幅も指定します。ストライプの幅は 512 バイト・ブロック単位またはバイト単位で指定します。ストライプの幅はストライプ・ユニットの倍数である必要があります。ストライプの幅は *stat()* システム・コールで返される優先の I/O サイズになります。ストライプ幅の指定方法については mkfs_xfs(1M) マン・ページを参照してください。

mkfs コマンドを *-b* オプション付きで使用するときは、*-d su=* および *-d sw=* オプションを利用して、ストライプ・ユニット数とストライプ幅 （ファイルシステム・ブロック単位） が指定できます。

RAID デバイスの場合、デフォルトのストライプ • ユニットは 0 で、機能は無効になっています。RAID デバイスでストライプ・ユニットおよびストライプ幅のサイズを設定するときは慎重に行ってください。RAID ユニットの I/O 操作の最適な動作を損なわないような値を選ぶ必要があります。これを怠ると、予期しないパフォーマンスの低下が発生することがあります。たとえば、ブロック書き込みが RAID ストライプ・ユニット境界のアラインメントに一致しなかったり、あるいは完全なストライプ・ユニットに一致しなかったりすると、RAID が強制的に read-modify-write サイクルを実行してしまいます。この動作はパフォーマンスに深刻に影響します。ストライプ・ユニット・サイズを適切に設定することで、XFS がアラインメントに一致しないアクセスを起こすことが避けられます。

ストライプ化 XLV ボリュームでは、XLV ボリュームを作成するときに指定したストライプ・ユニットがデフォルトで使用されます。ストライプ・ユニット・サイズを指定する際の注意事項については、第 3 章「XLV 論理ボリュームの概念」の「ストライプ化されたボリューム・エレメント」を参照してください。

ディスク・パーティションの再分割

XFS ファイルシステムまたは XLV 論理ボリュームを切替えるとき、ディスク・パーティションの再分割が必要になることがあります。次に、パーティションを再分割する必要がある場合を示します。

- システム・ディスクに独立した root および usr ファイルシステム用のパーティションがある場合は、root ファイルシステムの容量が不足することがあります。ディスク・パーティションを再分割すると、usr のサイズを減らして root の領域を増やしたり、root と usr を結合させて単一のパーティションにして容量不足の問題を解決します。
- root と usr ファイルシステムが結合されている方がシステム管理がしやすくなります。
- XLV 論理ボリュームを使用する場合は、XFS ログを小さなサブボリュームに割当てることができます。この場合は、ディスク・パーティションを再分割して、ログ・サブボリューム用に小さなパーティションを作成します。
- XLV 論理ボリュームを使用する場合は、ディスク・パーティションを再分割して、ストライプ化またはプレックス化が可能な同一サイズのパーティションを作成することができます。

ディスク・パーティションについては、第 1 章「ディスクの概念」で説明します。また、*fx* を使用したディスク・パーティションの再分割については、第 2 章の「fx によるディスク・パーティションの再分割」を参照してください。

XFS ファイルシステムの作成

ここでは、空のディスク・パーティションまたは XLV 論理ボリュームに XFS ファイルシステムを作成する方法を説明します。XLV 論理ボリュームの作成方法の詳細については、第 4 章「XLV 論理ボリュームの作成と管理」を参照してください。

アドバイス : ディスク・パーティションまたは論理ボリュームに XFS ファイルシステムを作成する場合は、*xfsm* コマンドのグラフィカル・ユーザ・インタフェースを使用できます。詳細については、オンライン・ヘルプを参照してください。

注意 : ファイルシステムを作成するとき、ディスク・パーティションまたは論理ボリュームにある既存のファイルすべてが破壊されます。

---

1. 121 ページの「XFS ファイルシステムの使用計画」を読み、次の手順を行う準備が整っているかを確認します。
2. 作成するファイルシステムのパーティションまたは論理ボリュームのデバイス名を確認します。下記の例では、*partition* です。たとえば、コントローラ 0、ドライブ・アドレス 2 の SCSI オプション・ディスクにおいてパーティション 7（ディスク全体）を使用する場合、*partition* は */dev/dsk/dks0d2s7* になります。*partition* の決定方法の詳細については、17 ページの表 1-4、第 3 章の「XLV 論理ボリュームについて」、および dks(7M) マン・ページを参照してください。
3. ディスク・パーティションがすでにマウントされている場合は、アンマウントします。

   # **umount** *partition*

   ディスク・パーティションのデータはすべて破壊されます。データを破壊しないでデータの変換を行うには、第 6 章の「オプション・ディスク上のファイルシステム EFS から XFS への変換」の手順を実行してください。
4. ログ・サブボリュームを持たないディスク・パーティションまたは XLV 論理ボリュームにファイルシステムを作成するときに、デフォルト値のブロック・サイズとログ・サイズを使用する場合、次の *mkfs* コマンドを使用して新しい XFS ファイルシステムを作成します。

   # **mkfs** *partition*

   例 6-1 に、デフォルト値を使用して XFS ファイルシステムを作成するコマンド行と、システム出力を示します。

例 6-1 デフォルト値を使用して XFS ファイルシステムを作成するための *mkfs* コマンド

```
# mkfs /dev/dsk/dks0d4s7
meta-data=/dev/dsk/dks0d4s7      isize=256    agcount=9, agsize=262144 blks
data     =                       bsize=4096   blocks=2222178, imaxpct=25
         =                       sunit=0      swidth=0 blks, unwritten=1
naming   =version2 bsize=4096
log      =internal log           bsize=4096   blocks=1200
realtime =none                   extsz=65536  blocks=0, rtextents=0
```

5. ログ・サブボリュームを持たないディスク・パーティションまたは XLV 論理ボリュームにファイルシステムを作成するときに、自分でブロック・サイズとログ・サイズを指定する場合、次の *mkfs* コマンドを使用して新しい XFS ファイルシステムを作成します。

   # **mkfs -b size=***blocksize* **-l size=***logsize partition*

   blocksize は、ファイルシステムのブロック・サイズです。122 ページの「ファイルシステムのブロック・サイズとエクステント・サイズの選択」を参照してください。*logsize* は、ログ・レコード専用領域のサイズです。124 ページの「ログ・タイプとログ・サイズの選択」を参照してください。デフォルト値は、4 KB ブロックと 1,000 ブロック・ログです。

例 6-2 に、XFS ファイルシステムを作成するコマンド行と、システム出力を示します。ファイルシステムには 10 MB の内部ログと 1 KB のブロック・サイズがあり、パーティション */dev/dsk/dks0d4s7* にあります。

例 6-2 *mkfs* 内部ログを持つ XFS に使用するコマンド

```
# mkfs -b size=1k -l size=10m /dev/dsk/dks0d4s7
meta-data=/dev/dsk/dks0d4s7      isize=256    agcount=9, agsize=1048576 blks
data     =                       bsize=1024   blocks=8888712, imaxpct=25
         =                       sunit=0      swidth=0 blks, unwritten=1
naming   =version 2              bsize=4096
log      =internal log           bsize=1024   blocks=10240
realtime =none                   extsz=65536  blocks=0, rtextents=0
```

6. ログ・サブボリューム（外部ログ用）を持つ XLV 論理ボリュームにファイルシステムを作成する場合は、次の *mkfs* コマンドを使用して、新しい XFS ファイルシステムを作成します。

   # **mkfs -b size=***blocksize volume*

   blocksize は、ファイルシステムのブロック・サイズです。122 ページの「ファイルシステムのブロック・サイズとエクステント・サイズの選択」を参照してください。*volume* は、ボリュームのデバイス名です。

   例 6-3 に、ブロック・サイズが 1 K バイトの論理ボリューム */dev/xlv/a* に XFS ファイルシステムを作成するコマンド行とそのシステム出力を示します。

例 6-3 外部ログを持つ XFS に使用する *mkfs* コマンド

```
# mkfs -b size=1k /dev/xlv/a
meta-data=/dev/xlv/a             isize=256    agcount=9, agsize=1048576 blks
data     =                       bsize=1024   blocks=8888712, imaxpct=25
         =                       sunit=0      swidth=0 blks, unwritten=1
naming   =version 2              bsize=4096
log      =volume log             bsize=1024   blocks=32768
realtime =none                   extsz=65536  blocks=0, rtextents=0
```

例 6-4 に、ログ、データ、リアルタイム・サブボリュームを持つ論理ボリューム */dev/xlv/xlv_data1* に XFS ファイルシステムを作成するコマンド行とそのシステム出力を示します。4,096 バイトのデフォルトのブロック・サイズが使用され、リアルタイム・エクステント・サイズは、128 KB に設定されます。

例 6-4 リアルタイム・サブボリュームを持つ XFS ファイルシステムに使用するコマンド

```
# mkfs -r extsize=128k /dev/xlv/xlv_data1
meta-data=/dev/xlv/xlv_data1     isize=256    agcount=9, agsize=262144 blks
data     =                       bsize=4096   blocks=2222178, imaxpct=25
         =                       sunit=0      swidth=0 blks, unwritten=1
naming   =version 2              bsize=4096
```

```
log      =volume log             bsize=4096   blocks=8192
realtime =volume rt              extsz=131072 blocks=1077787, rtextents=33680
```

7. バージョン 2 ディレクトリ・フォーマットでファイルシステムを設定し、ディレクトリのブロック・サイズがファイルシステムのブロック・サイズよりも大きい場合、次の *mkfs* コマンドを使って新規の XFS ファイルシステムを作成します。

   # **mkfs -b size=***blocksize* **-n size=***dirblocksize partition*

   blocksize はファイルシステムのブロック・サイズです（122 ページの「ファイルシステムのブロック・サイズとエクステント・サイズの選択」を参照）、*dirblocksize* はディレクトリのブロック・サイズです（123 ページの「ファイルシステムのディレクトリ・フォーマットとディレクトリ・ブロック・サイズの指定」を参照）。

   例 6-5 は、XFS ファイルシステムを作成するコマンド行とそのシステム出力を示しています。このファイルシステムは、ファイルシステム・ブロックが 512 バイト、ディレクトリ・ブロックが 4 k バイトで、*/dev/dsk/dks0d4s7* というパーティションにあります。このようなファイルシステムは、メールやニュースのファイルを格納するのに適しています。

例 6-5 ディレクトリのブロック・サイズを指定する XFS ファイルシステムの mkfs コマンド

```
# mkfs -b size=512 -n size=4k /dev/dsk/dks0d4s7
meta-data=/dev/dsk/dks0d4s7      isize=256    agcount=9, agsize=2097152 blks
data     =                       bsize=512    blocks=17777424, imaxpct=25
         =                       sunit=0      swidth=0 blks, unwritten=1
naming   =version 2              bsize=4096
log      =internal log           bsize=512    blocks=4944
realtime =none                   extsz=65536  blocks=0, rtextents=0
```

8. IRIX 6.5.5 よりも古いシステムにマウントするファイルを作成するため、バージョン 1 フォーマットのディレクトリを使用したい場合は、次の *mkfs* コマンドを使って新しい XFS ファイルシステムを作成します。

   # **mkfs -b -n version=1** *partition*

   XFS ファイルシステムを作成するコマンドラインと、それに対するシステム出力を例 6-6 に示します。このファイルシステムは 512 バイトのファイルシステム・ブロックとバージョン 1 ディレクトリ構造を持ち、パーティション */dev/dsk/dks0d4s7* に作成されます。

例 6-6 バージョン 1 ディレクトリの XFS ファイルシステムを作成する *mkfs* コマンド

```
# mkfs -b size=512 -n version=1 /dev/dsk/dks0d4s7
meta-data=/dev/dsk/dks0d4s7      isize=256    agcount=9, agsize=2097152 blks
data     =                       bsize=512    blocks=17777424, imaxpct=25
         =                       sunit=0      swidth=0 blks, unwritten=1
naming   =version 1              bsize=512
log      =internal log           bsize=512    blocks=4944
realtime =none                   extsz=65536  blocks=0, rtextents=0
```

9. ファイルシステムを使用可能にするためには、マウントする必要があります。次のコマンドでマウントします。

   ```
   # mkdir mountdir
   # mount partition mountdir
   ```

   ファイルシステムのマウントの詳細については、第 7 章の「手作業によるファイルシステムのマウント」を参照してください。

10. システムの起動時に、新しいファイルシステムが自動的にマウントされるようにシステムを構成するには、次の行をファイル */etc/fstab* に追加します。

    partition mountdir **xfs rw,raw=***rawpartition* **0 0**

    rawpartition は、*partition* のロー・バージョンです。たとえば、*partition* が */dev/dsk/dks0d2s7* の場合は、*rawpartition* は */dev/rdsk/dks0d2s7* です。

    ファイルシステムを自動的にマウントする方法の詳細については、第 7 章の「/etc/fstab ファイルによるファイルシステムのオートマウント」を参照してください。

inst からのファイルシステムの作成

注意 : ファイルシステムを作成すると、ディスク・パーティションまたは論理ボリュームにあるファイルはすべて破壊されます。

ファイルシステムを作成するには、*mkfs* を *inst* コマンドから使用できます。システム・ディスクに root ファイルシステムまたは usr ファイルシステムを作成する場合は、ミニルートから *inst* コマンドを使用する必要があります。*mkfs* には、次の 2 つの使用方法があります。

- 「Administrative Command Menu」で *mkfs* コマンドを使用します。*mkfs* コマンドによって XFS ファイルシステムが作成され、*mkfs* コマンド・オプションのデフォルト値が使用されます。引数を指定しない場合は、*mkfs* コマンドによって root ファイルシステムと usr ファイルシステム（ある場合）が作成されます。ほかのファイルシステムは、*mkfs* にデバイス・ファイル引数を指定して作成できます。
- シェルから実行します。シェルから *mkfs* コマンド（*shroot* コマンドではなく、*sh* コマンド）を実行すると、オプション付きの *mkfs* コマンド行を指定できます。

inst からのファイルシステムの作成については、『IRIX Admin: Software Installation and Licensing』を参照してください。

外部ファイルシステムの作成

IRIX オペレーティング・システムでは、*mkfp* コマンドを使用して、hfs (mac) ファイルシステムと dos (fat) ファイルシステムをフロッピー、光ディスク、SyQuest、Jaz、PC カード、Zip、光磁気 (MO) ディスクおよびハード・ドライブなどのデバイス上に作成することができます。

mkfp ユーティリティを使用すると、フロッピーと光ディスクに単一の *dos* パーティションを作成したり、ほかの媒体に複数の *dos* パーティションを作成することができます。ただし、*mkfp* ユーティリティは、ディスク全体で単一の hfs パーティションしか作成できません。また、*mkfp* ユーティリティを使用して、ディスク上の既存のパーティションを操作することはできません。

mkfp ユーティリティの使用方法については、mkfp(1M) マン・ページを参照してください。外部ファイルシステムの種類についての詳細は、filesystems(4) マン・ページを参照してください。また、外部ファイルシステムのチェックと修復については、fpck(1M) マン・ページを参照してください。

メモ： *mkfp* を使用してシステムにファイルシステムを作成する際に問題が生じた際には、ファイルシステムのプラットフォーム用のファイルシステム作成ユーティリティの使用が必要になる可能性があります。

別のディスクへの XFS ファイルシステムの拡張

ここでは、XFS ファイルシステムを別のディスクに拡張する手順について説明します。XFS ファイルシステムを別のディスクに拡張する場合、次の 2 つの可能性があります。

- XFS がディスク・パーティションにある場合
- XFS が XLV 論理ボリュームにある場合

XFS が XLV 論理ボリュームにある場合は、追加のボリューム・エレメントとして、論理ボリュームにディスクを追加できます。この方法については、第 4 章の「プレックスへのボリューム・エレメントの追加（XLV 論理ボリュームの拡張）」を参照してください。

次の手順は、*/mnt* ディスク・パーティションおよび新しいディスクから作成された XLV 論理ボリュームに、*/mnt* にマウントされた XFS を拡張する方法を示します。この手順では、すでに新しいディスクがシステムにインストールされており、パーティションに分割されているものとします。

注意 : 追加するディスクにあるすべてのファイルは、この手順の最中に破壊されます。

1. 拡張するファイルシステムのバックアップを作成します。
2. */mnt* ファイルシステムをアンマウントします。

```
# umount /mnt
```

3. *xlv_make* を使用して */mnt* パーティションと新しいディスクから XLV 論理ボリュームを作成します。*/mnt* パーティションはデータ・サブボリュームの先頭のボリューム・エレメントである必要があります。

```
# xlv_make
xlv_make> vol xlv0
xlv0
xlv_make> data
xlv0.data
xlv_make> plex
xlv0.data.0
xlv_make> ve dks0d4s7
xlv0.data.0.0
xlv_make> ve dks0d3s0
xlv0.data.0.1
xlv_make> end
Object specification completed
xlv_make> exit
Newly created objects will be written to disk.
Is this what you want?(yes) yes
Invoking xlv_assemble
```

4. */mnt* ファイルシステムをマウントします。

```
# mount /dev/xlv/xlv0 /mnt
```

5. *xfs_growfs* コマンドを使用して、論理ボリュームに XFS ファイルシステムを拡張します。

```
# xfs_growfs /mnt
meta-data=/mnt                 isize=256    agcount=9, agsize=2097152 blks
data     =                       bsize=512    blocks=17777424, imaxpct=25
         =                       sunit=0      swidth=0 blks, unwritten=1
naming   =version 2              bsize=4096
log      =internal               bsize=512    blocks=4944
realtime =none                   extsz=65536  blocks=0, rtextents=0
data blocks changed from 17777424 to 26399727
```

6. ディスク・パーティションでなく、論理ボリュームをマウントするために、ファイル */etc/fstab* の */mnt* のエントリを変更します。

```
/dev/xlv/xlv0 /mnt xfs rw,raw=/dev/rxlv/xlv0 0 0
```

これで、ファイルシステムの拡張は完了です。

システム・ディスク上のファイルシステム EFS から XFS への変換

注意：ここでの手順は、正しく行わないとデータが消去されてしまうことがあります。したがって、この操作は IRIX システム管理者が行うことをお薦めします。

ここでは、システム・ディスクの EFS を XFS に変換するための手順について説明します。システムによっては、システム・ディスクに root ファイルシステム（/ にマウント）と usr ファイルシステム（*/usr* にマウント）の 2 つのファイルシステムがある場合があります。または、root ファイルシステムと usr ファイルシステムが結合して / にマウントされていることもあります。この手順では、この両方の場合について説明します。ただし、システム・ディスク上で XLV 論理ボリュームが使用されていないものとします。次に、システム・ディスクを変換するための基本的な手順を示します。

1. ミニルートをロードします。
2. システム・ディスクのファイルシステムをダンプします。
3. 必要に応じて、システム・ディスクをパーティションに再分割します。
4. 新しい空の XFS を作成します。
5. ファイルシステムのダンプからファイルをリストアします。
6. システムを再起動します。

ここでの手順では、必要に応じてシステム・ディスクをパーティションに再分割できます。たとえば、root ファイルシステムと usr ファイルシステムを単一のファイルシステムにしたり、パーティションのサイズを変更して、root パーティションを大きくし、usr パーティションを小さくできます。詳細については、128 ページの「ディスク・パーティションの再分割」を参照してください。

最初の手順では、さまざまな変数の値を指定します。これらの値は、後の手順で使用されます。後で参照できるように、変数と値のリストを作成しておくと便利です。この手順では、各自の状況に合った手順のみ行ってください。また、この手順はすべて特権ユーザとして実行します。

注意：追加の *inst* コマンドやシェル・コマンドを使用しないで、必ずこの手順に従ってください。この手順に従わなかったり、別のディレクトリに変更したり、指定されていないときに *inst* シェルから *inst* メニューにアクセスすると、回復が難しい状況に陥ります。

1. 第 6 章の「XFS ファイルシステムの使用計画」を読み、この手順を開始できる状態かどうかを確認します。
2. バックアップが最新かどうかを確認します。この手順では一時的にシステム・ディスクからすべてのファイルが削除されるので、通常のバックアップ手順を使用して完全なバックアップを作成しておくことが重要です。システム・ディスクの完全なダンプは 11 で作成しますが、この手順で作成するバックアップのほかに通常のバックアップも必要です。
3. *devnm* を使用して、root ディスク・パーティション *rootpartition* のデバイス名を調べます。

   ```
   # devnm /
   /dev/dsk/dks0d1s0 /
   ```

4. システム・ディスクに独立した root ファイルシステムと usr ファイルシステムがある場合は、*devnm* を使用して、usr パーティションのデバイス名、つまり *usrpartition* を調べます。

   ```
   # devnm /usr
   /dev/dsk/dks0d1s6 /usr
   ```

5. バックアップ・デバイスとしてテープ・ドライブを使用している場合は、*hinv* を使用して、そのテープ・ドライブのコントローラおよびユニット番号（*tapecntlr* および *tapeunit*）を調べます。

   ```
   # hinv -c tape
   Tape drive: unit 2 on SCSI controller 0: DAT
   ```

   この例では、*tapecntlr* は 0 で、*tapeunit* は 2 です。
6. バックアップ・デバイスとしてディスク・ドライブを使用している場合は、*df* を使用してバックアップを保存するファイルシステムが格納されているパーティションのデバイス名 (*backupdevice*) とマウント・ポイント (*backupfs*) を調べます。

   ```
   # df
   Filesystem             Type  blocks     use   avail %use  Mounted on
   /dev/root               efs 1992630  538378 1454252  27%  /
   /dev/dsk/dks0d3s7       efs 3826812 1559740 2267072  41%  /disk3
   /dev/dsk/dks0d2s7       efs 2004550      23 2004527   0%  /disk2
   ```

   /disk2 にマウントされているファイルシステムには、システム・ディスクのバックアップを格納する十分なディスク領域があります。/ の使用可能ブロックが 538,378 ブロックであるのに対し、*/disk2* の使用可能ブロックは 2,004,527 ブロックです。*/disk2* の *backupdevice* は */dev/dsk/dks0d2s7* で、*backupfs* は */disk2* です。
7. */etc/fstab* の一時的なコピー */etc/fstab.xfs* を作成し、任意のエディタを使用して、そのコピーを編集します。

   ```
   # cp /etc/fstab /etc/fstab.xfs
   # vi /etc/fstab.xfs
   ```

   /etc/fstab.xfs で次の変更を行います。

- root ファイルシステムの行がある場合は、root ファイルシステム "/" の行で `efs` を `xfs` に置換えます。
- root ファイルシステムの行がない場合は、次の行を追加します。

  ```
  /dev/root   /   xfs rw,raw=/dev/rroot 0 0
  ```

- 独立した root ファイルシステムと usr ファイルシステムがあり、それを使用する場合は、usr ファイルシステムの行で `efs` を `xfs` に置換えます。
- 独立した root ファイルシステムと usr ファイルシステムがある場合、変換手順の際にディスク・パーティションを再分割して 2 つのファイルシステムを結合するには、usr ファイルシステムの行は削除します。

8. *`shutdown`* コマンド、または「ツールチェスト (Toolchest)」の「システム (System)」->「システム停止 (System Shutdown)」を使用して、ワークステーションを停止します。メッセージに応答して、5 つの項目から構成される「System Maintenance」メニューにアクセスします。
9. システム・ソフトウェア CD またはソフトウェアのディストリビューション・ディレクトリからミニルートを起動します。
10. *`inst`* でシェル・プロンプトに切替えます。

    ```
    Inst> sh
    ```

11. 次のコマンドを入力して、root ファイルシステムのフル・バックアップを作成します。

    ```
    # /root/sbin/dump 0uCf tapesize dumpdevice rootpartition
    ```

    tapesize はテープの容量（ディスクへのバックアップにも使用される）で、*dumpdevice* はテープ・ドライブの適切なデバイス名、またはダンプ・イメージが格納されるファイル名です。表 6-1 に、各種のテープ・ドライブおよびディスクに対する *tapesize* と *dumpdevice* の値を示します。表 6-1 の *tapecntlr* および *tapeunit* は、5 の *tapecntlr* および *tapeunit* です。

表 6-1 ファイルシステム・バックアップ用の *dump* 引数

バックアップ・デバイス	*tapesize*	*dumpdevice*
ディスク	2m	root ファイルシステム用に /root/*backupfs*/root.dump を使用し、usr ファイルシステム用に /root/*backupfs*/usr.dump を使用します。
DAT テープ	2m	/dev/rmt/tps*tapecntlr*d*tapeunit*nsv
DLT テープ	10m	/dev/rmt/tps*tapecntlr*d*tapeunit*nsv
EXABYTE 8mm モデル 8,200 テープ	2m	/dev/rmt/tps*tapecntlr*d*tapeunit*nsv

表 6-1 ファイルシステム・バックアップ用の *dump* 引数

バックアップ・デバイス	*tapesize*	*dumpdevice*
EXABYTE 8mm モデル 8,500 テープ	4m	/dev/rmt/tps*tapecntl*rd*tapeunit*nsv
QIC カートリッジ・テープ	150k	/dev/rmt/tps*tapecntl*rd*tapeunit*ns

12. usr ファイルシステムが独立している場合は、新しいテープ（テープを使用している場合）を挿入し、次のコマンドを指定して、usr ファイルシステムのフル・バックアップを作成します。

    ```
    # /root/sbin/dump 0uCf tapesize dumpdevice usrpartition
    ```

    tapesize はテープの容量（ディスクへのバックアップにも使用される）で、*dumpdevice* はテープ・ドライブの適切なデバイス名、またはダンプ・イメージが格納されるファイル名です。表 6-1 に、各種のテープ・ドライブおよびディスクに対する *tapesize* と *dumpdevice* の値を示します。

13. シェルを終了します。

    ```
    # exit
    ...
    Inst>
    ```

14. システム・ディスクのパーティションを再分割する必要がない場合は、18 に進んでください。

15. システム・ディスクのパーティションを再分割する場合は、*fx* のスタンドアロン・バージョンを使用します。この *fx* のバージョンは、Command Monitor から起動されるので、まず Command Monitor を起動してください。このためには、*inst* を終了してシステムを再起動し、システムを停止した後、Command Monitor を呼出します。次にこの手順の例を示します。

```
Inst> quit
...
Ready to restart the system.  Restart? { (y)es, (n)o, (sh)ell, (h)elp }: yes
...
login: root
# halt
...
System Maintenance Menu
...
Option? 5
Command Monitor.  Type "exit" to return to the menu.
>>
```

グラフィカルな「System Maintenance」メニューが搭載されているシステムの場合は、オプション 5 の代わりに、メニューの 5 番目の項目の「Enter Command Monitor」を選択します。

16. *fx* を起動し、ニーズに合わせてシステム・ディスクのパーティションを再分割します。次の例では、*fx* を使用して、独立した root パーティションと usr パーティションを結合させ、単一の root パーティションに切替える方法を示します。

```
>> boot stand/fx
84032+11488+3024+331696+26176d+4088+6240 entry: 0x89f97610
114208+29264+19536+2817088+60880d+7192+11056 entry: 0x89cd31c0
Currently in safe read-only mode.
Do you require extended mode with all options available? (no) Enter
SGI Version 6.4 ARCS   Sep 29, 1996
fx: "device-name" = (dksc) Enter
fx: ctlr# = (0) Enter
fx: drive# = (1) Enter
fx: lun# = (0) Enter
...opening dksc(0,1,0)
...drive selftest...OK
Scsi drive type == SGI     SEAGATE ST31200N8640

----- please choose one (? for help, .. to quit this menu)-----
[exi]t             [d]ebug/           [l]abel/           [a]uto
[b]adblock/        [exe]rcise/        [r]epartition/     [f]ormat
fx> repartition/rootdrive

fx/repartition/rootdrive: type of data partition = (xfs) Enter
Warning: you will need to re-install all software and restore user data
from backups after changing the partition layout.  Changing partitions
will cause all data on the drive to be lost.  Be sure you have the drive
backed up if it contains any user data.  Continue? yes

----- please choose one (? for help, .. to quit this menu)-----
[exi]t             [d]ebug/           [l]abel/           [a]uto
[b]adblock/        [exe]rcise/        [r]epartition/     [f]ormat
fx> exit
```

17. 9 と同じことを実行し、再度ミニルートをロードします。

18. root に対して XFS を作成します。

```
Inst> admin mkfs /dev/dsk/dks0d1s0
Unmounting device “/dev/dsk/dks0d1s0” from directory “/root”.

Make new file system on /dev/dsk/dks0d1s0 [yes/no/sh/help]: yes

About to remake (mkfs) file system on: /dev/dsk/dks0d1s0
This will destroy all data on disk partition: /dev/dsk/dks0d1s0.

        Are you sure? [y/n] (n): y

        Block size of filesystem 512 or 4096 bytes? 4096
```

```
Doing: mkfs -b size=4096 /dev/dsk/dks0d1s0
meta-data=/dev/rdsk/dks0d1s0     isize=256    agcount=8, agsize=31021 blks
data     =                       bsize=4096   blocks=248165, imaxpact=25
         =                       sunit=0      swidth=0 blks, unwritten=1
naming   =version 1              bsize=4096
log      =internal log           bsize=4096   blocks=1168
realtime =none                   extsz=65536  blocks=0, rtextents=0
Mounting file systems:

NOTICE: Start mounting filesystem: /root
NOTICE: Ending clean XFS mount for filesystem: /root
    /dev/miniroot            on  /
    /dev/dsk/dks0d1s0        on  /root


   Re-initializing installation history database
   Reading installation history .. 100% Done.
   Checking dependencies .. 100% Done.
```

19. *inst* でシェル・プロンプトに切替えます。

    ```
    Inst> sh
    ```

20. ディスクにバックアップを作成した場合は、バックアップが格納されるファイルシステムにマウント・ポイントを作成して、そのファイルシステムをマウントします。

    ```
    # mkdir /backupfs
    # mount backupdevice /backupfs
    ```

21. テープにバックアップを作成した場合は、テープ・ドライブに正しいテープを挿入し、次のコマンドを指定して、11 で作成したバックアップから root ファイルシステムにあるすべてのファイルをリストアします。

    ```
    # cd /root
    # mt -t /dev/rmt/tpstapecntlrdtapeunit rewind
    # restore rf dumpdevice
    ```

    リストア中、しばらく待たされることがあります。通常、出力は生成されずに時間が多少かかります。

22. ディスクにバックアップを作成した場合は、次のコマンドを指定して、11 で作成したバックアップから root ファイルシステムにあるすべてのファイルをリストアします。

    ```
    # cd /root
    # restore rf /backupfs/root.dump
    ```

23. 12 でテープに usr ファイルシステムのバックアップを作成した場合は、正しいテープをテープ・ドライブに挿入し、次のコマンドを指定して、バックアップしたすべてのファイルをリストアします。

    ```
    # cd /root/usr
    # mt -t /dev/rmt/tpstapecntlrdtapeunit rewind
    # restore rf dumpdevice
    ```

24. 12 でディスクに usr ファイルシステムのバックアップを作成した場合は、次のコマンドを指定して、バックアップしたすべてのファイルをリストアします。

    ```
    # cd /root/usr
    # restore rf /backupfs/usr.dump
    ```

25. 7 で作成した */etc/fstab* の新しいバージョンを所定の位置に移動します。最初のコマンドは、*/etc/fstab* の古いバージョンを保存するためのコマンドです。このコマンドはオプションです。

    ```
    # mv /root/etc/fstab /root/etc/fstab.old
    # mv /root/etc/fstab.xfs /root/etc/fstab
    ```

26. シェルと *inst* を終了して、システムを再起動します。

```
# exit
#
Calculating sizes .. 100% Done.

Inst> quit
...
Ready to restart the system.  Restart? { (y)es, (n)o, (sh)ell, (h)elp }: yes
Preparing to restart system ...

The system is being restarted.
```

オプション・ディスク上のファイルシステム EFS から XFS への変換

注意：ここでの手順は、正しく行わないとデータが消去されてしまうことがあります。したがって、この操作は IRIX システム管理者が行うことをお薦めします。

ここでは、オプション・ディスク（システム・ディスク以外のディスク）の EFS を XFS に変換する方法について説明します。XLV 論理ボリュームが使用されていないものとします。この手順は、特権ユーザが実行してください。

1. 第 6 章の「XFS ファイルシステムの使用計画」を読み、この手順を開始できる状態かどうかを確認します。
2. バックアップが最新のものかどうかを確認します。この手順では一時的に変換するファイルシステムからすべてのファイルが削除されるので、通常のバックアップ手順を使用して完全なバックアップを作成しておくことが重要です。システム・ディスクの完全なダンプは、4 で作成しますが、この手順で作成するバックアップのほかに通常のバックアップも必要です。

3. 作成するファイルシステムのパーティションのデバイス名（*partition* 変数）を確認します。たとえば、コントローラ 0、ドライブ・アドレス 2 にあるオプション・ディスクのパーティション 7（ディスク全体）を使用する場合は、*partition* は *`/dev/dsk/dks0d2s7`* になります。*partition*（特殊ファイル）の確認については、dks(7M) マン・ページを参照してください。

4. ディスク・パーティションのファイルは、変換プロセスで破壊されるので、すべてのファイルをテープまたはディスクにバックアップします。任意のバックアップ・コマンド（*`Backup`*、*`bru`*、*`cpio`*、*`tar`* など）を使用して、ローカルまたはリモートのテープ・ドライブ、あるいはディスクにバックアップできます。たとえば、ローカル・テープにダンプするコマンドは、次のとおりです。

   # **dump 0uCf** *tapesize dumpdevice partition*

   tapesize はテープの容量（ディスクへのバックアップにも使用される）で、*dumpdevice* はテープ・ドライブのデバイス名です。表 6-1 に、各種のローカルなテープ・ドライブおよびディスクの *tapesize* と *dumpdevice* の値を示します。この表で使用される *tapecntlr* と *tapeunit* の値は、hinv -c tape コマンドの出力から確認できます。

5. パーティションをアンマウントします。

   # **umount** *partition*

6. *`mkfs`* コマンドを使用して、新しく XFS を作成します。

   # **mkfs -b size=***blocksize* **-l size=***logsize partition*

   blocksize は、ファイルシステムのブロック・サイズです。122 ページの「ファイルシステムのブロック・サイズとエクステント・サイズの選択」を参照してください。また、*logsize* はログ・レコード専用領域のサイズです。124 ページの「ログ・タイプとログ・サイズの選択」を参照してください。例 6-2 に、このコマンド行とその出力の例を示します。

7. 次のコマンドを使用して、新しいファイルシステムをマウントします。

   # **mount** *partition mountdir*

8. */etc/fstab* ファイルの *partition* のエントリで、efs を xfs に置換えます。

   partition mountdir **xfs rw,raw=***rawpartition* **0 0**

   rawpartition は、*partition* のロー・バージョンです。

9. 4 で作成したバックアップからこのファイルシステムにファイルをリストアします。たとえば、4 で *`dump`* コマンドを指定した場合、テープからファイルをリストアするためのコマンドは、次のとおりです。

   # **cd** *mountdir*
   # **mt -t** *device* **rewind**
   # **restore rf** *dumpdevice*

   device の値は *dumpdevice* と同じですが、最後に `nsv` などの文字が付きません。

リストア中、しばらく待たされることがあります。通常、出力は生成されませんが、時間は多少かかります。

XFS ファイルシステムへ変換する際のディスク空き領域チェック

ファイルが同じ場合、EFS ファイルシステムより XFS ファイルシステムのほうがより多くのディスク領域を必要とします。ディスク領域が余分に必要な理由は、ブロック・サイズが EFS の 512 バイトよりも大きいことと、XFS ログを格納する領域が必要であるためです。ただし、XFS は通常、空き領域をよりコンパクトに表し、また i ノードを動的に割当てるため、空き領域の使用は比較的少なくてすみます。

ファイルシステムを XFS へ変換した後、およそどのくらいの空き領域が残るのか、次の手順で調べることができます。

1. 変換するファイルシステムのサイズをキロバイト単位で取得し、メガバイト単位に切り上げます。以下はその例です。

   ```
   df -k
   Filesystem           Type  kbytes     use   avail %use  Mounted on
   /dev/root             efs  969857  663306  306551  68%  /
   ```

   上記のファイルシステムは 969857 KB です。切り上げると 970 MB になります。

2. 内部ログ（124 ページの「ログ・タイプとログ・サイズの選択」を参照）を使用する場合、次のコマンドを使って変換後のファイルシステムでおよそどのくらい空き領域が必要か調べることができます。

   % **xfs_estimate -i** *logsize* **-b** *blocksize mountpoint*

   logsize はログのサイズです。*blocksize* は第 6 章の「ファイルシステムのブロック・サイズとエクステント・サイズの選択」で選択したユーザ・ファイルのブロック・サイズです。*mountpoint* はファイルシステムのマウント・ポイントであるディレクトリです。たとえば、

   ```
   % xfs_estimate -i 1m -b 4096 /
   / will take about 747 megabytes
   ```

   上のコマンドの結果は、XFS への変換後、ファイルシステム内のファイル（*blocksize* が 4096 バイトの場合）と、*logsize* で指定したサイズの内部ログがディスク領域をどのくらい占めるかを示します。

3. 外部ログを使用する場合、変換後のファイルシステムのファイルに必要なディスク空き領域を、次のコマンドで調べることができます。

   % **xfs_estimate -e 0 -b** *blocksize mountpoint*

blocksize は第 6 章の「ファイルシステムのブロック・サイズとエクステント・サイズの選択」で選択したユーザ・ファイルのブロック・サイズです。*mountpoint* はファイルシステムのマウント・ポイントであるディレクトリです。たとえば、

```
% xfs_estimate -e 0 -b 4096 /
/ will take about 746 megabytes
        with the external log using 0 blocks or about 1 megabytes
```

xfs_estimate で始まる最初の行は、XFS へ変換後、ファイルシステムのファイルがどれだけディスク領域を占めるかを示しています。上記の例では、別のディスク・パーティションに外部ログ用のディスク領域が必要です。出力結果の 2 行目は無視して構いません。

4. 手順 1 のファイルシステムのサイズと手順 2 および 手順 3 のファイルのサイズを比較します。たとえば、

   ```
   970 MB - 747 MB = 223 MB free disk space
   747 MB / 970 MB = 77% full
   ```

   この比較によって、ファイルシステムを XFS に変換した場合、ディスクの空き領域が十分であるかどうかがわかります。

変換後の空き領域が十分でない場合、以下のオプションを検討します。

- 空き領域が十分ではない場合の一般的な措置（必要のないファイルの削除、ファイルをテープにアーカイブする、ファイルを別のファイルシステムへ移動する、別のディスクを追加するなど）を行う。
- ディスクのパーティションを切り直して、ファイルシステムのディスク・パーティションのサイズを大きくする。
- root ファイルシステムに十分な空き領域がない場合、ファイルシステムを root と *usr* に分けて、1 つのパーティション内で root、*usr* の両ファイルシステムを使用する。
- ファイルシステムが XLV 論理ボリュームにある場合、ボリュームのサイズを大きくする。
- ログ・サブボリュームのある XLV 論理ボリュームを別の場所に作成し、すべてのディスク領域がユーザ・ファイルに割当てられるようにする。

XFS ファイルシステムに変換した場合に必要なダンプとリストア

135 ページの「システム・ディスク上のファイルシステム EFS から XFS への変換」および 141 ページの「オプション・ディスク上のファイルシステム EFS から XFS への変換」の手順に従ってファイルシステムを変換する場合、変換するファイルシステムをテープまたは別のディスクへダンプする必要があります。ダンプ先はダンプ・イメージを格納できるだけの十分な空き領域が必要です。ダンプのスピードは、テープよりディスクのほうが高速です。

システム・ディスクを変換する際に、*dump* と *restore* コマンドを使用します。オプション・ディスク上でファイルシステムを変換する場合は、任意のバックアップおよびリストア用のコマンドを使用できます。

テープ・ドライブにダンプする場合、次のガイドラインに従ってください。

- 変換するファイルシステムのダンプに必要な十分の量のテープを用意します。
- システム・ディスク上でファイルシステムを変換している場合、テープ・ドライブはローカルである必要があります。
- オプション・ディスク上で変換している場合、テープ・ドライブはローカル、リモートのいずれも可能です。

別のファイルシステムにダンプする場合、以下の条件を満たす必要があります。

- 変換するファイルシステムは、使用中の部分が 2 GB 以下である必要があります（EFS ファイルシステムにおけるダンプ・イメージ・ファイルの最大サイズ）。ただし、XFS ファイルシステムにダンプする場合はこの限りではありません。
- ダンプ先のファイルシステムには、変換するファイルシステムを格納するのに十分な空き領域が必要です。
- システム・ディスク上でファイルシステムを変換している場合、ダンプ先のファイルシステムはローカルである必要があります。
- オプション・ディスク上でファイルシステムを変換している場合、ダンプ先のファイルシステムはローカルまたはリモートのいずれも可能です。

ファイルシステムの保守

この章では、定期的にまたは必要に応じて実行する XFS ファイルシステムを保守するための管理手順について説明します。ファイルシステムに格納されているデータをバックアップするだけでなく、ファイルシステム自体を正しく保守することがきわめて重要です。ファイルシステムの保守を正しく行わないと、大切なシステムやユーザ情報を損失する可能性があります。

この章では、次の項目について説明します。

ファイルシステムの定期的な管理タスク

ファイルシステムを管理するには、次を行う必要があります。

- 使用可能な空き領域および空き i ノードを監視します。
- 長期間ファイルシステムの空き領域が不足している場合は、古いファイルの削除やディスク使用量の割当てなどを行い、問題を軽くするようにします。
- ファイルシステムのバックアップを作成します。

多くの定期的な管理ジョブは、シェル・スクリプトで実行できます。次にその方法を示します。

- 空きブロックと空き i ノードを調査し、空き領域が一定のしきい値を下回るファイルシステムを通知します。
- サイズが拡張するファイル（ログ・ファイルなど）を自動的に消去します。
- ディスク使用率が設定を超過している場合は強調表示します。

このようなスクリプトはすべて、*cron* コマンドで自動的に実行でき、出力は電子メールで送信できます。通常、*find*、*du*、*mail* およびシェル・コマンドを組合わせて使用されます。

プロセス・アカウンティング・システムも同じような機能を実行します。プロセス・アカウンティング・システムが条件に合わない場合は、*ckpacct* や *remove* などの */usr/lib/acct* のスクリプトを調べ、固有の管理スクリプトを構築してください。

ファイルシステムのマウントおよびアンマウント

第 5 章の「ファイルシステムのマウントとアンマウント」で説明したように、IRIX でアクセスするにはファイルシステムをマウントします。次に、*mount* と *umount* コマンド、および */etc/fstab* ファイルを使用して、ファイルシステムをマウントおよびアンマウントする方法について説明します。

アドバイス：*xfsm* コマンドのグラフィック・ユーザ・インタフェースを使用して、XFS ファイルシステムをマウントおよびアンマウントできます。詳細については、オンライン・ヘルプを参照してください。

手作業によるファイルシステムのマウント

mount コマンドを使用すると、ファイルシステムを手作業でマウントできます。*mount* コマンドの基本形式は、次のとおりです。

mount *device_file mount_point_directory*

mount *host*:*directory mount_point_directory*

device_file は、ブロック・デバイス・ファイルです。*host*:*directory* は、リモート・ホスト（NFS が必要）で *exportfs* コマンドを使用して、リモート・ホストにエキスポートされているリモート・ディレクトリのホスト名およびパス名です。*mount_point_directory* は、マウント・ポイント・

ディレクトリです。*mkdir* コマンドを使用してマウント・ポイントを作成しておく必要があります。

device_file または *mount_point_directory* を *mount* コマンド行で指定しない場合は、*mount* が */etc/fstab* ファイルをチェックして、足りない引数を調べます。*/etc/fstab* の詳細については、150 ページの「/etc/fstab ファイルによるファイルシステムのオートマウント」を参照してください。

たとえば、ファイルシステムを手作業でマウントする場合は、次のコマンドを使用します。

```
mount /dev/dsk/dks0d1s6 /usr
```

また、ニーモニック・デバイス・ファイル名を使用する場合は、次のコマンドを使用します。

```
mount /dev/usr /usr
```

/etc/fstab ファイルに記述されたファイルシステムに対して、次の *mount* コマンドを使用します。

```
mount /d2
```

また、次の *mount* コマンドも便利です。

mount -a *etc/fstab* ファイルに記述されたすべてのファイルシステムをマウントします。

mount -h *host*
etc/fstab ファイルに記述されたファイルシステムのうち、*host* という名前のシステムからリモート・マウントされているファイルシステムをすべてマウントします。

mount -o quota *device_file mount_point_directory*
ファイルシステム *device_file* を *mount_point_directory* でマウントします。このとき、ディスク割当てのトラッキングはオンにします。詳細については、162 ページの「XFS ファイルシステムでのディスク割当ての使用」を参照してください。

どのファイルシステムをマウントするかを指定するには、マウント・コマンドの **-t** *type* オプションを使用します。IRIX オペレーティング・システムがサポートするファイルシステムの種類の詳細については、filesystems(4) マン・ページを参照してください。

mount コマンドの詳細については、mount(1M) マン・ページを参照してください。

/etc/fstab ファイルによるファイルシステムのオートマウント

/etc/fstab ファイルには、システムをマルチユーザ・モードで起動すると自動的にマウントされるすべてのファイルシステムおよびスワップ・パーティションに関する情報があります。また、デバイス・ブロック・ファイルまたはマウント・ポイントしか *mount* コマンドに指定されていない場合も、*/etc/fstab* ファイルが使用されます。ただし、*/proc* ファイルシステムなど、*mount* コマンドでマウントされないファイルシステムは、*/etc/fstab* に記述されません。

ここでは、ファイルシステムのエントリを */etc/fstab* に追加する方法を示します。

システムの起動時にマウントされる各ファイルシステムに対しては、*/etc/fstab* ファイルに次のような行が記述されています。

```
/dev/dsk/dks0d2s7 /test xfs rw,raw=/dev/rdsk/dks0d2s7 0 0
```

この行の各フィールドは、次のように定義されています。

`/dev/dsk/dks0d2s7`
: ファイルシステムが置かれているパーティションのブロック・デバイス・ファイル。

`/test`
: ファイルシステムがマウントされるディレクトリ名（マウント・ポイント）。

`xfs`
: ファイルシステムの種類。この例では、XFS ファイルシステムです。

`rw, raw=`
: ファイルシステムのマウント時に使用可能なオプションの一部（完全なリストについては、fstab(4) マン・ページを参照）。この例では、ファイルシステムは読込み / 書込み可能でマウントされているので、*root* およびほかのユーザはこのファイルシステムに書込みを行うことができます。raw= オプションには、ファイルシステムのロー・デバイスのファイル名を指定します。これは、オプション・リストの最後に指定する必要があります。

`0 0`
: ダンプ・サイクルの頻度と *fsck* の受渡し優先度です。この 2 つの数字は、オプション・リストの最後のオプション (raw=) の後に追加してください。詳細については、fstab(4) マン・ページを参照してください。

第 7 章の「手作業によるファイルシステムのマウント」で説明したように、ファイルシステムをすでにマウントしている場合は、*mount* コマンドを使用して、適切な */etc/fstab* のエントリを確認できます。

```
mount -p
```

このコマンドを実行すると、*/etc/fstab* 形式の新しいファイルシステムを含め、現在マウントされているすべてのファイルシステムが表示されます。新しいファイルシステムに対する行を */etc/fstab* にコピーします。

mount コマンドは、*/etc/fstab* を順に読取ります。したがって、別のファイルシステムの下にマウントされているファイルシステムは、マウント・ポイントを確定するために、*/etc/fstab* に指定されている親パーティションの後に読取られる必要があります。

システム・ディスクの スワップ・パーティション（パーティション 1）は、*/etc/fstab* に列挙されません。ただし、システムに追加された スワップ・パーティションは列挙されます。スワップ・パーティションでは、マウント・ポイント・フィールドは使用されません。詳細については、『IRIX Admin: System Configuration and Operation』および swap(1M) マン・ページを参照してください。

また、*/etc/fstab* エントリの詳細については、fstab(4) マン・ページを参照してください。

リモート・ファイルシステムのオートマウント

オプションの NFS ソフトウェアがある場合は、リモート・ファイルシステムにアクセスすると自動的にリモート・ファイルシステムをマウントできます（たとえば、*cd* コマンドを使用してディレクトリをファイルシステムに変更する場合）。リモート・ファイルシステムは、*exportfs* コマンドを使用してエキスポートする必要があります。

すべての使用可能なオプションも含め、オートマウントの設定については、automount(1M) および exportfs(1M) マン・ページを参照してください。これらのコマンドの詳細については、『ONC3/NFS Administrator's Guide』を参照してください。

ファイルシステムのアンマウント

ファイルシステムは、システムが停止すると自動的にアンマウントされます。ファイルシステムを手作業でアンマウントするには、*umount* コマンドを使用します。表 7-1 に、*umount* コマンドの基本的な 3 つの形式を示します。この表の 1 番目または 2 番目の形式のどちらかを使用すると、ローカルなファイルシステムをアンマウントできます。どちらの形式を使用しても結果は同

じです。また、リモート・ファイルシステムに対しては、1 番目または 3 番目の形式のどちらかを使用できます。

表 7-1 *umount* コマンドの形式

コマンド	説明
umount *mount_point_directory*	*mount_point_directory* はディレクトリのパス名で、ファイルシステムのマウント・ポイントです。この形式はローカル・ファイルシステムまたはリモート・ファイルシステムで使用できます。
umount *device_file*	*device_file* はブロック・デバイス・ファイル名です。この形式はローカル・ファイルシステムにのみ使用できます。
umount *host*:*directory*	*host*:*directory* はリモート・ディレクトリです。この形式はリモート・ファイルシステムにのみ使用できます。
umount -a	*/* と */usr* ファイルシステムを除いて、現在マウントされているすべてのファイルシステム（*/etc/mtab* に列挙）をアンマウントします。このコマンドは、*mount -a* コマンドを補うものではありません。*mount -a* コマンドは */etc/fstab* に列挙されているファイルシステムすべてをマウントします。

たとえば、*/d2* にマウントされているローカル・ファイルシステムまたはリモート・ファイルシステムをアンマウントするには、次のコマンドを使用します。

```
umount /d2
```

パーティション */dev/dsk/dks0d1s7* のファイルシステムをアンマウントするには、次のコマンドを使用します。

```
umount /dev/dsk/dks0d1s7
```

リモートマウントされている (NFS) ファイルシステム *depot:/usr/spool/news* をアンマウントするには、次のコマンドを使用します。

```
umount depot:/usr/spool/news
```

アンマウントを行う場合は、そのファイルシステムが使用中でないことを確認してください。ファイルシステムを使用中にアンマウントしようとすると、「Resource busy」メッセージが表示されます。エラーメッセージとその対処方法については、umount(1M) マン・ページを参照してください。

ディスク領域の管理

ある時点でディスク領域が不足する場合があります。これは、単に新しいファイルを作成する際にディスク領域を使用する以外に、不要なファイルが作成されたり、放置されているためです。

- 不要になったファイルを削除していない可能性があります。古いファイルは、しばしば必要以上に長い期間システム上にあります。
- 通常のシステム・オペレーションによってファイルのサイズが大きくなる場合があります（たとえば */var/adm/SYSLOG* などのログ・ファイル）。通常は *cron* が 1 週間に 1 回ファイルを交替し、ファイルが大きくなりすぎないように管理します（*var/spool/cron/crontabs/root* を参照）。ただし、定期的にこのファイルをチェックして、正常に管理されているかどうか、またディスクの空き領域がいつ少なくなったかを確認する必要があります。
- 特に、*/tmp*、*/usr/tmp*、*/var/tmp* ディレクトリはたくさんのファイルが蓄積されています。これらのファイルは、テキスト・エディタやそのほかのプログラムで処理されたファイルのコピーです。このようなテンポラリ・ファイルは、ファイルを作成したプログラムでは削除できない場合があります。
- */usr/tmp*、*/var/tmp*、*/var/spool/uucppublic* の各ディレクトリは、共有ディレクトリです。これらのディレクトリは、ほかのシステムやサイトと受渡されるファイルの一時的なコピーを格納するために使用します。これらのディレクトリは、*/tmp* ディレクトリとは異なり、システムが再起動されてもディレクトリ内のファイルは削除されません。サイトの管理者は、このようなディレクトリで使用されているディスクの空き領域を入念に監視してください。
- 古いファイルを dumpster（ごみ箱）にドラッグしても、これらのファイルはシステムから完全に削除されません。
- */var/adm/crash* の *vmcore* と *unix* が削除されずに蓄積されています。
- クラッシュしたアプリケーション・プログラムからのバイナリ・コア・ダンプである *core* ファイルが削除されていません。

アドバイス：『Personal System Administration Guide』の 第 6 章「ディスク領域の解放」に、不要なファイルを識別する方法が示されています。

次に、ディスクの空き領域のモニタリング、不要なファイルの検索、ユーザ別のディスク使用量の制限方法について説明します。

空き領域と空き i ノードのモニタリング

df コマンドを使用すると、空き領域と空き i ノードをすぐにチェックできます。

```
% df
Filesystem                Type  blocks     use   avail %use  Mounted on
/dev/root                  xfs 1939714 1326891  612823  68%  /
```

avail カラムに、空き領域がブロック単位で表示されます。

空き i ノード数を確認するには、次のコマンドを使用します。

```
% df -i
Filesystem      Type  blocks     use   avail %use    iuse  ifree %iuse  Mounted
/dev/root        xfs 1939714 1326891  612823  68%   14491 195031    7%  /
```

最初の *df* リストと似たリストが表示されます。ただしこのリストには、使用中の i ノード数、空き i ノード数（使用可能）、および使用中の i ノードの比率も表示されます。XFS ファイルシステムでは、空き i ノード数は、必要に応じて割当てることができる最大値になります。XFS は必要なだけの、i ノードを割当てます。XFS では、i ノードの使用率は使用量がかなり多いファイルシステムでのみ非常に高くなります。XFS ファイルシステムの使用量がかなり多くても XFS の性能は落ちません。

キー・ファイルとディレクトリのモニタリング

ほとんどのシステムには、通常のシステム・オペレーションによってファイルのサイズが大きくなるキー・ファイルおよびディレクトリがあります。表 7-2 に、この例を示します。

表 7-2 拡張されるファイルまたはディレクトリ

ファイル	説明
/etc/wtmp	システム・ログインの履歴
/tmp	テンポラリ・ファイルのディレクトリ（root ファイルシステム）
/var/adm/avail/availlog	使用可能なモニタのログ・ファイル（availmon(5) マン・ページを参照）
/var/adm/avail/notifylog	使用可能なモニタのログ・ファイル（availmon(5) マン・ページを参照）
/var/adm/sulog	*su* コマンドの履歴
/var/cron/log	*cron* の動作の履歴
/var/spool/lp/log	*lp* の動作の履歴

表 7-2 拡張されるファイルまたはディレクトリ **（続き）**

ファイル	説明
/var/spool/uucp	*uucp* ログ・ファイルのディレクトリ
/var/tmp	テンポラリ・ファイルのディレクトリ

拡張ファイルをチェックする頻度は、システムの稼働状況およびディスク領域の問題がどの程度深刻かによって異なります。*tail* コマンドと *mv* コマンドを組合わせて使用すると、拡張ファイルを適切なサイズに保持できます。

```
# tail -50 /var/adm/sulog > /var/tmp/sulog
# mv /var/tmp/sulog /var/adm/sulog
```

このシーケンスでは、*/var/adm/sulog* の最後の 50 行をテンポラリ・ファイルに格納し、そのテンポラリ・ファイルを */var/adm/sulog* に移動します。この結果、ファイルの内容を最新の 50 エントリに減らします。*cron* を使用して、毎週自動的にこれらのコマンドを実行するようにしておくと便利です。*cron* を使用した標準のタスクの自動化については、cron(1M) マン・ページを参照してください。

テンポラリ・ディレクトリ内の消去

/tmp ディレクトリとそのすべてのサブディレクトリ内は、システムを再起動すると自動的に消去されます。これは、*chkconfig* コマンドの **nocleantmp** オプションで制御します。デフォルトでは、**nocleantmp** はオフに設定されているので、*/tmp* 内は削除されます。

/var/tmp ディレクトリ内は、システムを再起動しても自動的に消去されません。これは IRIX システムの標準設定です。必要に応じてシステムを再起動する際に */var/tmp* 内が自動的に消去されるように IRIX を設定できますが、通常この標準設定は変更しません。多くのユーザは、システムを再起動しても */var/tmp* ディレクトリのファイルは削除されないと想定しています。したがって、設定を変更する場合は、必ず前もってユーザに通知してください。

システムの再起動の際に */var/tmp* のファイルを自動的に消去するよう IRIX を設定するには、次の手順に従ってください。

1. */var/tmp* ディレクトリの標準設定を変更すること、およびシステムを再起動すると */var/tmp* のファイルがすべて削除されることをシステムを使用するすべてのユーザに通知します。このとき、電子メールを送り、*/etc/motd* ファイルにメッセージを置きます。

   この変更は、少なくとも 1 週間前にはユーザに知らせてください。

2. */etc/init.d/rmtmpfiles* ファイルを同じディレクトリ内の新しいファイル（*/etc/init.d/rmtmpfiles2* など）にコピーします。

```
# cd /etc/init.d
# cp rmtmpfiles rmptmpfiles2
```

3. *rmtmpfiles2* を開いて編集します。

```
# vi rmtmpfiles2
```

4. ファイル内で次のようなコマンド・ブロックを検索します。

```
# make /var/tmp exist
if [ ! -d /var/tmp ]
then
        rm -f /var/tmp # remove the directory
        mkdir /var/tmp
fi
```

5. fi ステートメントの前に、次の行を追加します。

```
else
        # clean out /var/tmp
        rm -f /var/tmp/*
```

完全なコマンド・ブロックは、次のようになります。

```
# make /var/tmp exist
if [ ! -d /var/tmp ]
then
        rm -f /var/tmp # remove the directory
        mkdir /var/tmp
else
        # clean out /var/tmp
        rm -f /var/tmp/*
fi
```

6. ファイルを保存して、エディタを終了します。
7. */etc/rc2.d* ディレクトリ内の新しいファイルに対するリンクを */etc/init.d/README* に記述されている命名規則に従って作成します。

```
# cd ../rc2.d
# ln -s ../init.d/rmtmpfiles S59rmtmpfiles2
```

未使用のファイルの検索

ファイルシステムを整理するには、最近使用していないファイルを検索し、削除します。最近アクセスされていないファイルは、*find* コマンドで検索できます。

find コマンドは、コマンド行で指定したディレクトリから順に、検索条件に合ったファイルを検索します。検索条件には、すべての標準ファイル、*.trash* で終わるすべてのファイル、指定した日付以前のファイルなどを指定できます。*find* コマンドは、検索条件を満たすファイルを

見つけると、指定されたタスク（ファイルの削除、ファイル名の出力、ファイルのパーミッションの変更など）を実行します。

次に例を示します。

```
# find /usr -local -type f -mtime +60 -print > /usr/tmp/deadfiles &
```

この例で使用されるパラメータは、次のとおりです。

/usr *find* コマンドの開始パス名を指定します。

-local ローカル・システムのファイルのみを検索します。

-type f *find* コマンドに対して、標準ファイルだけを検索して、特殊なファイル、ディレクトリおよびパイプは無視するように指定します。

-mtime +60 60 日以内に変更されなかったファイルのみを検索します。

-print **-type** または **-mtime** を満たすファイルが見つかった場合に、そのパス名を出力するように指定します。

> /usr/tmp/deadfiles &
テンポラリ・ファイル */usr/tmp/deadfiles* に出力を行い、バックグラウンドで実行します。出力が大きくなる場合は、検索結果をファイルにリダイレクトすることをお薦めします。

また、次に示す例では、*find* コマンドを使用して、テンポラリ・ディレクトリ内で 7 日以上経過したファイルを検索して、そのファイルを削除します。この場合に使用するコマンドは、次のとおりです。

```
# find /var/tmp -local -type f -atime 7 -exec rm {} \;
# find /tmp -local -type f -atime 7 -exec rm {} \;
```

次の例では、*find* コマンドを使用して、1 週間以上経過したすべてのコア・ファイルを検索し、削除する方法を示します。

```
# find  / -local -type f -name core -atime +7 -exec rm {} \;
```

cron コマンドを使用してファイルの検索プロセスや削除プロセスを自動化する方法については、cron(1M) マン・ページを参照してください。

ディスク領域を多く使用するアカウントの識別

ディスクを多く使用するアカウントを調べるには、*du*、*find*、割当てコマンド、および *diskusg* というコマンドを使用します。

du コマンドによるディスク領域のチェック

du コマンドは、次に示すようにファイルおよびディレクトリのディスクの使用量をブロック単位で表示します。

```
# du /usr/share/catman/u_man
5       /usr/share/catman/u_man/cat1/audio
266     /usr/share/catman/u_man/cat1/Xm
1956    /usr/share/catman/u_man/cat1/X11
72      /usr/share/catman/u_man/cat1/Inventor
413     /usr/share/catman/u_man/cat1/dmedia
752     /usr/share/catman/u_man/cat1/explorer
12714   /usr/share/catman/u_man/cat1
1       /usr/share/catman/u_man/cat3/audio
63      /usr/share/catman/u_man/cat3
12      /usr/share/catman/u_man/cat6/video
1077    /usr/share/catman/u_man/cat6
92      /usr/share/catman/u_man/cat2
425     /usr/share/catman/u_man/cat4
170     /usr/share/catman/u_man/cat5
13      /usr/share/catman/u_man/cat1m
14557   /usr/share/catman/u_man
```

/usr/share/catman/u_man ディレクトリ内のすべてのディレクトリのブロック数が表示されます。デフォルトでは、*du* コマンドは 512 バイト・ブロック単位でディスクの使用量を表示します。1,024 バイト・ブロック単位でディスクの使用量を表示するには、**-k** オプションを指定します。

```
# du -k /usr/people/ralph
```

-s オプションを指定すると、特定のディレクトリにおけるディスクの使用量の概要が表示されます。

```
# du -s /usr/people/alice
```

du コマンドおよびそのオプションの詳細については、du(1M) マン・ページを参照してください。

find コマンドによるディスク使用量のチェック

find コマンドは、指定したサイズを超えたファイルを検索するために使用します。

```
# find /usr -local -type f -size +10000 -print
```

この例では、usr ファイルシステムで 512 バイト・ブロックが 10,000 個以上あるすべてのファイルおよびディレクトリのパス名が表示されます。

ディスク割当てアカウンティングによるディスク使用量のモニタリング

第 5 章の「ディスクの割当て」で説明したように、ディスク割当てシステムを使用すると、ディスク使用量の強制的な制限を設定せずに、ディスク使用量を監視できます。ディスク割当てのアカウンティングはユーザ単位、またはプロジェクト単位で行えます。

XFS ファイルシステムに対して次のようなコマンドを使用することで、ディスク使用量アカウンティングの自動開始、終了、およびディスク使用量の報告ができます。

- ユーザ単位のディスク使用量アカウンティングを自動的に開始するには、*/etc/fstab* ファイルのエントリに次のように *qnoenforce* オプションを挿入します。

  ```
  /dev/root / xfs rw,qnoenforce,raw=/dev/rroot 0 0
  ```

  プロジェクト単位のディスク使用量アカウンティングを自動的に開始するには、*/etc/fstab* ファイルのエントリに次のように p*qnoenforce* オプションを挿入します。

  ```
  /dev/root / xfs rw,pqnoenforce,raw=/dev/rroot 0 0
  ```

- ファイルシステムをマウントするときに、root 以外のファイルシステムでユーザ単位のディスク使用量アカウンティングを手動でオンにするには、次の *mount* コマンドを使用します。

  ```
  # mount -o qnoenforce fsname rootdir
  ```

  fsname はファイルシステムのデバイス名です。*rootdir* はファイルシステムがマウントされるディレクトリです。

  ファイルシステムをマウントするときに、root 以外のファイルシステムでプロジェクト単位のディスク使用量アカウンティングを手動でオンにするには、次の *mount* コマンドを使用します。

  ```
  # mount -o pqnoenforce fsname rootdir
  ```

- root ファイルシステムでユーザ単位のディスク使用量アカウンティングを手動でオンにするには、次のコマンドを実行します。最初の *quotaon* コマンドは強制的ディスク使用量アカウンティングを開始します。次の *quotaoff -o* コマンドは強制的な使用量制限を解除しています。

  ```
  # /usr/etc/quotaon -v /
  # /usr/etc/quotaoff -v -o enforce /
  # reboot
  ```

  root ファイルシステムでプロジェクト単位のディスク使用量アカウンティングを手動でオンにするには、次のコマンドを実行します。

  ```
  # /usr/etc/quotaon -v -o pquota /
  # /usr/etc/quotaoff -v -o pqenforce /
  # reboot
  ```

- ファイルシステムでユーザ単位のディスク使用量アカウンティングを停止するには、次のコマンドを実行します。

```
# /usr/etc/quotaoff fsname
```

プロジェクト単位のディスク使用量アカウンティングを停止するには、次のコマンドを実行します。

```
# /usr/etc/quotaoff -o pquota fsname
```

- ディスク使用量についての情報を取得するには、160 ページの「quot コマンドによるディスク使用量のチェック」と 159 ページの「ディスク割当てアカウンティングによるディスク使用量のモニタリング」で説明されているコマンドを使用します。

quot コマンドによるディスク使用量のチェック

quot コマンドは、ファイルシステムのユーザごとのディスク使用量を出力します。*quot* コマンドはディスク割当てシステムの一部ですが、このコマンドを使用するために、ディスク割当てシステムを使用する必要はありません。XFS ファイルシステムに対しては、ディスク割当てシステムを強制的な使用制限を設定せずにオンにする必要があります。手順については、159 ページの「ディスク割当てアカウンティングによるディスク使用量のモニタリング」を参照してください。

quot コマンドの出力を使用して、ユーザに各自のディスク使用量を通知できます。次にディスク使用量を表示するコマンドの例を示します（これは root ファイルシステムの例です）。

```
# /usr/etc/quot /
/dev/root (/):
  371179    root
  265712    ellis
   12606    aevans
    7927    demos
    5526    bin
    2744    lp
     682    uucp
     379    guest
     207    adm
       7    sys
```

quota コマンドによる XFS ファイルシステムのディスク使用量のチェック

quota コマンドを使用すると、ユーザ単位、またはプロジェクト単位でファイルシステムのディスク使用量と、ディスク使用制限の設定についての報告を見ることができます。*XFS* ファイルシステムでは、この機能を使用するには（たとえ使用量の強制的な制限を行わなくても）ディスク割当て機能をオンにする必要があります。使用量の強制的な制限を行わずにディスク使用量を監視する方法については、*159* ページの「ディスク割当てアカウンティングによるディスク使用量のモニタリング」を参照してください。

quota コマンドの出力に関する説明は、165 ページの「XFS ファイルシステム上でのプロジェクト単位ディスク割当て制限の設定」を参照してください。

diskusg コマンドによるディスク使用量のチェック

diskusg コマンドは、プロセス・アカウンティング・サブシステムの一部で、*quot* と同じ目的で実行します。ただし、*diskusg* コマンドは、通常は一般的なシステム・アカウンティングの一部として使用されます。このコマンドを使用すると、各ユーザのディスクの使用量に関する情報が生成されます。

```
# /usr/lib/acct/diskusg /dev/root
0       root    736795
2       bin     11035
3       uucp    1342
4       sys     9
5       adm     1011
9       lp      5418
126     ellis   528263
993     demos   15737
998     guest   740
5315    aevans  24836
```

diskusg コマンドは、*/etc/passwd* ファイルで識別されるユーザごとに 1 行ずつ情報を出力します。出力内容は、ユーザの UID 番号、ログイン名、そのアカウントで現在使用されているディスク（512 バイト・ブロック単位）の合計です。

diskusg コマンドの出力は、通常 *acctdisk*（*acct*(1M) マン・ページを参照）の入力となりますです。*acctdisk* はディスクのアカウンティング・レコードの合計を生成します。またこの合計は、ほかのアカウンティング・レコードとマージさせることもできます。アカウンティング・サブシステムの詳細については、『IRIX Admin: Backup, Security, and Accounting』および *acct*(4) マン・ページを参照してください。

Root ファイルシステムでの空き領域の不足

root ファイルシステムと usr ファイルシステムが独立しているシステムでは、次の場合に、root ファイルシステムでディスク領域が不足することがあります。

- root ファイルシステムにファイルを設定する新しいソフトウェア・オプションがインストールされた場合。
- root ファイルシステムに、追加のディスク領域を必要とする新しい IRIX リリースがインストールされた場合。

- ファイルシステムのアンマウント時に、作成されたファイルが目的のファイルシステムではなく、root ファイルシステムに偶然に格納された場合。たとえば、usr ファイルシステムのアンマウント時に */usr/tempfile* ファイルが作成されたとします。usr ファイルシステムが */usr* にマウントされると、*/usr/tempfile* ファイルにアクセスできなくなりますが、ディスクは使用されます。
- */tmp* にファイルを作成するアプリケーションで、多数のファイルまたは非常に大きなファイルが作成され、root ファイルシステムの領域を占有する場合。

次に、root ファイルシステムの領域が一杯になったときの対処方法を示します。

- 隠しファイルがあるかどうかをチェックします。root ファイルシステム以外のファイルシステムをアンマウントして（ミニルートから行うのが最も簡単）、各マウント・ポイント・ディレクトリの内容をリスト表示します。
- */lost+found* ディレクトリをチェックします。このディレクトリに大きなファイルが蓄積されている場合があります。
- root ファイルシステムと usr ファイルシステムを結合するか、usr ファイルシステムのディスク領域を root ファイルシステムに渡して、root ファイルシステムのサイズを増やします。
- */tmp* にファイルを作成しているアプリケーションで、問題の原因になるアプリケーションを調べ、テンポラリ・ファイルを */tmp* ではなく */usr/tmp* に格納するように設定を変更します。ほとんどのアプリケーションは、*TMPDIR* 環境変数を認識できるので、デフォルト以外のディレクトリを指定します。次に、*csh* の場合の例を示します。

  ```
  % setenv TMPDIR /usr/tmp
  ```

  sh では次のように実行します。

  ```
  % TMPDIR=/usr/tmp ; export TMPDIR
  ```

- */tmp* をマウントしているファイルシステムにします（第 5 章の「サブディレクトリとしてのファイルシステムのマウント」を参照）。必要に応じて、ほかのファイルシステムから */tmp* ファイルシステムを切分けることができます。

XFS ファイルシステムでのディスク割当ての使用

ここでは、XFS ファイルシステムでディスク割当てを管理する基本コマンドについて説明します。これ以外のコマンドについては、*quota*(1)、*edquota*(1M)、*quot*(1M)、および *repquota*(1M) マン・ページを参照してください。

ディスク割当てはユーザ単位で、またはプロジェクト ID を使ってプロジェクト単位で設定できます。プロジェクト ID の内容と設定方法については『IRIX Admin: Backup, Security, and Accounting』を参照してください。

XFS ファイルシステムでは、最初にファイルシステムに対するディスク割当て機能をオンにする必要があります。その後で、ユーザ単位またはプロジェクト単位で割当てを設定していきます。

XFS ファイルシステムでユーザ単位のディスク割当てをオンにする

ユーザ単位のディスク割当ては、次の方法でオンにできます。

- ユーザ単位のファイルシステムのディスク割当てを自動的にオンにするには、*/etc/fstab* のエントリに *quota* オプションを挿入します。例を次に示します。

  ```
  /dev/root / xfs rw,quota,raw=/dev/rroot 0 0
  ```

- root 以外のファイルシステムでユーザ単位のディスク割当てを手動でオンにするには、次のコマンドでファイルシステムをマウントします。

  ```
  # mount -o quota fsname rootdir
  ```

  fsname はファイルシステムのデバイス名です。*rootdir* はファイルシステムをマウントするディレクトリ名です。

- root ファイルシステムでユーザ単位のディスク割当てを手動でオンにするには、次のコマンドを実行します。

  ```
  # /usr/etc/quotaon -v /
  # reboot
  ```

XFS ファイルシステムでプロジェクト単位のディスク割当てをオンにする

プロジェクト単位のディスク割当て機能は、次の方法でオンにできます。

- プロジェクト単位のファイルシステムのディスク割当てを自動的にオンにするには、*/etc/fstab* のエントリに *pquota* オプションを挿入します。例を次に示します。

  ```
  /dev/root / xfs rw,pquota,raw=/dev/rroot 0 0
  ```

- root 以外のファイルシステムでプロジェクト単位のディスク割当て機能を手動でオンにするには、次のコマンドでファイルシステムをマウントします。

  ```
  # mount -o pquota fsname rootdir
  ```

  fsname はファイルシステムのデバイス名です。*rootdir* はファイルシステムをマウントするディレクトリ名です。

- root ファイルシステムでディスク割当て機能を手動でオンにするには、次のコマンドを実行します。

  ```
  # /usr/etc/quotaon -o pquota -v /
  # reboot
  ```

XFS ファイルシステム上でのユーザ単位ディスク割当て制限の設定

ファイルシステムでディスク割当て機能をオンにした後、次のコマンドを使用して、ファイルシステム上のユーザに対して制限を設定できます。**-n** オプションを追加すると、これらの各コマンドの結果をプレビューできます。**-n** オプションは、ドライラン・オプションです。

- ユーザの制限を対話形式で指定するには、次のコマンドを実行します。

  # **/usr/etc/edquota** *name* ...

  name はユーザ ID です。画面が消去され、EDITOR 環境変数で指定されるエディタ (*$EDITOR* が設定されていない場合は *vi*) に切替わります。このエディタで、*rootdir* でマウントされたファイルシステムについて、コマンド行の先頭に示されているユーザのディスク割当てを編集します。次のように表示されます。

  ```
  fs rootdir kbytes (soft = 0, hard = 0) inodes (soft = 0, hard = 0)
  ```

  soft と hard の値の最初のペアは、*rootdir* におけるファイルシステムの、ディスク使用量のソフト・リミットとハード・リミットをキロバイトの単位で表したものです。soft と hard の値の 2 番目のペアは、ファイルシステム内でユーザが所有できるファイル数のソフト・リミットとハード・リミットです。

  0 の値を編集して、目的の制限値を設定します。制限値が 0 の場合、制限は設定されません。制限値を設定したら、ファイルを保存し、エディタを終了します。コマンド行に複数のユーザ名 (*name)* を指定した場合は、次のユーザに対してエディタが表示されます。この行を編集して、2 番目のユーザの制限値を入力します。すべてのユーザ について行の編集を続けます。

- ユーザが別のユーザ (*proto_name*) と同じ制限値を持つことを指定するには、次のコマンドを実行します。

  # **/usr/etc/edquota -p** *proto_name name...*

- 対話形式にしないでユーザに制限値を設定するには、次のコマンドを実行します。

  # **/usr/etc/edquota -f** *rootdir* **-l **
  uid=*userid***,bsoft=***value***,bhard=***value***,isoft=***value***,ihard=***value*

  userid は数値のユーザ ID です。各 *value* はソフト・リミットまたはハード・リミットで、単位はキロバイトです。

- *edquota* コマンドへの入力として、コマンド *repquota -e* (167 ページの「XFS ファイルシステム上でのディスク割当ての管理」を参照) で作成されたファイル (*quotafile*) を使用するには、次のコマンドを実行します。

  # **/usr/etc/edquota -i** *quotafile*

XFS ファイルシステム上でのプロジェクト単位ディスク割当て制限の設定

ファイルシステムに対するディスク割当てをオンにした後で、そのファイルシステムでプロジェクト単位での使用量制限が設定できます。プロジェクトに対する使用量制限は、ユーザに対する制限と同様に、*edquota* コマンドを使って行えます。164 ページの「XFS ファイルシステム上でのユーザ単位ディスク割当て制限の設定」を参照してください。

edquota コマンドを使ってプロジェクト単位の使用量制限を設定するには、コマンドラインに *-j* オプションを挿入します。*edquota* コマンドで *-j* オプションを指定すると、コマンドライン上で指定されるすべての名前はプロジェクト名とみなされます。たとえば、プロジェクトに対する使用量制限を対話形式で指定するには、次のコマンドを実行します。

```
# /usr/etc/edquota -j name ...
```

name はプロジェクト *ID* です。*edquota* コマンドの詳細については *edquota*(1M) マン・ページを参照してください。

XFS のディスク割当て情報の表示

次に、ディスク割当てについての情報を表示するいくつかのコマンドを示します。

- 各ファイルシステムについてディスク割当て機能がオンまたはオフであることを示すレポートを表示するには、特権ユーザとして次のコマンドを実行します。

```
# /usr/etc/repquota -sa
/dev/xlv/g (/g):
----------------------
Status
        user quota accounting       : on
        user quota limit enforcement: on
        proj quota accounting       : on
        proj quota limit enforcement: on
Quota Storage
        user quota inum 67, blocks 2, extents 2
        proj quota inum 68, blocks 2, extents 2
Default Limits
        blocks time limit: 1.0 week
        files  time limit: 1.0 week
Cache
        dquots currently cached in memory: 4
```

出力の各セクションの内容は次のとおりです。

Status　このファイルシステムにおけるディスク使用量アカウンティング（on または off）とディスク割当ての制限（on または off）のステータスを表示します。

Quota Storage　blocks と extents は、ディスク割当て情報を保存するために使用するファイルシステム・ブロック数とエクステント数です。inum 値は、割当て情報が保存される i ノード番号で、内部でだけ使用されます。

Default Limits　blocks time limit と files time limit は、ユーザがディスク領域の使用量またはファイル数をそのソフト・リミット以下に抑えなければならない、デフォルトの時間の長さです。これらの時間制限は、コマンド *edquota -t* でユーザ単位に設定できます。

Cache　このセクションは内部でのみ使用されます。

- ディスク割当てについての情報を取得するには、次のコマンドを実行します。

```
# quota -v
Disk quotas for margo (uid 1606):
Filesystem    usage   quota   limit    timeleft  files   quota   limit   timeleft
/            138360       0       0              14971       0       0
/e          4156360   41200       0    1.6 days 222264       0       0
```

この出力のカラムの内容は次のとおりです。

Filesystem　ディスク割当て機能がオンの各ファイルシステムをリスト表示します。

usage　各ファイルシステム上でのユーザのディスク使用量をリスト表示します。

quota　各ファイルシステム上でのディスク使用量またはファイル数についてのユーザのソフト・リミットです。

limit　各ファイルシステム上でのディスク使用量またはファイル数についてのユーザのハード・リミットです。

timeleft　ディスク使用量またはファイル数についてのユーザのソフト・リミットが超えているファイルシステムに対して、ユーザが追加のディスク領域を使用したり、より多くのファイルを作成することが禁止されるようになるまでの日数を表示します。

files　各ファイルシステム上のユーザが所有するファイル数です。

- プロジェクトに対するディスク割当て情報を取得するには、次のコマンドを実行します。

```
# quota -j -v
Disk quotas for xfsproj (projid 260):
Filesystem   usage   quota   limit   timeleft  files   quota  limit   timeleft
/sprite01      230       0       0               17       0      0
```

- すべてのユーザのディスク使用量とディスク割当てについての情報を取得するには、次のコマンドを実行します。

```
# /usr/etc/quot -a
```

XFS ファイルシステム上でのディスク割当ての管理

ダンプするファイルシステムがディスク割当てを使用している場合、*xfsdump* は *repquota*(1M) を使って、ダンプするファイルシステムのルートに *xfsdump_quotas* という名前のファイルにディスク割当て情報を保存します。このファイルはダンプに含まれます。リストア時は、*edquota*(1M) を使うことでファイルシステムの割当て情報を復元できます。ただし元とは異なるパーティションやシステムにダンプをリストアするときは、*xfsdump_quotas* ファイルの UID やファイルシステム情報を書き替える必要があります。

すべてのファイルシステムの現在のユーザ単位割当て制限値を列挙するファイルを作成するには、特権ユーザとして次のコマンドを実行します。

```
# /usr/etc/repquota -a -e quotafile
```

すべてのファイルシステムの現在のプロジェクト単位割当て制限値を列挙するファイルを作成するには、特権ユーザとして次のコマンドを実行します。

```
# /usr/etc/repquota -j -a -e quotafile
```

EFS ファイルシステムでディスク割当て機能をこれまで使用してきた場合は、EFS で使用できるいくつかの割当てコマンドが、XFS ファイルシステムでは使用できないことに注意してください。使用できないコマンドは次のとおりです。

- *quotacheck*。*quotacheck* を手動で実行する必要はありません。
- *chkconfig quota on* と *chkconfig quota off*。ディスク割当て機能は、マウントの際にオンになります。このため、*chkconfig* ではなく *mount* オプションでディスク割当て機能のオンとオフを制御します。
- *chkconfig quotacheck on* と *chkconfig quotacheck off*。XFS では *quotacheck* は使用されないため、*chkconfig* コマンドの使用による影響はありません。
- */etc/init.d/quotas start*。このコマンドを実行しても、XFS でディスク割当てトラッキングに影響しません。
- *touch quotas*。各ファイルシステムのルート・ディレクトリに *quotas* というファイルを作成する必要はありません。割当て情報は XFS ファイルシステム構造の中に隠されています。
- 特権ユーザ以外による *repquota*。XFS では、特権ユーザだけが *repquota* コマンドを使用できます。

xfs_copy による XFS ファイルシステムのコピー

XFS を内部ログを含めてコピーするには *xfs_copy* コマンドが使用できます (外部ログを含む XFS ファイルシステムやリアルタイム・サブボリュームは *xfs_copy* コマンドではコピーできません)。1 つまたは複数のコピーがディスク・パーティション、論理ボリューム、またはファイル上に作成できます。それぞれのコピーは固有のファイルシステム識別子を与えられるため、同一システム上で個別のファイルシステムとして実行できます (*dd* のようにブロック単位コピーを行わないプログラムは、固有のファイルシステム識別子を生成しません)。複数のコピーは並列に作成されます。*xfs_copy*(1M) の詳細は マン・ページを参照してください。

次に、*xfs_copy* コマンドの例を示します。

```
# xfs_copy /dev/dsk/dks0d3s7 /dev/dsk/dks5d2s7
... 10%  ... 20%  ... 30%  ... 40%  ... 50%  ... 60%  ... 70%  ... 80%  ...
90%  ... 100%
Done.
All copies completed.
```

xfs_check と xfs_repair による XFS ファイルシステムの整合性チェック

xfs_check コマンドおよび xfs_repair コマンドのドライラン・モードを使用すると XFS ファイルシステムの整合性チェックを行えます。また、xfs_repair コマンドでは、ファイルシステムの整合性に関する問題を修復することも時々できます。

ファイルシステムの整合性チェック

XFS ファイルシステムの整合性をチェックするコマンドは、*xfs_check* と *xfs_repair -n* です。*fsck* は EFS ファイルシステムでだけ使用されます。*fsck* と異なり、*xfs_check* も *xfs_repair* もシステム起動時に自動的に起動されることはありません。これらのコマンドは、ファイルシステムの整合性に問題があると思われるときだけ使用してください。

xfs_check または *xfs_repair -n* を実行する前に、まず、チェックするファイルシステムをアンマウントしてください。アンマウントは、通常のシステム管理手順（*unmount* コマンドまたはシステム停止）を完全に行う必要があります。クラッシュやシステム・リセットによるアンマウントは認められません。ファイルシステムが完全にアンマウントされていない場合は、これを完全にマウントし、アンマウントしてから *xfs_check* または *xfs_repair -n* を実行します。

xfs_repair -n は、XFS ファイルシステムの整合性をチェックします。*xfs_repair -n* は、*xfs_check* よりも複雑なチェックを行いますが、拡張属性を持つファイルシステムや、XLV リアルタイム・サブボリューム上のファイルシステムをチェックすることはできません。*xfs_repair -n* のコマンド行は次のとおりです。

```
# xfs_repair -n device
```

device は、XFS ファイルシステムを格納しているディスク・パーティションまたは論理ボリュームのデバイス・ファイルです。*/dev/xlv/xlv0* などがこの例です。

整合性に問題がない場合の出力例を次に示します。

```
Phase 1 - find and verify superblock...
Phase 2 - scan filesystem freespace and inode maps...
        - found root inode chunk
Phase 3 - for each AG...
        - scan (but don't clear) agi unlinked lists...
        - process known inodes and perform inode discovery...
        - process newly discovered inodes...
        - agno = 0
        - agno = 1
        ...
Phase 4 - check for duplicate blocks...
        - setting up duplicate extent list...
        - check for inodes claiming duplicate blocks...
        - agno = 0
        - agno = 1
        ...
No modify flag set, skipping phase 5
Phase 6 - check inode connectivity...
        - traversing filesystem starting at / ...
        - traversal finished ...
        - traversing all unattached subtrees ...
        - traversals finished ...
        - moving disconnected inodes to lost+found ...
Phase 7 - verify link counts...
No modify flag set, skipping filesystem flush and exiting.
```

xfs_check も XFS の整合性をチェックします。xfs_check は、拡張属性を持つファイルシステムでも使用できます（attr(1) マン・ページを参照）。これに対して *xfs_repair* は、拡張属性について限定的なチェックだけを行います。*xfs_check* のコマンド行は次のとおりです。

```
# xfs_check device
```

整合性の問題が検出されない場合、メッセージは何も表示されません。

整合性のないファイルシステムの修復

-n オプションが指定されていない *xfs_repair* コマンドは、XFS ファイルシステムの整合性をチェックし、問題が検出された場合、可能であればそれを修復します。まず、チェックおよび修復するファイルシステムをアンマウントしてください。アンマウントは、通常のシステム管理手順（*unmount* コマンドまたはシステム停止）で完全に行う必要があります。クラッシュやシステム・リセットによるアンマウントは認められません。ファイルシステムが完全にアンマウントされていない場合は、これを完全にマウントし、アンマウントしてから *xfs_repair* を実行します。

非整合性が検出されたときにこれを修復する *xfs_repair* のコマンド行は次のとおりです。

```
# xfs_repair device
```

device は、XFS ファイルシステムを格納しているディスク・パーティションまたは論理ボリュームのデバイス・ファイルです。*/dev/xlv/xlv0* などがこの例です。これはマウントされていません。

問題がないファイルシステム上で *xfs_repair* を実行したときに表示される出力の例を以下に示します。

```
Phase 1 - find and verify superblock...
Phase 2 - zero log...
        - scan filesystem freespace and inode maps...
        - found root inode chunk
Phase 3 - for each AG...
        - scan and clear agi unlinked lists...
        - process known inodes and perform inode discovery...
        - agno = 0
        - agno = 1
        ...
        - process newly discovered inodes...
Phase 4 - check for duplicate blocks...
        - setting up duplicate extent list...
        - clear lost+found (if it exists) ...
        - check for inodes claiming duplicate blocks...
        - agno = 0
        - agno = 1
        ...
Phase 5 - rebuild AG headers and trees...
        - reset superblock counters...
Phase 6 - check inode connectivity...
        - ensuring existence of lost+found directory
        - traversing filesystem starting at / ...
        - traversal finished ...
        - traversing all unattached subtrees ...
        - traversals finished ...
```

```
        - moving disconnected inodes to lost+found ...
Phase 7 - verify and correct link counts...
done
```

整合性がないシステム上での *xfs_repair* の使い方の詳細については、171 ページの「XFS ファイルシステムの問題の修復」を参照してください。

fpck による外部ファイルシステムの整合性のチェック

IRIX オペレーティング・システムには、hfs(mac) および dos(fat) ファイルシステムのチェックと修復を行う *fpck* コマンドが用意されています。深刻なセクタの破損や修復不可能なエラーなどのファイルシステムの構成に関する主な破損を *fpck* が検出すると、エラー・メッセージが表示されます。深刻ではないファイルシステムの問題については警告メッセージが表示されます。

メモ：外部システムを修復するには、その外部のオペレーティング・システムの修復ツールを使用する方が効果的です。

---

fpck ユーティリティーの使用方法については、fpck(1M) マン・ページを参照してください。外部ファイルシステムの種類についての詳細は filesystems(4)、外部ファイルシステムの作成ついては mkfp(1M) マン・ページを参照してください。

XFS ファイルシステムの問題の修復

xfs_repair コマンドは、XFS ファイルシステムの整合性をチェックし、検出された問題によってはその問題を解決します。ここでは、*xfs_repair* で発生するメッセージと、*xfs_repair* がファイルシステムを修復できない場合の対処方法について説明します。

共通エラー・メッセージ

xfs_repair と、*xfs_repair* が実行する修復処理から出力される共通エラー・メッセージは、次のとおりです。

`disconnected inode 242002, moving to lost+found`

xfs_repair は、使用中の i ノードを検出しましたが、これがファイルシステムに接続されていません。i ノードはファイルシステムの *lost+found* ディレクトリに移動されます。その名前は、i ノード番号になります。この例の場合

は、242002 です。切断された i ノードがディレクトリの場合は、ディレクトリのサブツリーが保持されます。すべての子ノードが自動的に一緒に移動され、ディレクトリのサブツリー全体が *lost+found* に移動します。

`imap claims in-use inode 2444941 is free, correcting imap`
ファイルシステム内の i ノード割当てマップは、i ノード 2444941 が空いていると認識していますが、i ノード自体は依然として使用中です。*xfs_repair* は i ノード・マップを修正して、i ノードが使用中であることを指定します。

`entry references free inode 2444940 in shortform directory 2444922 junking entry "fb" in directory inode 2444922`
ディレクトリ・エントリは、*xfs_repair* が実際に空き状態であると判断した i ノードを指しています。*xfs_repair* はディレクトリ・エントリを破棄します。*shortform* は、小さなディレクトリを表します。大きなディレクトリの場合、エントリの削除は、通常、2 パス・プロセスです。この場合、2 番目のメッセージは、`marking bad entry`（不良エントリの指定）、`marking entry to be deleted`（削除するエントリの指定）、または `will clear entry`（エントリの消去）というようなメッセージが表示されます。

`resetting inode 241996 nlinks from 5 to 3`
xfs_repair は、i ノード（リンク）を指すディレクトリ・エントリ数と、i ノードに記録されているリンク数が一致しないことを検出しました。その値が修正されました（この例の場合、5 から 3 に修正）。

`cleared inode 2444926`
i ノードに異常が検出されましたが修正できません。このため、*xfs_repair* はこれを長さが 0 の空き i ノードに変換しました。これは通常、i ノードが他で使用されているブロックを要求したり、i ノード自体の破損が原因です。通常、`cleared inode` メッセージの前に、i ノードの消去が必要となる原因を示す 1 つまたは複数のメッセージが表示されます。

ファイルが lost+found にある場合のエラー・メッセージ

xfs_repair がファイルシステムの *lost+found* ディレクトリにファイルとディレクトリを置いた場合、これを自分で削除しないと、次に *xfs_repair* を実行したときに、そのファイルとディレクトリの i ノードが一時的に切断されます。これらは、*xfs_repair* が終了する前に再接続されます。*lost+found* 内の i ノードが切断すると、以下のようなメッセージが表示されます。

```
Phase 1 - find and verify superblock...
Phase 2 - zero log...
        - scan filesystem freespace and inode maps...
        - found root inode chunk
Phase 3 - for each AG...
```

```
        - scan and clear agi unlinked lists...
        - process known inodes and perform inode discovery...
        - agno = 0
        - agno = 1
        ...
        - process newly discovered inodes...
Phase 4 - check for duplicate blocks...
        - setting up duplicate extent list...
        - clear lost+found (if it exists) ...
        - clearing existing "lost+found" inode
        - deleting existing "lost+found" entry
        - check for inodes claiming duplicate blocks...
        - agno = 0
imap claims in-use inode 242000 is free, correcting imap
        - agno = 1
        - agno = 2
        ...
Phase 5 - rebuild AG headers and trees...
        - reset superblock counters...
Phase 6 - check inode connectivity...
        - ensuring existence of lost+found directory
        - traversing filesystem starting at / ...
        - traversal finished ...
        - traversing all unattached subtrees ...
        - traversals finished ...
        - moving disconnected inodes to lost+found ...
disconnected inode 242000, moving to lost+found
Phase 7 - verify and correct link counts...
done
```

この例の場合、i ノード 242000 が、以前に実行された *xfs_repair* によって *lost+found* に移動された i ノードです。今回の *xfs_repair* の実行によって、ファイルシステムに矛盾がないことが検出されました。*lost+found* ディレクトリが空の場合、フェーズ 4 では *lost+found* ディレクトリの消去と削除についてのメッセージだけが表示されます。*lost+found* ディレクトリに i ノードがある場合、左揃えで `imap claims` メッセージと `disconnected inode` メッセージが表示されます。1 つの i ノードに対して 1 対のメッセージが表示されます。

xfs_repair がファイルシステムを修復できない場合の対処方法

xfs_repair がファイルシステムを正しく修復できなかった場合、再び *xfs_repair* コマンドを 2 回以上実行してみてください。*xfs_repair* を繰返し実行することで、さらに多くの修復を行うことができる場合があります。*xfs_repair* を 3 回実行しても整合性の問題を修復できない場合、失敗した位置によって対処の方法が異なります。

- *xfs_repair* がフェーズ 1 で失敗する場合は、失われたファイルをバックアップから復元する必要があります。
- *xfs_repair* がフェーズ 2 以降で失敗する場合は、バックアップを作成し、ファイルシステムにファイルを復元することによって、ディスクからファイルを復元できるかもしれません。

xfs_repair がフェーズ 2 以降で失敗した場合は、次の手順に従ってください。

1. ファイルシステムを *mount -r*（読取り専用）でマウントします。
2. *xfsdump* でファイルシステム・バックアップを作成します。
3. *mkfs* を使用して同じディスク・パーティションまたは XLV 論理ボリュームに新しいファイルシステムを作成します。
4. *xfsrestore* でバックアップからファイルを復元します。

xfsdump と *xfsrestore* については、『IRIX Admin: Backup, Security, and Accounting』を参照してください。

ログ・リカバリを実行しないファイルシステムのマウント

ファイルシステムが破損し、通常の方法ではファイルシステムのマウントが成功しないとき、*mount* コマンドの **-0 novecover** オプションを使用してファイルシステムをマウントすると、データの一部を回復できる場合があります。このオプションを使用すると、ログ・リカバリを実行せずにファイルシステムをマウントできます。このオプションを使用する際は、読込み専用でファイルシステムをマウントしてください。

ファイルシステムが正しくアンマウントされなかったために、ファイルシステムをノン・リカバリ・モードでマウントした場合、ファイルシステムに矛盾が発生し、そのファイルシステム内のすべてのデータを読込めなくなる可能性があります。ただし、ある状況下では、ほかの方法でアクセスできないデータを回復できる場合もあります。

マウント用のコマンドとコマンド・オプションについては、mount(1M) および fstab(4) マン・ページを参照してください。

root ファイルシステムに対する xfs_repair の実行

root ファイルシステムが破損した場合、root ファイルシステムに対して *xfs_repair* を実行することができます。その場合、次の手順に従ってミニルートから *xfs_repair* を実行します。

1. ミニルートを起動します。ミニルートのインストール手順については『IRIX Admin: Software Installation and Licensing』を参照してください。
2. ミニルートの「メイン・メニュー (Main Menu)」から「管理コマンド (Administrative Commands)」メニューを選択します。
3. *sh* を選択してシングル・ユーザ・シェルを表示します。
4. root ファイルシステムに対して *xfs_repair* を実行します。root ファイルシステムはほとんどの場合 */dev/dsk/dks0d1s0* です。

帯域保証 I/O のシステム管理

帯域保証 I/O (GRIO: Guaranteed-Rate I/O) は、ユーザ・アプリケーションがシステムの I/O リソースの一部を独占的に使用できるように予約するメカニズムです。たとえば、GRIO を使用して、リアルタイムでデータ・ストリームを検索および格納できます。GRIO は競合アプリケーション間でシステム・リソースを管理し、新しいプロセス処理が既存の処理パフォーマンスに影響を与えないようにします。GRIO は、XFS ファイルシステムのリアルタイム・サブボリュームにあるファイルに対してのみ読み書きが行えます。GRIO を使用するには、*eoe.sw.xfsrt* サブシステムをインストールする必要があります。

この章では、帯域保証 I/O に関する重要な概念、GRIO を使用するためのシステムの構成方法、GRIO を使用するアプリケーションで使用される XLV 論理ボリュームの作成手順を説明します。

この章では、次の項目について説明します。

詳細については grio(5) マン・ページを参照してください。

メモ：デフォルトでは、IRIX は 4 つの GRIO ストリーム（GRIO の並列使用）をサポートします。ストリーム数を 40 に増やすには、高性能の帯域保証 I/O（5 ～ 40 ストリーム・ソフトウェア・オプション）を購入してください。ストリーム数をさらに増やす場合は、高性能の帯域保証 I/O（ストリーム無制限のソフトウェア・オプション）を購入してください。

帯域保証 I/O の概要

帯域保証 I/O システム (GRIO) を使用すると、アプリケーションはファイルシステム間の特定の I/O 帯域幅を予約できます。アプリケーションは、ファイル記述子、データ・レート、継続時間、開始時間を提供して保証を要求します。ファイルシステムは実現可能なパフォーマンスを算出し、要求が認可されると、指定された時間内にパフォーマンスの要求レベルを満たすことを保証します。この結果、プログラマはシステム I/O パフォーマンスを予測する必要がなくなります。また、GRIO は、ビデオ・オン・デマンドなどのメディア・デリバリ・システムに適しています。

GRIO は、少ない I/O リソースに多数の異なるプロセスが同時にアクセスする環境用に設計されています。GRIO を使用すると、リソースがすでにフル稼働しており、これ以上の使用はパフォーマンスを低下させるということをアプリケーションによって判断できます。

すべてのシステム・リソースにアクセスする単一のアプリケーションを実行する場合は、GRIO のメカニズムを使用する必要はありません。競合するアプリケーションがないので、アクセスする前にリソースを予約してもアプリケーションは何も取得しません。

アプリケーションは、GRIO 予約をするためにシステムと取決めを行います。取決めの内容は、システムから一定期間にシステム・リソースの帯域幅の一部を提供するというものです。GRIO によってサポートされるシステム・リソースは、XFS ファイルシステムのリアルタイム・サブボリュームに常駐しています。要求によって指定されたプロセスおよびファイルに予約を転送できます。

GRIO 予約はデータ・レートをファイルシステムと関連付けます。データ・レートは、一定期間あたりのバイト数（タイム・カンタム）で定義されます。アプリケーションは、特定の時刻にファイルシステムとのデータの送受信を開始し、一定期間だけ継続します。たとえば、*/dev/xlv/video1* にあるファイルシステムに対して、1.29 秒あたり 1.2 MB のデータの送受信を 3 時間継続するように予約できます。この例では、1.29 秒が予約のタイム・カンタムです。

アプリケーションがシステムに対して予約を要求すると、システムはこの要求を受理または拒否します。要求が受理されると、アプリケーションはその予約を特定のファイルと関連付けます。予約の時刻になると、ファイルへのアクセスが開始され、予約期間中はタイム・カンタムごとに予約したバイト数を受信します。システムが予約を拒否した場合は、指定された時間にリソースに予約できる最大の帯域幅を返します。アプリケーションは、その使用可能な帯域幅が要求を満たすかどうかを判断し、帯域幅を下げて別の予約を要求したり、別の時間帯に予約することができます。

GRIO 予約は、期限切れになるまで、または明示的に **grio_unreserve_bw()** ライブラリ・コールが実行されるまで継続します。詳細については、grio_unreserve_bw(3X) マン・ページを参照してください。GRIO 予約は、予約と関連付けられているファイルを最終的に閉じたときにも削除されます。

プロセスがファイルに対して帯域保証を取得している場合は、別のファイル記述子が使用されていてもプロセスによるファイルの参照には帯域保証を使用します。ただし、同じファイルにアクセスするほかのプロセスは、保証なしに参照を行うか、または独自の保証を取得する必要があります。2 番目のプロセスが、保証を取得しているプロセスからファイル記述子を引継いだときも必要になります。

親子関係のあるプロセス・グループ内のプロセス間でのファイル記述子の共有は、GRIO のファイルでもサポートされています。またプロセスは保証も共有します。たとえば、あるファイルに対する帯域保証が 2Mb/s のプロセスがフォークされ、親と子が同じファイルにアクセスした場合、親と子は、合わせて 2Mb/s の帯域保証を受取ります。子に 4Mb/s の帯域保証が必要な場合は、ファイルを閉じ、再び開いて新たに 4Mb/s を取得する必要があります。

次に、4 種類の重要な GRIO のサイズを示します。

最適 I/O サイズ
: 最適 I/O サイズは、システムが実際にディスクに対して発行する I/O 動作のサイズです。XLV ボリュームのリアルタイム・サブボリュームにあるすべてのディスクが同じ最適 I/O サイズを持つ必要があります。異なる XLV ボリュームのリアルタイム・サブボリュームにあるディスクの最適 I/O サイズは異なる場合があります。詳細については、192 ページの「/etc/grio_disks ファイルのフォーマット」を参照してください。

XLV ボリュームのストライプ・ユニットのサイズ
: XLV ボリュームのストライプ・ユニットのサイズは、ストライプ内の単一ディスクに書込まれるデータ量です。XLV ボリュームのストライプ・ユニットのサイズは、そのサブボリュームにあるディスクに対する最適 I/O サイズの偶数倍である必要があります。詳細については、第 3 章の「XLV 論理ボリュームについて」を参照してください。

予約サイズ（レート）
: 予約サイズは、アプリケーションが 1 タイム・カンタムあたりに発行する I/O の合計です。

アプリケーション I/O サイズ
: アプリケーション I/O サイズは、アプリケーションが発行する個々の I/O 要求のサイズです。アプリケーション I/O サイズは、予約サイズと同じにすることをお薦めします。予約サイズはアプリケーション I/O サイズの偶数倍で、アプリケーション I/O サイズは、最適 I/O サイズの偶数倍である必要があります。

アプリケーションは、指定されたタイム・カンタム以内にすべての I/O 要求が発行されることを確認するので、システムはデータ・レートを保証できます。

GRIO 保証タイプ

予約サイズと継続時間を指定するほか、アプリケーションで任意の保証タイプを指定する必要があります。次に、帯域保証を取得するときに決定する必要がある 4 つの項目を示します。

- 帯域保証はファイル・ベース、またはファイルシステム・ベースで作成できます。
- 帯域保証は専用にすることも共有することもできます。
- 帯域保証タイプには、固定ロータ、スリップ・ロータ、またはノンロータがあります。
- 帯域保証では、デッドライン・スケジューリング、リアルタイムのスケジューリングが行えます。また、ノンスケジュールも可能です。

ユーザがオプションを指定しない場合、デフォルトの帯域保証のオプションには、共有オプション、ノンロータ・オプション、およびデッドライン・スケジューリングが使用されます。ファイルごとの保証、またはファイルシステムごとの保証は、その予約を作成するために **libgrio** コールによって決定されます。このコールは、**grio_reserve_file()** または **grio_reserve_file_system()** ライブラリ・コールのどちらかになります。

ファイルごとの保証とファイルシステムごとの保証

ファイルごとの保証では、特定のファイルにかぎり指定された帯域保証を使用できます。ファイルシステムごとの保証では、指定されたファイルシステムにある任意のファイルに保証を転送できます。

専用保証と共有保証

専用保証は、保証を取得したプロセスでのみ使用でき、別のプロセスに保証を転送できません。一方、共有保証はあるプロセスから別のプロセスへの転送が可能です。ただし、同時に 2 つのプロセスで共有保証を使用できません。

ロータ保証とノンロータ保証

ロータ保証（固定またはスリップ）は、VOD（ビデオ・オン・デマンド）とも呼ばれています。この保証では、ディスク・ドライブあたりでサポートされるストリームを追加できます。ただし、アプリケーションが I/O 要求をいつ、どこで発行するかの管理を必要とします。

ロータ保証は、ストライプ化されたリアルタイム・サブボリュームを使用する場合にかぎりサポートされます。アプリケーションがファイルにアクセスするとき、そのアクセスは、ストライプ内の各ドライブで分散合計された時間となります。アプリケーションは、1 タイム・カンタムの間に 1 つのディスクしかアクセスできないため、アクセスは、シーケンシャルであるとみなされます。したがって、アプリケーションがタイム・カンタムあたりのアクセスで必要とされるデータのキロバイト数でストライプ・ユニットを設定する必要があります。ボリューム・エレメントは作成される際に、*xlv_make* コマンドを使用して、ストライプ・ユニットを設定します。スリップ・ロータ保証を持つアプリケーションが異なるディスクのデータにアクセスを試みる場合は、目的のディスクにアクセスできるように、システムはプロセスのロータ・スロットの変更を試みます。アプリケーションが固定ロータ保証を持つ場合、指定されたディスクへアクセスするための適切なタイム・カンタムが確保できるまで、その保証は一時的に中止されます。

順次にファイルにアクセスしない、ファイル内をスキップする固定ロータを予約しているアプリケーションでは、パフォーマンスに影響があります。たとえば、リアルタイム・サブボリュームが 4 台に分割されたストライプに作成される場合、シークの完了後、最初の I/O 要求に対するタイムカンタムの 4 倍（ボリュームのストライプのサイズ）の時間がかかります。

ノンロータ保証には、上記のような制限はありません。通常、ノンロータ保証を持つアプリケーションは、ストライプ・サイズ・ユニット全体のファイルにアクセスします。ただし、帯域保証の範囲内にあるかぎり、ペナルティなしで、より小さなユニットまたはより大きなユニットにアクセスできます。ファイルを順次にアクセスする必要はありませんが、そのアクセスは、ストライプの境界上にある必要があります。アプリケーションが、許容されている保証より速くファイルへのアクセスを試みる場合、システムのアクションは、スケジューリング保証のタイプによって判断されます。

ロータ保証とノンロータ保証の比較例

システムに 8 つのディスクがあり、各ディスクが、1 秒あたり 23 個の 64 KB オペレーションをサポートしているものとします。コマンド *grio_bandwidth* を使用すると、特定のディスクで 1 秒あたりに実行できる指定サイズの I/O オペレーション数を調べることができます。ノンロータ GRIO では、アプリケーションが秒あたり 512 KB のデータを必要としている場合は、8 つのディスクは 8 台に分割されたストライプに整列されます。ストライプ・ユニットは、64 KB です。各アプリケーションの読込み / 書込み操作は 512 KB で、この結果、ストライプ内の各ディスクに対する読込み / 書込み操作が同時に行われます。アプリケーションは、ファイルの任意の部分へアクセス可能です。ただし、読込み / 書込み操作は必ずストライプの境界から開始します。この設定は、1 秒あたり 512 KB のデータを転送する 23 個のプロセス・ストリームを提供します。

ロータ保証の場合、8 つのドライブに 512 KB の最適 I/O サイズが割当てられます。各ドライブは、1 秒あたり読込み / 書込み操作を 7 つサポートできます。転送サイズを大きくするとシークが少なくなるので、転送速度を上げることができます (7 x 512 KB vs 23 x 64 KB)。再度、ドライ

ブは 8 台に分割されたストライプに配列されます。ただし、1 つのストライプ・ユニットは 512 KB です。各ドライブは、秒あたり 7 つの 512 K ストリームをサポートでき、合計で 8 * 7 = 56 ストリームになります。56 個の各ストリームには、期間が設定されています（時間のバケット）。期間には 8 種類あり、各期間には 7 種類のプロセスがあります。したがって、8 * 7 = 56 プロセスは、設定されたタイム・ユニットでデータにアクセスします。いずれかの期間におけるプロセスで、アクセスできるのは 1 つのディスクだけです。

ロータ保証を使用すると、同じディスク数でサポート可能なストリーム数を 2 倍以上にできます。ただし、その分時間の許容が厳しくなります。各ストリームでは、1 タイム・カンタム内に読込み / 書込み操作を発行する必要があります。プロセスが発行するコールが遅延し、リアルタイム・スケジューリングが使用されている場合は、ディスクのプロセスに対する次の期間まで、その要求は受理されません。この例では、最大 8 秒の遅延を意味しています。帯域保証を取得するには、アプリケーションがファイルに順次にアクセスする必要があります。この期間が下にあるストライプに移動して、各プロセスはファイルの次の 512 KB にアクセスできます。

リアルタイム・スケジューリング、デッドライン・スケジューリング、およびノンスケジュール予約

予約のスケジューリングには、リアルタイム・スケジューリング、デッドライン・スケジューリング、およびノンスケジュール予約の 3 種類があります。

リアルタイム・スケジューリングでは、アプリケーションが一定期間に一定量のデータを取得します。タイム・カンタム中はいつでもデータを返すことができます。この種類の予約は、小規模なバッファリングのみ行うアプリケーションで使用されます。アプリケーションが、その帯域保証以上のデータを要求した場合、システムは、帯域幅が保証されるまでアプリケーションを一時停止します。

デッドライン・スケジューリングでは、アプリケーションは、一定期間に最低限のデータ量を取得します。この保証は、大きなバッファ領域を持つアプリケーションで使用されます。アプリケーションは、少なくとも帯域保証と同じ転送速度で I/O を要求し、この転送速度がその帯域保証を超えたとき、使用可能な追加のデバイスの帯域幅がない場合は、一時停止します。

ノンスケジュール予約では、アプリケーションが取得する保証は、システムの帯域の予約のみです。システムでは予約制限を強制しないので、システムにある保証の I/O の転送速度は保証できません。ノンスケジュール予約を使用する場合は、十分注意してください。

GRIO システムのコンポーネント

GRIO 機構は、システム・デーモン、サポート・コマンド、設定ファイル、およびアプリケーション・ライブラリなど複数のコンポーネントから構成されています。

システム・デーモンは、*ggd* です。システム・デーモンは、システムの起動時にスクリプト */etc/rc2.d/S94grio* から起動します。このデーモンは必ず起動し、ほかのデーモンとは異なり、*chkconfig* コマンドでオン / オフの切替えができません。ロック・ファイルが */tmp* ディレクトリで作成されると、デーモンの 2 つのコピーが同時に実行されません。*ggd* デーモンに対して帯域保証の要求が出されます。デーモンは、GRIO 設定ファイル */etc/grio_disks* を読取ります。

/etc/grio_disks では、システムでサポートされているディスク・ドライブのタイプのパフォーマンス特性を示します。各ハードウェアで、各サイズ（64KB、128KB、256KB、512KB）の I/O 動作を 1 秒あたりに実行できる数も示しています。このファイルを編集して、追加のドライブ・タイプもサポートできるようにファイルを編集できます。コマンド *grio_bandwidth* を使用すると、特定のディスクで 1 秒あたりに実行できる指定サイズの I/O オペレーション数を調べることができます。このファイルのフォーマットについては、第 8 章の「/etc/grio_disks ファイルのフォーマット」で説明します。

コマンド *grio_bandwidth* を使用すると、特定のディスクで 1 秒あたりに実行できる指定サイズの I/O オペレーション数を調べることができます。

/usr/lib/libgrio.so ライブラリには、アプリケーションによる GRIO セッションの設定を可能にするルーチンの集合があります。このライブラリ・ルーチンは、アプリケーション・プログラムが *ggd* デーモンと通信を行うことができる唯一の方法です。また、このライブラリには、アプリケーションによって、ファイルシステムで使用可能なすべての帯域幅をチェックできるライブラリ・ルーチンもあります。このライブラリ・ルーチンによって、実際に要求を出すよりも速く特定の予約が認可されるかどうかを即座に取得できます。

GRIO 用のハードウェア構成の必要条件

帯域保証 I/O では、次の基準を満たすハードウェア構成が必要です。

- ログまたはデータ・サブボリュームのボリューム・エレメントではなく、リアルタイム・サブボリュームのボリューム・エレメントのみ、単一のディスクに配置します。この設定は、ソフト保証には推奨されており、ハード保証には必須です。
- リアルタイム・サブボリュームで作成する各 XLV ボリュームは、データ・サブボリュームを含む必要があります。データ・サブボリュームを使用する予定がない場合も同様です。デー

タ・サブボリュームは、XFS が i ノードおよびほかのファイルシステム内部情報を保存するときに使用されます。

- XLV 論理ボリュームのデータ・サブボリュームとログ・サブボリュームで使用されるディスクでは、再試行の機能をオンにする必要があります。データ・サブボリュームとログ・サブボリュームには、ファイルシステムの重要な情報が格納されていて、臨時のディスク・エラーを提供できません。

GRIO 用のシステム構成

注意 : ここでの手順は、正しく行わないとデータが消去されてしまうことがあります。したがって、この操作は IRIX システム管理者が行うことをお薦めします。

ここでは、GRIO 用にシステムを構成する方法について説明します。リアルタイム・サブボリュームを持つ XLV 論理ボリュームを作成し、ファイルシステムをボリュームに作成してマウントします。最後に、*ggd* デーモンを設定し、再起動します。

1. XLV 論理ボリューム用にディスク・パーティションを選択し、第 8 章の「GRIO 用のハードウェア構成の必要条件」で説明するとおり、ハードウェア構成を確認します。また、第 8 章の「ディスク・エラー回復機能のオフ」で説明するとおり、ディスク・ドライブ・パラメータを変更します。作成する各リアルタイム・サブボリュームに対して、データ・ディスク・パーティションとサブボリュームを作成するようにしてください。
2. XLV 論理ボリュームを構成する際に、使用する変数の値を決定します。

   vol_name リアルタイム・サブボリュームを持つボリューム名。

   rate このボリュームを使用するアプリケーションがデータをアクセスするときの転送速度。*rate* は、1 KB で分割されたストリーム（レート）ごとのタイム・カンタムあたりのバイト数です。この情報は、アプリケーションに関する出版物から、またはアプリケーションの開発者から取得し利用できます。

   num_disks ボリュームのリアルタイム・サブボリュームにあるディスク数。

   stripe_unit リアルタイム・ディスクがストライプ化されているときの（ビデオ・オン・デマンドには必須、それ以外には推奨）次の書込みを行う前のディスクに書込まれたデータ量。512 バイト・セクタで表されます。

   次に、ノンロータ保証の場合を示します。

   stripe_unit = *rate* * 1K / (*num_disks* * 512)

次に、ロータ保証の場合を示します。

stripe_unit = *rate* * 1K / 512

extent_size　ファイルシステムのエクステント・サイズ。

次に、ノンロータ保証の場合を示します。

extent_size = *rate* * 1K

次に、ロータ保証の場合を示します。

extent_size = *rate* * 1K * *num_disks*

opt_IO_size　最適 I/O サイズ。キロバイトで表します。デフォルトの *opt_IO_size* の値は、64（64 KB）、128（128 KB）、256（256 KB）、512（512 KB）の 4 つです。*/etc/grio_disks* ファイルを編集して、ほかの値を追加できます。詳細については、192 ページの「/etc/grio_disks ファイルのフォーマット」を参照してください。

ノンロータ保証の場合、*opt_IO_size* は *stripe_unit* の偶数の因数で、64 以上である必要があります。

ロータ保証の場合、*opt_IO_size* は、*rate* の偶数の因数である必要があります。*rate* と同じ *opt_IO_size* に設定することをお薦めします。

表 8-1 に、これらの変数値の例を示します。

表 8-1　GRIO 用の XLV 論理ボリュームの構成に使用される変数値の例

変数	保証タイプ	コメント	値の例
vol_name	任意	この名前は、ボリューム /dev/xlv/*vol_name* のデバイス名の最後のコンポーネントと一致します。	xlv_grio
rate	任意	この例では、ストリームあたり 1 秒 512 KB	512
num_disks	任意	この例では、4 ディスク	4
stripe_unit	ノンロータ	512 * 1 K/(4*512)	256
	ロータ	512 * 1 K/512	1024
extent_size	ノンロータ	512 * 1 K	512 KB
	ロータ	512 * 1 K * 4	2048 KB
opt_IO_size	ノンロータ	128/1 = 128 or 128/2 = 64 are possible	64
	ロータ	*rate* と同じ	512

3. XLV 論理ボリュームを作成するスクリプト・ファイル *xlv_make* を作成します。詳細については、第 4 章の「xlv_make によるボリューム・オブジェクトの作成」を参照してください。例 8-1 に、ボリュームのスクリプト・ファイルの例を示します。

例 8-1 GRIO に使用されるボリュームの設定ファイル

```
# Configuration file for logical volume vol_name. In this
# example, data and log subvolumes are partitions 0 and 1 of
# the disk at unit 1 of controller 1. The real-time
# subvolume is partition 0 of the disks at units 1-4 of
# controller 2.
#
vol vol_name
data
plex
ve dks1d1s0
log
plex
ve dks1d1s1
rt
plex
ve -stripe -stripe_unit stripe_unit dks2d1s0 dks2d2s0 dks2d3s0 dks2d4s0
show
end
exit
```

4. *xlv_make* を実行して、ボリュームを作成します。

```
# xlv_make script_file
```

script_file は、手順 3 で作成した *xlv_make* スクリプト・ファイルです。

5. 次のコマンドを実行して、ファイルシステムを作成します。

```
# mkfs -r extsize=extent_size /dev/xlv/vol_name
```

6. ファイルシステムを即座にマウントするには、次のコマンドを実行します。

```
# mkdir mountdir
# mount /dev/xlv/vol_name mountdir
```

mountdir はファイルシステムのマウント・ポイントであるディレクトリの完全なパス名です。

7. システムの起動時に新しいファイルシステムが自動的にマウントされるようにシステムを構成するには、次の行を */etc/fstab* に追加します。

```
/dev/xlv/vol_name mountdir xfs rw,raw=/dev/rxlv/vol_name 0 0
```

8. *ggd* デーモンを再起動します。

```
# /etc/init.d/grio stop
# /etc/init.d/grio start
```

これでユーザ・アプリケーションを開始できます。リアルタイム・サブボリュームのボリュームで作成されたファイルに、帯域保証 I/O を使用してアクセスできます。

GRIO 用の追加手順

次に、目的に応じてディスクおよび GRIO システム・コンポーネントを設定するための手順について説明します。この調整手順を実行すると、ディスク・ドライブから不良データが返される可能性があるのでお薦めできませんが、データの整合性よりデータのアクセス速度を重視したい場合には役立ちます。

ディスク・エラー回復機能のオフ

注意：ディスク・ドライブ・パラメータの設定は、必ず認可されたディスク・ドライブのタイプでのみ正しく行ってください。手順を間違えたり、認可されていないディスク・ドライブのタイプで行った場合は、ディスク・ドライブに致命的なダメージを与えることがあります。ディスク・ドライブ・パラメータの設定は、IRIX システム管理者が行ってください。

次に、ディスク・ドライブ・パラメータを設定する手順を示します。この例では、表 8-2 で示したすべてのパラメータを、コントローラ 131、ドライブ・アドレス 1 に変更します。

表 8-2　GRIO のディスク・ドライブ・パラメータ

パラメータ	新しい設定
自動不良ブロックの再配置（読取り）	オフ
自動不良ブロックの再配置（書込み）	オフ
エラー回復の遅延（このパラメータをオフにすると、継続読取り (RC) ビットがオンになります）	オフ

1. エキスパート・モードで、*fx* を起動します。

```
# fx -x
fx version 6.4, Sep 29, 1996
```

2. メッセージに応答して、変更するパラメータを持つディスクを指定します。

```
fx: "device-name" = (dksc) Enter
fx: ctlr# = (0) 131
fx: drive# = (1) 1
```

```
fx: lun# = (0)
...opening dksc(131,1,0)


...drive selftest...OK
```

3. 次に、ディスク・ドライブのディスク・ドライブ・タイプが、SGI 0664N1D または SGI 0664N1D 4I4I であることを確認します。これからのディスク・ドライブ・タイプは、ディスク・パラメータの変更のために認可されています。ディスク・ドライブ・タイプは出力の次の行に表示されます。表のリストにある認可されたタイプの 1 つであることを確認します。

```
Scsi drive type == SGI     0664N1D         6s61
----- please choose one (? for help, .. to quit this menu)-----
[exi]t               [d]ebug/             [l]abel/
[b]adblock/          [exe]rcise/          [r]epartition/
```

4. 次のコマンドを使用すると、ディスク・ドライブ・パラメータの現在の設定が表示されます。このコマンドでは、階層メニューにあるコマンドを階層ごとにスラッシュで区切って指定しています。

```
fx > label/show/parameters

----- current drive parameters-----
Error correction enabled          Enable data transfer on error
Don't report recovered errors     Do delay for error recovery
Don't transfer bad blocks         Error retry attempts          10
Do auto bad block reallocation (read)
Do auto bad block reallocation (write)
Drive readahead  enabled          Drive buffered writes disabled
Drive disable prefetch   65535    Drive minimum prefetch         0
Drive maximum prefetch   65535    Drive prefetch ceiling     65535
Number of cache segments     4
Read buffer ratio        0/256    Write buffer ratio         0/256
Command Tag Queueing disabled


----- please choose one (? for help, .. to quit this menu)-----
[exi]t               [d]ebug/             [l]abel/
[b]adblock/          [exe]rcise/          [r]epartition/
```

表 8-2 のパラメータは、上から 順に Do auto bad block reallocation (read) (自動不良ブロックの再配置（読取り）)、Do auto bad block reallocation (write) (自動不良ブロックの再配置（書込み）)、Do delay for error recovery (エラー回復の遅延) に対応しており、いずれも現在オンに設定されています。

5. 次のコマンドを使用して、ディスク・ドライブ・パラメータの設定を開始し、変更するパラメータが表示されるまで **<Enter>** キーを押します。

```
fx> label/set/parameters
fx/label/set/parameters: Error correction = (enabled) Enter
```

```
fx/label/set/parameters: Data transfer on error = (enabled) Enter
fx/label/set/parameters: Report recovered errors = (disabled) Enter
```

6. エラー回復のパラメータの遅延をオフに変更するには、メッセージに対して、"disable" と入力します。

```
fx/label/set/parameters: Delay for error recovery = (enabled) disable
```

7. 変更の必要がないパラメータに対しては、**<Enter>** キーを押します。

```
fx/label/set/parameters: Err retry count = (10) Enter
fx/label/set/parameters: Transfer of bad data blocks = (disabled) Enter
```

8. 自動不良ブロックを再配置するパラメータを変更するには、メッセージに対して "disable" と入力します。

```
fx/label/set/parameters: Auto bad block reallocation (write) = (enabled) disable
fx/label/set/parameters: Auto bad block reallocation (read) = (enabled) disable
```

9. 残りのパラメータに対して **<Enter>** キーを押します。

```
fx/label/set/parameters: Read ahead caching = (enabled) Enter
fx/label/set/parameters: Write buffering = (disabled) Enter
fx/label/set/parameters: Drive disable prefetch = (65535) Enter
fx/label/set/parameters: Drive minimum prefetch = (0) Enter
fx/label/set/parameters: Drive maximum prefetch = (65535) Enter
fx/label/set/parameters: Drive prefetch ceiling = (65535) Enter
fx/label/set/parameters: Number of cache segments = (4) Enter
fx/label/set/parameters: Enable CTQ = (disabled) Enter
fx/label/set/parameters: Read buffer ratio = (0/256) Enter
fx/label/set/parameters: Write buffer ratio = (0/256) Enter
```

10. ディスク・ドライブ・パラメータを変更する場合は、次のメッセージに対して "yes" と入力し、*fx* を終了します。

```
 * * * * * W A R N I N G * * * * *
about to modify drive parameters on disk dksc(131,1,0)! ok? yes

----- please choose one (? for help, .. to quit this menu)-----
[exi]t             [d]ebug/           [l]abel/           [a]uto
[b]adblock/        [exe]rcise/        [r]epartition/     [f]ormat
fx> exit
```

11. ディスク・ドライブ・パラメータを変更する場合は、次に示す確認のメッセージに対して **<Enter>** キーを押します。

```
label info has changed for disk dksc(131,1,0).  write out changes? (yes) <Enter>
```

ggd デーモンの再起動

/etc/grio_disks、または */etc/config/ggd.options* ファイルを編集した後、*ggd* を再起動して変更内容を有効にする必要があります。次のコマンドで *ggd* を再起動します。

```
# /etc/init.d/grio stop
# /etc/init.d/grio start
```

ggd を再起動すると、現在の帯域保証が無効になります。

リアルタイム・プロセスとしての *ggd* の実行

ggd をリアルタイム・プロセスとして実行すると、CPU を専有して GRIO 要求を独占的に実行できます。マルチプロセッサ・システムで次の手順に従い、リアルタイム・プロセスとして *ggd* を実行します。

1. */etc/config/ggd.options* ファイルを作成または編集して、ファイルに **-c** *cpunum* を追加します。*cpunum* は、GRIO のプロセッサ数です。この結果、CPU が GRIO 用に専有され、指定のプロセスだけを実行するように制限され、ノンプリエンプティブであると設定されます。GRIO を使用するプロセスは、リアルタイム・プロセス、および CPU *cpunum* でのみ実行可能なプロセスとして指定します。この方法については、sysmp(2) マン・ページを参照してください。
2. *ggd* デーモンを再起動します。手順については、189 ページの「ggd デーモンの再起動」を参照してください。
3. *ggd* を再起動した後、次のコマンドを実行して目的の CPU の設定を確認します。次の例では、*cpunum* は 3 です。

   ```
   # mpadmin -s
   processors: 0 1 2 3 4 5 6 7
   unrestricted: 0 1 2 5 6 7
   isolated: 3
   restricted: 3
   preemptive: 0 1 2 4 5 6 7
   clock: 0
   fast clock: 0
   ```

4. *ggd* を再起動した後、上記以外の CPU に対してリアル・タイムのプロセスを実行するように指定するには、次のコマンドを実行します。

   ```
   # mpadmin -rcpunum2
   # mpadmin -Icpunum2
   # mpadmin -Ccpunum2
   ```

リアルタイム・サブボリュームの使用

XLV 論理ボリュームのリアルタイム・サブボリュームに作成するファイルを、リアルタイム・ファイルと呼びます。ここでは、リアルタイム・ファイルの特殊機能について説明します。

リアルタイム・サブボリューム上のファイルとコマンド

リアルタイム・ファイルの特殊機能によって、標準の IRIX コマンドが通常とは異なる動作をします。その内容を次に示します。

- 標準のコマンドではリアルタイム・ファイルを作成できません。特別に記述されたプログラムだけが、リアルタイム・ファイルを作成できます。作成の方法は、第 8 章の「リアルタイム・サブボリューム上でのファイル作成」で説明されています。
- リアルタイム・ファイルは、ほかのファイルと同様に *ls* を使って表示します。ただし、*ls* 出力から、特定のファイルがデータ・サブボリューム上にあるか、またはリアルタイム・サブボリューム上のリアルタイム・ファイルであるかは判断できません。特別に記述されたプログラムだけが、ファイルの種類を判断できます。F_FSGETXATTR **fcntl()** システム・コールがそのプログラムで、ファイルがリアルタイム・ファイルなのか標準のデータ・ファイルなのかを判断できます。ファイルがリアルタイム・ファイルの場合、fsxattr 構造体の fsx_xflags フィールドの XFS_XFLAG_REALTIME ビットはオンです。
- *df* コマンドは、デフォルトで、データ・サブボリュームのディスク領域を表示します。**-r** オプションを指定すると、リアルタイム・サブボリュームのディスク領域と使用量が追加されます。*df* は、リアルタイム・サブボリュームがいっぱいのときに、ファイルシステム内にディスクの空き領域があることを報告し、*df -r* は、データ・サブボリュームがいっぱいのときに、ディスクの空き領域があることを報告します。

リアルタイム・サブボリューム上でのファイル作成

リアルタイム・ファイルを作成するには、fsxattr 構造体の fsx_xflags フィールドに XFS_XFLAG_REALTIME ビットを設定し、F_FSSETXATTR **fcntl()** システム・コールを使用します。これは、ファイルを新規作成または書込み用に開いて、データを書込む前に行う必要があります。一度ファイルにデータを書込むと、そのファイルを標準データ・ファイルからリアルタイム・ファイルに変更することも、リアルタイム・ファイルとして作成されたファイルを標準データ・ファイルに変更することもできません。

リアルタイム・ファイルは、ダイレクト I/O を使用した場合のみ読取りまたは書込みが可能です。このため、リアルタイム・ファイルに対する **read()** および **write()** システム・コール操作は、F_DIOINFO **fcntl()** システム・コールによって指定される条件を満たしている必要があります。**open()** システム・コールに対する O_DIRECT オプションの詳細については、open(2) マン・ページを参照してください。

GRIO ファイルのフォーマット

次では、*/etc/grio_disks*、および */etc/config/ggd.options* の 2 つの GRIO 設定ファイルについて説明します。

/etc/grio_disks ファイルのフォーマット

/etc/grio_disks ファイルには、システムで使用可能な各種ディスク・ドライブの I/O 帯域幅のパラメータに関する情報があります。

デフォルトでは、*/etc/grio_disks* には、64 KB、128 KB、256 KB、および 512 KB の最適 I/O サイズに対して、Silicon Graphics でサポートしているディスクのパラメータがあります。表 8-3 に、これらのディスクを列挙します。表 8-4 に、最適 I/O サイズと、表 8-3 に列挙されている各ディスクが 1 秒あたりに処理できる最適 I/O サイズ要求の数を示します。

表 8-3 デフォルトで */etc/grio_disks* にあるディスク

ディスク ID 文字列
"SGI IBM DFHSS2E 1111"
"SGI SEAGATE ST31200N8640"
"SGI SEAGATE ST31200N9278"
"SGI 066N1D 4I4I"
"SGI 0064N1D 4I4I"
"SGI 0664N1D 4I4I"
"SGI 0664N1D 6S61"
"SGI 0664N1D 6s61"
"SGI 0664N1H 6s61"
"IBM OEM 0663E15 eSfS"
"IMPRIMIS94601-15 1250"
"SEAGATE ST4767 2590"

表 8-4 最適 I/O サイズとサポートされる秒あたりの要求の数

最適 I/O サイズ	秒あたりの要求の数
65536	23
131072	16
262144	9
524288	5

ほかのディスクを追加、または別の最適 I/O サイズを指定するには、*/etc/grio_disks* ファイルに情報を追加する必要があります。*/etc/grio_disks* を変更した場合は、変更内容を有効にするために *ggd* デーモンを再起動する必要があります。189 ページの「ggd デーモンの再起動」を参照してください。

/etc/grio_disks のレコードには次の 2 つの形式があります。

```
ADD "disk id string" optimal_iosize number_optio_per_second

REPLACE devicename optal_iosize number_optio_per_second
```

最初のフィールドがキーワード ADD である場合、次のフィールドは、ドライブの製造元のディスク ID 文字列を示す 28 文字の文字列です。その次のフィールドはデバイスの最適 I/O サイズをバイトで示す整数値です。最後のフィールドは、ディスクが 1 秒間に満たすことが可能な最適 I/O 要求の数を示す整数値です。

次に、レコードの例を示します。

```
ADD     “SGI     SEAGATE ST31200N9278”  64K     23

ADD     “SGI             0064N1D 4I4I”  50K     25
```

最初のフィールドがキーワード *REPLACE* の場合は、次のフィールドはディスク・デバイスのパス名になります。パス名の詳細については、grio(1M) マン・ページを参照してください。3 番目のフィールドは、そのデバイスで使用される最適 I/O サイズを示す整数と、そのサイズで 1 秒間に実行できる I/O 操作の数です。

次に、*REPLACE* レコードの例を示します。

```
REPLACE /dev/rdsk/dks136d1s0 50K 20
```

/etc/config/ggd.options ファイルのフォーマット

/etc/config/ggd.options には、*ggd* デーモンのコマンド行オプションがあります。次に、このファイルに記述できるオプションを示します。

-c *cpunum* GRIO 要求を独占的に実施するために CPU *cpunum* を専有します。

-o *iosize* すべてのデバイスに、デフォルトの最適 I/O サイズを指定します。(64、128、256、512 など)。

このファイルを変更した場合は、*ggd* を再起動して、変更内容を有効にする必要があります。詳細については、189 ページの「ggd デーモンの再起動」を参照してください。

EFS ファイルシステム

メモ：今後の IRIX リリースでは、EFS ファイルシステムのサポートは予定されていません。EFS ファイルシステムを XFS ファイルシステムに変換する方法については、第 6 章「ファイルシステムの作成と拡張」を参照してください。

EFS ファイルシステムは、IRIX のオリジナル・ファイルシステムです。付録 A では、EFS ファイルシステムについて説明し、EFS ファイルシステムでさまざまな管理タスクを行うための情報を提供します。

付録 A では、次の項目について説明します。

EFS ファイルシステムの概要

EFS ファイルシステムは、IRIX のオリジナル・ファイルシステムです。このファイルシステムには、標準の UNIX ファイルシステムを拡張したエクステント *(extents)* があり（この後の説明を参照）、Extent File System (EFS) と呼ばれています。EFS ファイルシステムの最大サイズは、約 8 GB です。EFS は、512 バイトのファイルシステム・ブロックを使用し、ファイルの最大許容量は、2 GB から 1 バイト引いたサイズになります。

EFS の高度な機能は、EFS が単一の i ノード・テーブルではなく、データ・ブロックに非常に近い複数の i ノード・テーブルを保持すること、およびフリー・ブロックのリストの代わりにフリー・ブロックのトラックを保持するためのビット・マップを使用していることです。

i ノードが作成されるのは、ファイルの作成時ではなく EFS の作成時です。ファイルが作成されると i ノードが割当てられます。したがって、ファイルシステムで作成できる最大ファイル数は、ファイルシステムの i ノード数によって制限されます。デフォルトでは、作成される i ノード数は、パーティションのサイズまたは論理ボリュームのサイズの関数で決まります。通常、パーティションまたは論理ボリュームで 4 KB ごとに 1 つの i ノードが作成されます。i ノード数を指定するにはファイルシステムを作成する *mkfs* コマンドに **-n** オプションを指定します。i ノードはディスク領域を使用するので、i ノード数が増えるとファイルに使用できるディスク領域が減ります。

EFS の最初のブロックは使用されません。ファイルシステムに関する情報は、スーパーブロックと呼ばれるファイルシステムの 2 番目のブロック（ブロック 1）に格納されています。スーパーブロックに含まれる情報を次に示します。

- 物理ブロックと論理ブロックでのファイルシステムのサイズ。
- 読取り専用フラグ。このフラグが設定されている場合、ファイルシステムは読取り専用です。
- スーパーブロック修正済みフラグ。このフラグが設定されている場合、スーパーブロックは修正済みです。
- 最終更新日と最終更新時刻。
- インデックス・ノード（*i* ノード）の割当て総数。
- 空き i ノードの総数。
- 空きブロックの総数。
- 空きブロック・ビットマップの開始ブロック番号。

スーパーブロック・ビットマップの後には、i ノードとデータ・ブロックが続きます。ファイルを構成するデータ・ブロックの連続したグループはエクステントと呼ばれます。1 つの i ノードには、12 のエクステント・アドレスがあります。エクステントの長さは、1 ～ 148 の連続したブロックの範囲です。

i ノードには 12 のエクステント・アドレスがあり、合計で 1536 ブロック（786,432 バイト）を保持できます。ファイル容量が大きいために 12 のエクステントでは足りない場合、各エクステントは最大 148 のインダイレクト・エクステントのアドレスを使用してロードします。これで、インダイレクト・エクステントにはファイルを設定する実際のデータが含まれます。EFS は、インダイレクト・エクステントを使用するため、最大 2 GB までファイルを作成できます（ただしファイルシステムに十分なディスク領域がある場合）。

ファイルシステムの最後のブロックは、ファイルシステムのスーパーブロックのコピーです。これは、スーパーブロックに格納されている重要な情報のバックアップです。

EFS ファイルシステムの作成

ディスク・パーティションまたは論理ボリュームを EFS ファイルシステムに変換するには、*mkfs* コマンドを使用する必要があります。このコマンドは、ディスク・パーティションまたは論理ボリュームをデータ・ブロック、i ノード、および空きリストの各領域に分割し、該当する i ノード・テーブル、スーパーブロック、およびブロック・マップを書込みます。また、ファイルシステムのルート・ディレクトリを作成し、*lost+found* ディレクトリを作成します。

次に、EFS ファイルシステムを作成するための *mkfs* コマンドの例を示します。

```
# mkfs -t efs /dev/rdsk/dks0d2s7
```

mkfs を使用して EFS ファイルシステムを作成した後は、*fsck* コマンドを実行してディスクの整合性を確認します。*fsck* コマンドの詳細については、204 ページの「EFS ファイルシステムの問題の修復」を参照してください。

EFS ファイルシステムの作成手順の詳細については、197 ページの「EFS ファイルシステムの作成手順」、mkfs(1M) マン・ページ、mkfs_efs(1M) マン・ページを参照してください。

EFS ファイルシステムの作成手順

ここでは、ディスク・パーティションまたは論理ボリュームに EFS ファイルシステムを作成し、マウントする方法について説明します。論理ボリュームの作成については、第 4 章「XLV 論理ボリュームの作成と管理」を参照してください。この手順では、ディスクまたは論理ボリュームが空の状態であることを前提とします。この手順ではデータが破壊されるため、ディスクまたは論理ボリュームに重要なデータがある場合は必ずバックアップを作成してください。

アドバイス：「ツールチェスト (Toolchest)」の「システム (System)」->「システム・マネージャ (System Manager)」->「ハードウェアとデバイス (Hardware and Devices)」->「ディスク・マネージャ (Disk Manager)」を使用して、ディスク・パーティションに EFS ファイルシステムを作成できます。詳細については、『Personal System Administration Guide』の 第 3 章「ディスク・マネージャ」を参照してください。

注意：ファイルシステムの作成時にディスク・パーティションまたは論理ボリュームにあるファイルはすべて破壊されます。

1. ファイルシステムを作成するパーティションまたは論理ボリュームのデバイス名を確認します。これは、次の手順の例の *partition* の値です。たとえば、コントローラ 0、ドライブ・アドレス 2 にある SCSI オプション・ディスクのパーティション 7（ディスク全体）を使用する場合、*partition* は */dev/dsk/dks0d2s7* です。*partition* を確認する方法については、第 3 章の「XLV 論理ボリュームについて」、および dks(7M) マン・ページを参照してください。

2. ディスク・パーティションがすでにマウントされている場合は、アンマウントします。

   # **umount** *partition*

   ディスク・パーティションのデータはすべて破壊されます。データを破壊しないでデータの変換を行うには、第 6 章の「オプション・ディスク上のファイルシステム EFS から XFS への変換」の手順を実行してください。

3. *mkfs* コマンドを使用して、新しいファイルシステムを作成します。

   # **mkfs -t efs /dev/rdsk/dks0d2s7**

   mkfs の引数は、ディスク・パーティションまたは論理ボリュームのブロック・デバイスまたはキャラクタ・デバイスです。どちらかのデバイスを使用できます。

   上記の例では、*mkfs* はファイルシステムのパラメータにデフォルト値を使用します。デフォルト値以外のパラメータを使用する場合は、*mkfs* コマンド行で指定できます。コマンド行パラメータとプロトタイプ・ファイルの使用方法については、mkfs_efs(1M) マン・ページを参照してください。

4. ファイルシステムを使用するには、事前にマウントする必要があります。

   # **mkdir /rsrch**
   # **mount /dev/dsk/dks0d2s7 /rsrch**

   ファイルシステムのマウントについては、第 7 章の「手作業によるファイルシステムのマウント」を参照してください。

5. システムの起動時にこのファイルシステムが自動的にマウントされるようにシステムを設定するには、*/etc/fstab* ファイルに新しいファイルシステムのエントリを追加します。

   /dev/dsk/dks0d2s7 /rsrch efs rw,raw=/dev/rdsk/dks0d2s7 0 0

   ファイルシステムを自動的にマウントする方法については、第 7 章の「/etc/fstab ファイルによるファイルシステムのオートマウント」を参照してください。

別のディスクへの EFS ファイルシステムの拡張

ここでは、EFS ファイルシステムを別のディスクに拡張する手順について説明します。

次の手順は、*/disk2* ディスク・パーティションおよび新しいディスクから作成された XLV 論理ボリュームの */disk2* にマウントしたファイルシステムの EFS を拡張する方法を示します。この手順では、すでに新しいディスクがシステムにインストールされており、パーティションに区分されていることを前提とします。

注意：追加するディスクにあるすべてのファイルは、この手順の途中で破壊されます。

1. 拡張するファイルシステムのバックアップを作成します。
2. */disk2* ファイルシステムをアンマウントします。

   ```
   # umount /disk2
   ```

3. *xlv_make* を使用して */disk2* パーティションと新しいディスクから XLV 論理ボリュームを作成します。*/disk2* パーティションはデータ・サブボリュームの先頭のボリューム要素である必要があります。

   ```
   # xlv_make
   xlv_make> vol xlv0
   xlv0
   xlv_make> data
   xlv0.data
   xlv_make> plex
   xlv0.data.0
   xlv_make> ve dks0d2s7
   xlv0.data.0.0
   xlv_make> ve dks0d3s7
   xlv0.data.0.1
   xlv_make> end
   Object specification completed
   xlv_make> exit
   Newly created objects will be written to disk.
   Is this what you want?(yes) yes
   Invoking xlv_assemble
   ```

4. growfs コマンドを使用して論理ボリュームに EFS ファイルシステムを拡張します。

   ```
   # growfs /dev/xlv/xlv0
   ```

5. 拡張したファイルシステムで *fsck* を実行します。

   ```
   # fsck /dev/xlv/xlv0
   ```

6. 論理ボリュームをマウントします。

```
# mount /dev/xlv/xlv0 /disk2
```

7. ディスク・パーティションではなく論理ボリュームをマウントするため、ファイル */etc/fstab* の */disk2* のエントリを変更します。

```
/dev/xlv/xlv0 /disk2 efs rw,raw=/dev/rxlv/xlv0 0 0
```

EFS ファイルシステムのチェック

fsck コマンドは、EFS ファイルシステムの整合性とデータの完全性をチェックします。通常、ファイルシステムはシステムの起動時に自動的にチェックされます。root ファイルシステム以外のファイルシステムはチェック中にアンマウントされる必要があります。次の場合は *fsck* を手作業で実行することができます。

- バックアップの作成前
- リストアの実行後
- ディスク・メンテナンスの実行後
- ソフトウェアのインストール前
- ファイルシステムを手作業でマウントする前
- *fsck* を自動的に実行し、多くのエラーが発生したとき

fsck を実行する手順と、問題が見つかった際の処理方法については、204 ページの「EFS ファイルシステムの問題の修復」を参照してください。

root ファイルシステム以外の EFS ファイルシステムの整合性をチェックする場合は、まずそのファイルシステムをアンマウントしてください。root ファイルシステムは、マウントされていてもチェックできます。アンマウントするには、そのファイルシステムを明示的にアンマウントするか、またはシステムを一度停止してから、再度シングル・ユーザ・モードで起動します。ファイルシステムをアンマウントする方法については、第 7 章の「ファイルシステムのアンマウント」を参照してください。システムを一度停止して再度シングル・ユーザ・モードで起動する方法については、single(1M) マン・ページを参照してください。アンマウントされたファイルシステムのチェックについては、201 ページの「アンマウントされたファイルシステムのチェック」で説明されています。

システムを停止できないとき、およびファイルシステムをアンマウントできないときに緊急にファイルシステムのチェックを行う必要がある場合は、書込み不可モードで *fsck* を実行します。*fsck* コマンドはファイルシステムをチェックしますが、矛盾を検出しても変更や修正を行いません。この手順については、“ マウントされたファイルシステムのチェック ” in 付録 A で説明されています。

複数のファイルシステムを同時にチェックすると効率的です。これはパラレル・チェックと呼ばれています。パラレル・チェックには、*fsck* **-m** フラグを使用します。このオプションを含む *fsck* コマンドのオプションの詳細については、fsck(1M) マン・ページを参照してください。

アンマウントされたファイルシステムのチェック

単一のアンマウントされたファイルシステムをチェックするには、*root* として次のコマンドを実行します。

```
# fsck filesystem
```

filesystem は、ファイルシステムのディスク・パーティションまたは論理ボリュームのデバイス・ファイル名です。*/dev/usr*、*/dev/dsk/dsk0d2s7*、*/dev/dsk/lv2* などがその例です。詳細については、第 3 章の「XLV 論理ボリュームについて」と 第 5 章の「ファイルシステム名」を参照してください。

fsck は、一連の手順やフェーズを実行します。チェックを行ってもまったくエラーが検出されない場合があります。

```
fsck: Checking /dev/usr
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
7280 files 491832 blocks 38930 free
```

エラーが検出されない場合、ファイルシステムのチェックは完了です。

ファイルシステムでエラーが検出されると、*fsck* はエラー・メッセージを表示します。詳細については、“EFS ファイルシステムの問題の修復 ” in 付録 A で説明されています。

マウントされたファイルシステムのチェック

システムを停止できないとき、およびファイルシステムをアンマウントできないときに緊急にファイルシステムのチェックを行う必要がある場合は、書込み不可モードで *fsck* を実行します。*fsck* コマンドはファイルシステムをチェックしますが、矛盾が検出されても変更や修正を行いません。

たとえば、次のコマンドを入力すると、*fsck* が書込み不可モードで起動されます。

```
# fsck -n /dev/usr
```

矛盾があっても修復は行われません。エラーを修復する場合は、**-n** オプションを付けずに再度 *fsck* を実行する必要があります。この手順を使用すると、個々のファイルシステムにおけるエラーの重大性を判断できます。ただし、ファイルシステムがアクティブであるために、実際には存在しない矛盾が *fsck* によって検出されることがあります。

EFS ファイルシステムの再編成

時間が経過するにつれ、EFS ファイルシステムで断片化が起こります。断片化が発生すると、空き領域ブロックが少なくなり、ファイルが複数のエクステントにまたがります。*fsr* コマンドを EFS ファイルシステムで実行すると、ファイルシステムが再編成され、断片化したファイルのレイアウトが最適化され、空き領域がまとめられます。その結果、全体のパフォーマンスが向上します。

デフォルトでは、*fsr* は 1 週間に 1 度 *crontab* から自動的に実行されるように設定されています。マウントされたファイルシステムが EFS ファイルシステムであることを fsr コマンドが検出すると、*fsr_efs* コマンドが呼び出されます。*fsr* コマンドについては *fsr*(1M) リファレンス・ページを参照してください。*fsr_efs* オプションのコマンドについては、*fsr_efs*(1M) マン・ページを参照してください。

EFS ファイルシステムのディスク領域の管理

ディスク領域を管理する際、次の EFS ファイルシステムの特徴に留意してください。

- ディスク領域の不足に気付いた場合、まず EFS ファイルシステムの root の *lost+found* ディレクトリの容量が一杯でないかどうかを確認してください。root でログインするとこのディレクトリをチェックできます。また、ディレクトリ内のファイルを削除することもできます。
- EFS ファイルシステムでは、ファイルシステムの内容が約 90 ～ 95% 以上になるとシステムのパフォーマンスが低下することがあります。ただし、ディスクのサイズによってパフォーマンス低下の程度は異なります。たとえば、使用率が 97% の大きなディスクの空きディスク・ブロック数は、同じ使用率の小さなディスクの空きディスク・ブロック数よりも大きくなります。使用可能な空き領域をチェックし、常に十分なディスク領域を保つための必要な管理手順を行ってください。

EFS ファイルシステム上でのディスク割当ての使用

ディスク割当てを使用してユーザのディスク使用量を制限する方法については、第 5 章の「ディスクの割当て」で説明されています。ここでは、EFS ファイルシステム上でディスク割当てを実施および監視する方法について説明します。詳細については、quota(1)、edquota(1M)、quot(1M)、quotacheck(1M)、quotaon(1M)、repquota(1M) 、および quotas(4) マン・ページを参照してください。

EFS ファイルシステム上でのディスク割当ての実施

EFS ファイルシステム上でソフト・ディスク割当てを実施するには、次の手順に従ってください。

1. quotas サブシステムを有効にするには、次のコマンドを実行します。

   ```
   # chkconfig quotas on
   # chkconfig quotacheck on
   ```

2. ディスク割当てを行う各ファイルシステムの root ディレクトリに、*quotas* という名前のファイルを作成します。このファイルは長さがゼロで、*root* だけに書込みを許可するように設定します。*quotas* ファイルを作成するには、各ファイルシステムのルート・ディレクトリで *root* として次のコマンドを実行します。

   ```
   # touch quotas
   ```

3. 各ユーザに割当て量を設定します。*edquota* コマンドを使用して、各ユーザに対して制限を設定できます。たとえば、ユーザ ID sedgwick に対して 100 MB と 100 個の i ノードのソフト・リミットを設定するには、次のコマンドを実行します。

   ```
   # /usr/etc/edquota sedgwick
   ```

   画面が消去され、EDITOR 環境変数で指定されたエディタに切替わります （EDITOR が設定されていない場合は *vi*）。そのエディタでユーザのディスク割当てを編集します。次のように表示されます。

   ```
   fs /  kbytes(soft=0, hard=0)  inodes(soft=0, hard=0)
   ```

   ファイルシステムが最初に表示されます。この場合は、root ファイルシステム (*/*) が表示されます。ディスク領域は、メガバイト単位ではなくキロバイト単位で指定します。したがって、100 メガバイトを指定する場合は、100 に 1,024 を掛けてください。i ノード数は、そのまま入力します。

4. この行を次のように編集します。

   ```
   fs / kbytes(soft=102400, hard=0)  inodes(soft=100, hard=0)
   ```

5. 正しい数値を入力後ファイルを保存し、エディタを終了します。値を 0 のままにしておくと制限は設定されません。この例ではソフト・リミットだけが設定され、ハード・リミットの値は設定されていません。

6. *edquota* コマンドの **-p** オプションを使用して、複数のユーザに同じ数値を割当てます。割当て量が明示されていない場合、ユーザが使用できるディスク領域や作成できるファイル数に制限はありません。
7. 割当てを有効にするために、*quotaon* コマンドを実行します。正確に割当てるために、ファイルシステムのマウント直後にローカル・ファイルシステムでこのコマンドを実行してください。*quotaon* コマンドを指定すると、特定のファイルシステムに対して割当てを有効にできます。また、**-a** オプションを使用した場合は、*/etc/fstab* ファイルに示されているすべてのファイルシステムに対して割当てを有効にできます。*/etc/fstab* ファイルの詳細については、fstab(4) マン・ページを参照してください。

これ以降、システムの起動時にディスク割当てが自動的に実行されます。*/etc/init.d/quotas* スクリプトが割当ての有効化を行い、*chkconfig* コマンドを使用して **quotas** 設定フラグをチェックし、割当てを有効にするかどうかを判断します。割当てをオフにする場合は、*quotaoff* コマンドを使用します。

EFS ファイルシステム上でのディスク割当ての監視

割当てファイル内のレコードが、ユーザに割当てられている実際のブロック数とファイル数に一致していることを定期的にチェックしてください。このチェックには *quotacheck* コマンドを使用します。このコマンドを実行するために、ファイルシステムをアンマウントしたり、割当てシステムを無効にする必要はありません。ただし、アクティブなファイルシステムでは、結果が多少不正確になることがあります。

quotacheck フラグが *chkconfig* でオンに指定されている場合は、システムの起動時に *quotacheck* コマンドが */etc/init.d/quotas* スクリプトで自動的に実行されます。*quotacheck* コマンドは実行時間が長いため、起動時にこのコマンドを実行することをお薦めします。

EFS ファイルシステムの問題の修復

fsck コマンドは、EFS の整合性をチェックし、検出された問題によってはそれを解決します。ここでは、*fsck* の各フェーズで発生するメッセージ、メッセージの意味、およびその処理方法について説明します。

共通エラー・メッセージ

次の略語は、*fsck* エラー・メッセージで使用されます。

BLK	ブロック番号
DUP	重複ブロック番号
DIR	ディレクトリ名
MTIME	ファイルの最終変更日と最終変更時刻
UNREF	参照されない

以降の説明では、1 文字の省略文字を使用します。

B	ブロック番号
F	ファイル名（またはディレクトリ名）
I	i ノード番号
M	ファイル・モード
O	ファイルの所有者のユーザ ID
S	ファイル・サイズ
T	ファイルの最終変更日と最終変更時刻
X	リンク・カウント、BAD、DUP、または MISSING ブロックの数、あるいはファイル数（状況による）
Y	訂正されたリンク・カウント番号、またはファイルシステムのブロック数（状況による）
Z	空きブロック数

実際の *fsck* の出力では、これらの省略文字は該当する数字に置換えられます。

各フェーズで 2 種類のエラー・メッセージが表示されることがあります。このとき、ファイルシステムのチェックを継続するかどうかを尋ねる *fsck* のメッセージが続けて表示されますが、この場合のエラーは致命的なエラーであると考えてください。そして、コマンドの実行を中止し、問題の原因を調べてください。

`CAN NOT READ: BLK` *B* `(CONTINUE?)`
: ファイルシステムに指定されたブロック番号 *B* の読取りの要求が失敗しました。このエラーは重大な問題であることを示しています。ハードウェアの故障、または *fsck* がファイルシステムにないブロックを読取ろうとしている可能性があります。**n** を押して、*fsck* を中止します。システムを停止し、「System Maintenance」メニューに移行し、ディスク・ドライブおよびコントローラについてハードウェア診断テストを行います。

CAN NOT WRITE: BLK *B* (CONTINUE?)
: ファイルシステムに指定されたブロック番号 *B* の書込みの要求が失敗しました。ディスクが書込み禁止であるか、またはハードウェアに問題がある可能性があります。**n** を押して、*fsck* を中止します。ディスクが読取り専用に設定されていないことを確認します（ディスクによっては読取り専用になっている）。ディスクが書込み禁止でない場合はシステムを停止し、「 System Maintenance」メニューに移行し、ディスク・ドライブおよびコントローラについてハードウェア診断テストを行います。

初期化フェーズ

コマンド行の構文がチェックされます。ファイルシステムのチェックが実行される前に *fsck* によってテーブルが設定され、ファイルが開かれます。初期化エラーが発生すると、*fsck* は終了します。

フェーズ 1　ブロックおよびサイズのチェック

このフェーズでは、i ノードのリストをチェックします。次の操作を行ってエラーの状態を報告します。

- i ノード・タイプのチェック
- ゼロ・リンク・カウント・テーブルの設定
- 不良ブロックまたは重複ブロックに対する i ノード・ブロック番号のチェック
- i ノードのサイズのチェック
- i ノードの形式のチェック

フェーズ 1 のエラー・メッセージ

フェーズ 1 には、情報メッセージ、CONTINUE? プロンプトを伴うメッセージ、および CLEAR? プロンプトを伴うメッセージの 3 種類のエラー・メッセージがあります。フェーズ 1 のメッセージに対して入力する応答は、*fsck* 機能に影響を与えます。入力可能な応答については、" フェーズ 1 の応答 " in 付録 A で説明します。通常、特に指定がないかぎり Yes と応答します。

UNKNOWN FILE TYPE I=*I* (CLEAR?)
: i ノード *I* のモード・ワードは、i ノードが、パイプ、特殊文字の i ノード、標準の i ノード、ディレクトリ i ノード、シンボリック・リンク、またはソケットでないことを示しています。

LINK COUNT TABLE OVERFLOW (CONTINUE?)
: リンク・カウント 0 が割当てられた i ノードを含む *fsck* の内部テーブルに余分な空きがありません。

B BAD I=*I*
: i ノード *I* にあるブロック番号 *B* が、ファイルシステムの最初のデータ・ブロック番号より小さいか、またはファイルシステムの最後のデータ・ブロック番号より大きくなっています。このエラー状態は、ファイルシステムの範囲外のブロック番号が i ノード *I* に多数ある場合にフェーズ 1 で EXCESSIVE BAD BLKS エラー状態を引起こします。このエラー状態は、フェーズ 2 とフェーズ 4 においても、BAD/DUP エラー状態を引起こします。

EXCESSIVE BAD BLOCKS I=*I* (CONTINUE?)
: ファイルシステムの最初のデータ・ブロック番号より小さいか、または i ノード *I* と関連付けられたファイルシステムの最後のブロック番号より大きいブロック数が、許容値（通常は 50）を超えています。

B DUP I=*I*
: i ノード *I* にあるブロック番号 *B* が、別の i ノードにすでに含まれています。このエラー状態は、別の i ノードに含まれているブロック番号が多数 i ノード *I* にある場合に EXCESSIVE BAD BLKS エラー状態を引起こします。このエラー状態は、フェーズ 2 とフェーズ 4 においても、フェーズ 1B と BAD/DUP エラー状態を引起こします。通常、このエラーが初めて表示されたときは No と応答し、2 度目に表示されたときは、ほかの i ノードが含んでいるファイルを認識している場合は、Yes と応答します。

EXCESSIVE DUP BLKS I=*I* (CONTINUE?)
: ほかの i ノードで要求されているブロック数が、許容値（通常は 50）を超えています。

DUP TABLE OVERFLOW (CONTINUE?)
: 重複ブロック番号がある *fsck* の内部テーブルに余分な空きがありません。

PARTIALLY ALLOCATED INODE I=*I* (CLEAR?)
: i ノード *I* が、割当ても解除もされていません。

RIDICULOUS NUMBER OF EXTENTS (*n*) (max allowed *n*)
: エクステントの数が、システムに設定可能な最大数を上回っています。

ILLEGAL NUMBER OF INDIRECT EXTENTS (*n*)
: エクステントの数またはエクステントを指すポインタ（インダイレクト・エクステント）が、エクステントを記述するための i ノードのスロット数を超えています。

BAD MAGIC IN EXTENT
: エクステントを指すポインタには、マジック番号があります。この番号が無効な場合は、エクステント指すポインタが破損する可能性があります。

EXTENT OUT OF ORDER
: エクステントのファイル内の位置の情報が、ほかのエクステント・ポインタに関して、該当するエクステント・ポインタと矛盾しています。

ZERO LENGTH EXTENT
: エクステントの長さがゼロです。

ZERO SIZE DIRECTORY
: i ノードが、サイズが 0 のディレクトリを含んでいるのは誤りです。対応する i ノードが消去されます。

DIRECTORY SIZE ERROR
: ディレクトリのサイズは、整数のブロック数である必要があります。サイズは、そのエクステントに基づいて再度計算されます。

DIRECTORY EXTENTS CORRUPTED
: 上記のサイズの計算に失敗した場合は、*fsck* がこのメッセージを出力し、i ノードを消去するかどうかを尋ねてきます。

NUMBER OF EXTENTS TOO LARGE
: エクステントの数またはエクステントを指すポインタ（インダイレクト・エクステント）が、エクステントを記述するための i ノードのスロット数を超えています。

POSSIBLE DIRECTORY SIZE ERROR
: エクステントのポインタ長から算出したディレクトリのブロック数が、i ノードのサイズ・フィールドから算出した数と矛盾しています。

POSSIBLE FILE SIZE ERROR
: エクステントのポインタ長から算出したファイル内のブロック数が、i ノードのサイズ・フィールドから算出した数と矛盾しています。この場合、*fsck* で i ノードを消去するオプションがあります。

フェーズ 1 の応答

表 1-1 に、フェーズ 1 のプロンプトに対する応答の意味を示します。

表 1-1 *fsck* のフェーズ 1 の応答の意味

プロンプト	応答	意味
CONTINUE?	n	コマンドを終了します。
CONTINUE?	y	コマンドの実行を継続します。このエラー状態は、ファイルシステムの完全なチェックが可能ではないことを示します。*fsck* の 2 度目の実行は、ファイルシステムを再チェックします。

表 1-1 *fsck* のフェーズ 1 の応答の意味

プロンプト	応答	意味
CLEAR?	n	エラー状態を無視します。ユーザがほかの方法でその問題を修復する場合にかぎり No と応答します。
CLEAR?	y	i ノード *I* の値をゼロにして、i ノードの割当てを解除します。この結果、この i ノードを指す各ディレクトリのエントリに対して、フェーズ 2 で UNALLOCATED エラー状態を引起こす可能性があります。

フェーズ 1B Bad Dup のリスキャン

ファイルシステムで重複ブロックが検出された場合は、ファイルシステムがリスキャンされ、以前にそのブロックを含んでいた i ノードを検索します。重複ブロックが検出されると、次に示す情報メッセージが出力されます。

B DUP I=*I*　i ノード *I* にあるブロック番号 *B* は、すでに別の i ノードに含まれています。このエラー状態は、フェーズ 2 において BAD/DUP エラー状態を引起こします。重複したブロックを持つ i ノードは、このエラー状態と、フェーズ 1 の DUP エラー状態を調べて判断できます。

フェーズ 2　パス名のチェック

このフェーズでは、パス名ツリーをルート・ディレクトリから走査します。*fsck* は、チェックしているファイルシステムのディレクトリにあるファイルが使用している各 i ノードを調べます。

参照したファイルにはマークを付け、後で非参照ファイルを検出できるようにします。コマンドは、すべてのリンク・カウントを累計し、フェーズ 4 で検出されるリンク・カウントをチェックします。

フェーズ 2 では、次の事項が原因で発生するエラー状態を通知します。

- 不正なルート i ノード・モードおよび不正なステータス
- 範囲外のディレクトリ i ノード・ポインタ
- 不良 i ノードを指すディレクトリのエントリ

ルート・ディレクトリがすべてのパス名の検索を開始するので、*fsck* は、最初にルート・ディレクトリ i ノードをチェックします。

ルート・ディレクトリ i ノードが破損していたり、そのタイプがディレクトリでない場合は、*fsck* はエラー・メッセージを出力します。通常、ルート・ディレクトリに致命的な問題がある場合は、ファイルシステムを回復できません。*fsck* では、状況によっては処理を続けることができます。

フェーズ 2　エラー・メッセージ

次に、ルート・ディレクトリ i ノードでの問題が原因で出力されたメッセージを示します。それに対する応答については、“ フェーズ 2 の応答 ” in 付録 A で説明します。

`ROOT INODE UNALLOCATED. TERMINATING`
: ルート i ノードが不正な情報を示しています。この問題を修復する方法はないため、コマンドは停止します。

  root ファイルシステムでこの問題が発生した場合は、IRIX を再度インストールする必要があります。別のファイルシステムでこの問題が生じた場合は、*mkfs* コマンドを使用して、ファイルシステムを再度作成し、ファイルとデータをバックアップから回復する必要があります。

`ROOT INODE NOT A DIRECTORY. FIX?`
: ルート・ディレクトリ i ノードは、ディレクトリではないことを示しています。通常、このエラーは致命的です。通常、Yes と応答します。

`DUPS/BAD IN ROOT INODE. CONTINUE?`
: ルート・ディレクトリのブロック・アドレス情報に問題があります。通常、Yes と応答します。

そのほかのフェーズ 2 のメッセージでは、REMOVE? プロンプトが表示されます。次に、それらのメッセージを示します。

I `OUT OF RANGE I=`*I* `NAME=`*F* `(REMOVE?)`
: ディレクトリ・エントリ *F* に、i ノード・リストの最後の番号よりも大きい i ノード番号 *I* があります。通常、Yes と応答します。

`UNALLOCATED I=`*I* `OWNER=`*O* `MODE=`*M* `SIZE=`*S* `MTIME=`*T* `NAME=`*F*`(REMOVE?)`
: ディレクトリ・エントリ *F* に、割当てられていない i ノード *I* があります。所有者 *O*、モード *M*、サイズ *S*、変更時刻 *T*、ファイル名 *F* が出力されます。ファイルシステムがマウントされておらず、**-n** オプションが指定されていない場合、また、エントリが指す i ノードのサイズが 0 である場合、そのエントリは自動的に削除されます。

`DUP/BAD I=`*I* `OWNER=`*O* `MODE=`*M* `SIZE=`*S* `MTIME=`*T* `DIR=`*F* `(REMOVE?)`
: フェーズ 1 またはフェーズ 1B で、ディレクトリ・エントリ *F*、ディレクトリ i ノード *I* と関連付けられた重複ブロックまたは不良ブロックが検出されました。所有者 *O*、モード *M*、サイズ *S*、変更時刻 *T*、ディレクトリ名 *F* が出力さ

れます。通常、初めてこのエラーが表示された場合は、No と応答し、2 度目に表示された場合、ほかの i ノードが含んでいるファイルを認識している場合は、Yes と応答します。

`DUP/BAD I=`*I* `OWNER=`*O* `MODE=`*M* `SIZE=`*S* `MTIME=`*T* `FILE=`*F* `(REMOVE?)`

フェーズ 1 またはフェーズ 1B で、ファイル・エントリ *F*、i ノード *I* と関連付けられた重複ブロックまたは不良ブロックが検出されました。所有者 *O*、モード *M*、サイズ *S*、変更時刻 *T*、ファイル名 *F* が出力されます。通常、初めてこのエラーが表示された場合は、No と応答し、2 度目に表示された場合、ほかの i ノードが含んでいるファイルを認識している場合は、Yes と応答します。

フェーズ 2 の応答

表 1-2 に、フェーズ 2 のプロンプトに対する応答の意味を示します。

表 1-2 *fsck* のフェーズ 2 の応答の意味

プロンプト	応答	意味
FIX?	n	*fsck* を終了します。
FIX?	y	i ノード・モードがディレクトリを示していなくても、*fsck* は、ディレクトリとして i ノードの内容を扱います。ディレクトリに対して何も操作を行っていない場合、また、i ノード・モードのみが不正に設定されている場合は、ディレクトリを回復できます。
CONTINUE?	n	*fsck* を終了します。
CONTINUE?	y	*fsck* は、チェックを行って継続しようとします。ルート・ディレクトリがまだ読取り可能な場合は、ファイルシステムの一部を回復できます。
REMOVE?	n	エラー状態を無視します。ユーザがほかの方法でその問題を修復する場合にかぎり、No と応答します。
REMOVE?	y	不正なディレクトリのエントリを削除します。

フェーズ 3　接続性のチェック

fsck のフェーズ 3 は、フェーズ 2 で検出されたディレクトリで参照されなかったものを検索し、それらとの再接続を試みます。次の事項が原因で発生するエラー状態を通知します。

- 参照されないディレクトリ
- 存在しない、または一杯の *lost+found* ディレクトリ

フェーズ 3　エラー・メッセージ

フェーズ 3 では情報メッセージ、および RECONNECT? プロンプトを伴うメッセージの 2 種類のエラー・メッセージがあります。それに対する応答については、" フェーズ 3 の応答 " in 付録 A で説明します。

`UNREF DIR I=`*I* `OWNER=`*O* `MODE=`*M* `SIZE=`*S* `MTIME=`*T* `(RECONNECT?)`

ディレクトリ i ノード *I* は、ファイルシステムを走査したときにディレクトリのエントリに接続されていませんでした。ディレクトリ i ノード *I* の所有者 *O*、モード *M*、サイズ *S*、変更時刻 *T* が出力されます。*fsck* コマンドは、空でないディレクトリの再接続を強制します。通常、yes と応答します。

`SORRY. NO lost+found DIRECTORY`

lost+found ディレクトリが、ファイルシステムのルート・ディレクトリにありません。*fsck* は、*lost+found* のディレクトリにリンクする要求を無視します。参照されないファイルは削除します。

fsck **-l** を使用して、できるだけ早く *lost+found* ディレクトリを回復し、そのディレクトリを再度作成します。

`SORRY. NO SPACE IN lost+found DIRECTORY`

ファイルシステムのルート・ディレクトリにある *lost+found* ディレクトリに別のエントリを追加するための空き領域がありません。*fsck* は *lost+found* のディレクトリとリンクする要求を無視します。参照されないファイルは削除されます。

fsck **-l** を使用して、できるだけ早く *lost+found* ディレクトリを回復し、そのディレクトリ内を消去します。

`DIR I=`*I1* `CONNECTED. PARENT WAS I=`*I2*

ディレクトリ i ノード *I1* の *lost+found* ディレクトリへの接続が正常に行われたことを示すメッセージです。ディレクトリ i ノード *I1* の親 i ノード *I2* は、*lost+found* ディレクトリの i ノード番号に置換えられます。

フェーズ 3 の応答

表 1-3 に、フェーズ 3 のプロンプトに対する応答の意味を示します。

表 1-3 *fsck* のフェーズ 3 の応答の意味

プロンプト	応答	意味
RECONNECT?	n	エラー状態を無視します。これは、フェーズ 4 で UNREF エラー状態を引起こします。ユーザがほかの方法でその問題を修復する場合にかぎり、No と応答します。
RECONNECT?	y	ディレクトリ i ノード *I* を損失ファイル (*lost+found*) のディレクトリのファイルシステムに再接続します。ディレクトリ i ノード *I* を *lost+found* に接続したときに問題が生じた場合は、lost+found エラー状態を引起こす可能性があります。リンクが成功した場合は、CONNECTED 情報メッセージが表示されます。

フェーズ 4　リファレンス・カウントのチェック

このフェーズでは、フェーズ 2 および 3 で表示されるリンク・カウント情報をチェックし、参照されない標準のファイルを検索します。次の事項が原因で発生するエラー状態を通知します。

- 参照されないファイル
- 存在しない、または一杯の *lost+found* ディレクトリ
- ファイル、ディレクトリ、または特殊ファイルの不正なリンク・カウント
- 参照されないファイルとディレクトリ
- ファイルおよびディレクトリにある不良ブロックと重複ブロック
- フリー i ノードの不正な総カウント

フェーズ 4　エラー・メッセージ

フェーズ 4 には、次の 5 種類のエラー・メッセージがあります。

- 情報メッセージ
- RECONNECT? プロンプトを伴うメッセージ
- CLEAR? プロンプトを伴うメッセージ
- ADJUST? プロンプトを伴うメッセージ

- FIX? プロンプトを伴うメッセージ

これらのメッセージに対する応答については、“ フェーズ 4 の応答 ” in 付録 A で説明します。通常、特に指定がないかぎり Yes と応答します。

UNREF FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (RECONNECT?)
: i ノード *I* は、ファイルシステムを走査したときにディレクトリのエントリに接続されていませんでした。i ノード *I* の所有者 *O*、モード *M*、サイズ *S*、変更時刻 *T* が出力されます。**-n** オプションを省略し、ファイルシステムがマウントされていない場合は、空のファイルが自動的に消去されます。空でないファイルは消去されません。

SORRY. NO lost+found DIRECTORY
: ファイルシステムのルート・ディレクトリに *lost+found* ディレクトリがありません。*fsck* は、*lost+found* のファイルとリンクする要求を無視します。

  fsck **-l** を使用して、できるだけ早く *lost+found* ディレクトリを回復し、そのディレクトリを作成します。

SORRY. NO SPACE IN lost+found DIRECTORY
: ファイルシステムのルート・ディレクトリにある *lost+found* ディレクトリに別のエントリを追加するための空き領域がありません。*fsck* は、*lost+found* のファイルをリンクする要求を無視します。

  fsck **-l** を使用して、できるだけ早く *lost+found* ディレクトリを回復し、そのディレクトリ内を消去します。

(CLEAR)
: 直前の UNREF エラー状態に示された i ノードに再接続できません。したがって、i ノードは消去されます。

LINK COUNT FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* COUNT=*X* SHOULD BE *Y* (ADJUST?)
: i ノード *I* のリンク・カウントが *X* ではなく、*Y* である必要があります。所有者 *O*、モード *M*、サイズ *S*、変更時刻 *T* が出力されます。

LINK COUNT DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* COUNT=*X* SHOULD BE *Y* (ADJUST?)
: i ノード *I* のリンク・カウント（ディレクトリ）が *X* ではなく、*Y* である必要があります。ディレクトリ i ノード *I* の所有者 *O*、モード *M*、サイズ *S*、変更時刻 *T* が出力されます。

LINK COUNT *F* I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* COUNT=*X* SHOULD BE *Y* (ADJUST?)
: *F* i ノード *I* のリンク・カウントが *X* ではなく、*Y* である必要があります。ファイル名 *F*、所有者 *O*、モード *M*、サイズ *S*、変更時刻 *T* が出力されます。

UNREF FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR?)
: ファイル i ノード *I* はファイルシステムを走査したときにディレクトリのエントリに接続されていませんでした。i ノード *I* の所有者 *O*、モード *M*、サイズ *S*、変更時刻 *T* が出力されます。**-n** オプションを省略し、そのファイルシステムがマウントされていない場合は、空のファイルが自動的に消去されます。空でないディレクトリは消去されません。通常、初めてこのエラーが表示された場合は、no と応答し、2 度目に表示された場合、ほかの i ノードが含んでいるファイルを認識しているならば、yes と応答します。

UNREF DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR?)
: ディレクトリ i ノード *I* はファイルシステムを走査したときにディレクトリのエントリに接続されていませんでした。i ノード *I* の所有者 *O*、モード *M*、サイズ *S*、変更時刻 *T* が出力されます。**-n** オプションを省略し、そのファイルシステムがマウントされていない場合は、空のディレクトリが自動的に消去されます。空でないディレクトリは消去されません。通常、初めてこのエラーが表示された場合は、no と応答し、2 度目に表示された場合、ほかの i ノードが含んでいるファイルを認識しているならば、yes と応答します。

BAD/DUP FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR?)
: フェーズ 1 またはフェーズ 1B で、ファイル i ノード *I* と関連付けられている重複ブロックまたは不良ブロックが検出されました。i ノード *I* の所有者 *O*、モード *M*、サイズ *S*、変更時刻 *T* が出力されます。通常、初めてこのエラーが表示された場合は、no と応答し、2 度目に表示された場合、ほかの i ノードが含んでいるファイルを認識しているならば、yes と応答します。

BAD/DUP DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR?)
: フェーズ 1 またはフェーズ 1B で、ディレクトリ i ノード *I* と関連付けられている重複ブロックまたは不良ブロックが検出されました。i ノード *I* の所有者 *O*、モード *M*、サイズ *S*、変更時刻 *T* が出力されます。通常、初めてこのエラーが表示された場合は、no と応答し、2 度目に表示された場合、ほかの i ノードが含んでいるファイルを認識しているならば、yes と応答します。

FREE INODE COUNT WRONG IN SUPERBLK (FIX?)
: 空き i ノードの実際のカウントが、ファイルシステムのスーパーブロックのカウントと一致しません。

フェーズ 4 の応答

表 1-4 に、フェーズ 4 プロンプトに対する応答の意味を示します。

表 1-4 *fsck* のフェーズ 4 の応答の意味

プロンプト	応答	意味
RECONNECT?	n	このエラー状態を無視します。後でフェーズ 4 で CLEAR エラー状態を引起こします。
RECONNECT?	y	i ノード *I* を損失ファイル (*lost+found*) のディレクトリのファイルシステムに再度接続します。i ノード *I* を *lost+found* に接続したときに問題が生じた場合は、このフェーズで lost+found エラー状態を引起こす可能性があります。
CLEAR?	n	エラー状態を無視します。ユーザがほかの方法でその問題を修復する場合にかぎり、No と応答します。
CLEAR?	y	i ノードの内容をゼロにして、i ノードの割当てを解除します。
ADJUST?	n	エラー状態を無視します。ユーザがほかの方法でその問題を修復する場合にかぎり、No と応答します。
ADJUST?	y	ファイルの i ノード *I* のリンク・カウントをフェーズ 2 で計算されたリンク・カウントに置換えます。
FIX?	n	エラー状態を無視します。ユーザがほかの方法でその問題を修復する場合にかぎり、No と応答します。
FIX?	y	問題を修復します。

フェーズ 5　空きリストのチェック

フェーズ 5 では、空きブロック・リストをチェックします。次の事項が原因で発生するエラー状態を通知します。

- 空きブロック・リストの不良ブロック
- 不良な空きブロック・カウント
- 空きブロック・リストの重複ブロック
- 空きブロック・リストにないファイルシステムの未使用ブロック
- 空きブロックの不正な総カウント

フェーズ 5 エラー・メッセージ

フェーズ 5 には、次の 4 種類のエラー・メッセージがあります。

- 情報メッセージ
- CONTINUE? プロンプトを伴うメッセージ
- FIX? プロンプトを伴うメッセージ
- SALVAGE? プロンプトを伴うメッセージ

これらのメッセージに対する応答については、" フェーズ 5 の応答 " in 付録 A で説明します。通常、Yes と応答します。

`FREE BLK COUNT WRONG IN SUPERBLOCK (FIX?)`
空きブロックの実際のカウントが、ファイルシステムのスーパーブロックのカウントと一致しません。

`BAD FREE LIST (SALVAGE?)`
このメッセージは必ず、フェーズ 5 の情報メッセージの後に表示されます。

フェーズ 5 の応答

表 1-5 に、フェーズ 5 プロンプトに対する応答の意味を示します。

表 1-5 *fsck* のフェーズ 5 の応答の意味

プロンプト	応答	意味
CONTINUE?	n	コマンドを終了します。
CONTINUE?	y	残りの空きブロックのリストを無視し、*fsck* の実行を継続します。フェーズ 5 の後の方で、このエラー状態は必ず BAD BLKS IN FREE LIST をエラー状態を引起こします。
FIX?	n	エラー状態を無視します。ユーザがほかの方法でその問題を修復する場合にかぎり、No と応答します。
FIX?	y	スーパーブロックのカウントを実際のカウントに置換えます。
SALVAGE?	n	エラー状態を無視します。ユーザがほかの方法でその問題を修復する場合にかぎり、No と応答します。
SALVAGE?	y	実際の空きブロックのビットマップを新しい空きブロックのビットマップに置換えます。

フェーズ 6　空きリストの回復

このフェーズでは、空きブロックのビットマップを再構成します。このフェーズで生成されるエラー・メッセージはないので、応答する必要はありません。

クリーンアップ・フェーズ

ファイルシステムのチェックが終了すると、クリーンアップ機能が実行されます。クリーンアップ・フェーズでは、ファイルシステムやその状態に関する忠告メッセージが表示されます。

クリーンアップ・フェーズのメッセージ

X `files` *Y* `blocks` *Z* `free`
: チェックしたファイルシステムには、*X* 個のファイルがあり、*Y* 個のブロックが使用され、*Z* 個の空きブロックがあることを示します。

`SUPERBLOCK MARKED DIRTY`
: ファイルシステムをマウントする前に、*fsck* を実行する必要があるかどうかを判断するために、スーパーブロックのフィールドをシステム・コマンドが問い合わせます。このフィールドが “clean” でない場合は、*fsck* が消去するかどうか尋ねます。

`PRIMARY SUPERBLOCK WAS INVALID`
: プライマリ・スーパーブロックが破損して使用不能になり、*fsck* がセカンダリ・スーパーブロックを認識した場合は、プライマリ・スーパーブロックをバックアップと置換えるかを尋ねます。

`SECONDARY SUPERBLOCK MISSING`
: セカンダリ・スーパーブロックがなく、*fsck* がそのための領域（最後のシリンダ・グループの後）を認識した場合は、セカンダリ・スーパーブロックを作成するかどうかを尋ねます。

`CHECKSUM WRONG IN SUPERBLOCK`
: チェックサムが間違っているので、ファイルシステムがマウントできません。

`***** FILE SYSTEM WAS MODIFIED *****`
: 現在のファイルシステムが *fsck* によって変更されたことを示します。

`***** REMOUNTING ROOT... *****`
: マウントされている root ファイルシステムが *fsck* によって変更されたことを示します。自動再マウントが行われ、インコア・データ構造とファイルシステムの整合性を保持します。

索引

F

G

H

I

L

M

N

P

Q

R

S

U

V

X

あ

え

お

か

き

こ

さ

し

す

せ

そ

た

て

と

な

に

の

は

ひ

ふ

へ

ほ

ま

み

め

ゆ

よ

り

れ

ろ