PERFORCE 2007.2
システム管理者ガイド

May2007

目　次

はじめに

このマニュアルについて

本書は『PERFORCE 2007.2 システム管理者ガイド』です。

本書は、PERFORCE サーバのインストール、構成、および保守を担当するユーザを対象としています。本書が扱う項目には、本来の「システム管理者」の仕事（ソフトウェアのインストールおよび構成、稼働時間およびデータの一貫性の確保など）のほかに、PERFORCE ユーザの設定、PERFORCE ディポのアクセス制御の構成、PERFORCE ユーザのパスワードの再設定など、「PERFORCE 管理者」の仕事も含まれています。

PERFORCE はシステムの特別な許可を必要としませんので、PERFORCE 管理者は通常、root レベルのアクセスを必要としません。サイトの事情によっては、必ずしも PERFORCE 管理者がシステム管理者である必要はありません。

UNIX と Windows、どちらのバージョンの PERFORCE サーバも、コマンドライン・インターフェイスを介して管理します。PERFORCE のコマンドライン・クライアントについて詳しく知りたい場合は、『コマンド・リファレンス』をご覧ください。

PERFORCE の使用方法を知りたい場合は

PERFORCE サーバの管理のほかに PERFORCE の使用も計画しており、ユーザの観点からの PERFORCE に関する情報が必要な場合は、P4 ユーザーズ・ガイド』を参照してください。

関連文書はすべて当社の Web サイト、`http://www.perforce.com` より入手可能です。

ご意見・ご感想をお待ちしています

当社では、ユーザのみなさんからの本書の関するご意見をお待ちしています。とりわけ、初めて PERFORCE をお使いになったユーザのみなさんのご意見は、ぜひお聞かせ願いたいと思います。本書を読んで十分な情報は得られましたか？ ご感想をお聞かせください。ご意見・ご感想の宛先は右記のとおりです。`manual@perforce.com`

第 1 章

PERFORCE との出会い：インストールとアップグレード

本章では、PERFORCE サーバのインストール方法と、すでにインストールされている PERFORCE サーバのアップグレードの方法を説明します。

> *すでにインストールされている PERFORCE サーバをリリース 2000.2 以前から 2005.1 以降にアップグレードする場合は、事前に 17 ページの「サーバをアップグレードする」の注意事項をお読みください。*

本章では、インストール時に考慮すべきことを簡単に概説するとともに、セキュリティや管理に関する基本的な情報も紹介します。管理者の仕事に関するより詳しい情報は後の章に譲ります。

Windows | *UNIX と Windows で PERFORCE の動作が異なる場合には、その旨を記載します。Windows にのみ関わる情報については、121 ページの「PERFORCE と Windows」をご覧ください。*

*本書に紹介されている例の多くは*UNIX *版 PERFORCE サーバを前提にしていますが、ほとんどの場合、これらの例は Windows にも* UNIX *にも共通しています。*

OS X | UNIX *に関する説明はそのまま Mac OS X にも当てはまります。*

PERFORCE サーバの Unicode モードおよび日本語の入出力に関しては、本書とともに提供されている i18nnotes.txt をご参照ください。

PERFORCE を入手する

PERFORCE は少なくとも 2 つの実行ファイル、すなわちサーバ（`p4d`）と少なくとも 1 つの PERFORCE クライアント・プログラム（UNIX の場合は `p4`、Windows の場合は `p4.exe` または `p4v.exe`）を必要とします。

これらのサーバおよびクライアントの実行ファイルは、PERFORCE 社の Web サイトにある下記の Downloads ページから入手することができます。

```
http://www.perforce.com/perforce/loadprog.html
```

お使いのプラットフォームに応じたファイルを選び、ディスクに保存してください。

UNIX 環境下でのインストール

p4 および p4d はどのディレクトリにもインストールできますが、UNIX では通常、クライアント・プログラムは /usr/local/bin に置かれ、サーバ・プログラムは一般に /usr/local/bin または独自のルート・ディレクトリに置かれます。クライアント・プログラムは、p4d ホストに TCP/IP でアクセスできるあらゆるマシンにインストールできます。

PERFORCE サーバ・ファイルへのアクセスを制限するため、p4d 実行ファイルは PERFORCE サーバを実行する目的で生成された PERFORCE ユーザ・アカウントによって所有し、動作させるようにしてください。

PERFORCE の使用を開始するには

1. PERFORCE の Web サイトから p4 と p4d のファイルをダウンロードする。
2. ダウンロードした p4 および p4d ファイルを実行可能にする。
3. PERFORCE のデータベースおよびバージョン化ファイルを保持するサーバ・ルート・ディレクトリを作成する。
4. p4d に TCP/IP ポートを設定して、PERFORCE サーバにどのポートが接続待ちかを知らせる。
5. サーバ（p4d）を起動する。
6. P4PORT 環境変数を設定して、クライアント・プログラム用に PERFORCE サーバ・マシンの名前または TCP/IP アドレスと p4d ポート番号を指定する。

ダウンロードしたファイルを実行可能にする

UNIX （または Mac OS X）では、実行可能プログラム（p4、p4d）を実行可能にする必要があります。プログラムをダウンロードしたら、次のように chmod コマンドを使ってそれらを実行可能にしてください。

```
chmod +x p4
chmod +x p4d
```

PERFORCE サーバのルート・ディレクトリを作成する

PERFORCE サーバは、ユーザがサブミットしたすべてのファイルとシステムが生成したメタデータを独自のルート・ディレクトリのファイルおよびサブディレクトリに保存します。このディレクトリを*サーバ・ルート*といいます。

サーバ・ルートは環境変数 P4ROOT で設定するか、p4d 起動時に -r *root_dir* オプションで指定します。PERFORCE のクライアント・プログラムが P4ROOT のディレクトリまたは環境変数を使用することはありません。p4d が P4ROOT 変数を使用する唯一のプロセスです。

PERFORCE のファイルはすべてサーバ・ルートに保存されるので、サーバ・ルートの中身は時間とともにふくらみます。ディスク容量の管理については 20 ページの「インストールと管理に関する情報」で簡単に触れ、さらに詳しくは 110 ページの「ディスク容量の割り当て」で説明します。

PERFORCE サーバは特権的アクセスを必要としません。p4d を root またはその他の特権ユーザとして動作させる必要はありません。詳しくは、23 ページの「UNIX: P4D を非特権ユーザとして動作させる」をご覧ください。

サーバ・ルートはどこにでも置くことができますが、p4d を動作させるアカウントはサーバ・ルートとそのすべての下位ディレクトリに対して read、write、execute のパーミッションを持っていなければなりません。セキュリティ確保のため、p4d を動作させるアカウントのファイル作成モードマスク umask(1) は、他のユーザによるサーバ・ルート・ディレクトリへのアクセスを許可しない値に設定します。

PERFORCE サーバに接続待ちするポートを指定する

p4d サーバと PERFORCE クライアント・プログラムは TCP/IP で通信します。p4d は、起動すると、(デフォルトでは) ポート 1666 で接続待ちします。PERFORCE クライアントも (デフォルトでは) p4d サーバが perforce という名前のホスト上にあり、ポート 1666 で接続待ちしていると想定します。

p4d に別のポートで接続待ちさせる場合は、そのポートを p4d 起動時に -p *port_num* オプションで指定するか (例 : p4d -p 1818)、環境 (レジストリ) 変数 P4PORT で設定します。

P4ROOT とは異なり、環境変数 P4PORT は PERFORCE サーバと PERFORCE クライアント・プログラムの双方が使用するもので、PERFORCE サーバ・マシンと PERFORCE クライアント・ワークステーションの双方で設定しなければなりません。

PERFORCE クライアント・プログラムにサーバの接続ポートを指定する

PERFORCE クライアント・プログラムは、p4d サーバがどのマシン上にあり、どの TCP/IP ポートで接続待ちしているかを知っている必要があります。各ユーザの環境変数 P4PORT を *host*:*port* に設定します。ただし、*host* は p4d が動作しているマシンの名前、*port* は p4d が接続待ちしているポートの番号です。

例 :

	クライアント・プログラムの動作
dogs:3435	PERFORCE クライアント・プログラムはホスト dogs 上のポート 3435 で接続待ちしている p4d サーバに接続します。
x.com:1818	PERFORCE クライアント・プログラムはホスト x.com 上のポート 1818 で接続待ちしている p4d サーバに接続します。

PERFORCE クライアント・プログラムが p4d と同じホスト上で動作している場合には、P4PORT で p4d のポート番号だけを設定すれば十分です。p4d が perforce という名前またはエイリアスのホスト上で動作していて、ポート 1666 で接続待ちしている場合には、クライアント向けの P4PORT を未設定のままにします。

P4PORT の設定	クライアント・プログラムの動作
3435	PERFORCE クライアント・プログラムはクライアント・プログラムと同一のマシン上のポート 3435 で接続待ちしている p4d サーバに接続します。
<not set>	PERFORCE クライアント・プログラムは perforce という名前またはエイリアスのホストのポート 1666 で接続待ちしている p4d サーバに接続します。

p4d ホストの名前が perforce ではない場合には、ユーザのワークステーションの /etc/hosts ファイルで本来のホスト名に perforce をエイリアスとして設定するか、Sun の NIS または Internet DNS を使用して同様の操作を行うことにより、ユーザの手間をいくらか省くことができます。

サーバを起動する

環境変数 P4PORT と P4ROOT を設定したら、次のコマンドにより p4d をバックグラウンドで起動します。

```
p4d &
```

p4d を動作させるだけなら、このコマンドで十分ですが、ほかにエラー・ログ、チェックポイント、ジャーナルなどを制御するオプションを付けることができます。

例 : *PERFORCE サーバ起動コマンド*

p4d を起動するときに -p オプションを付けると P4PORT を、-r オプションを付けると P4ROOT をオーバーライドすることができます。同様に、-J オプションでジャーナル・ファイルを指定することができ、-L オプションでエラー・ログ・ファイルを指定することができます。これらのオプションを付けたコマンドは、例えば次のようになります。

```
p4d -r /usr/local/p4root -J /var/log/journal -L /var/log/p4err -p 1818 &
```

-r、-J、-L（およびその他の）オプションについては、25 ページの「PERFORCE のサポート : バックアップとリカバリ」で説明します。サーバ・オプションの全リストは、137 ページの「PERFORCE サーバ（p4d）・リファレンス」に掲載されています。

サーバを停止する

PERFORCE サーバを停止するときは、次のコマンドを使用します。

```
p4 admin stop
```

`p4 admin stop` コマンドを使用することができるのは PERFORCE スーパー・ユーザだけです。

99.2 より前のリリースの PERFORCE をご使用の場合は、`p4d` サーバのプロセス ID を確認し、UNIX シェルから手動でこのプロセスを終了させる必要があります。`p4d` が `SIGKILL` 信号を受け取ったときにファイルを更新中であれば、データベースが不整合を起こす可能性があるので、`kill -9`（`SIGKILL`）ではなく `kill -15`（`SIGTERM`）を使用します。

Windows 環境下でのインストール

PERFORCE を Windows 上にインストールするには、PERFORCE 社の Web サイトにある Downloads ページから入手できる PERFORCE インストーラ（`perforce.exe`）を使用します。

```
http://www.perforce.com/perforce/loadprog.html
```

PERFORCE インストーラには、次のようなオプションが用意されています。

- PERFORCE コマンドライン・クライアント・ソフトウェアのインストール（“User install”）

  ユーザ・インストール・オプションでは PERFORCE コマンドライン・クライアント（`p4.exe`）のみがインストールされます。その他の Windows 環境の PERFORCE クライアントである PERFORCE ビジュアル・クライアント（P4V）や PERFORCE Windows クライアント（P4Win）などとサードパーティのプラグインは、個別にダウンロードおよびインストールが可能です。

- PERFORCE の Windows サーバまたはサービスとしてのインストール（“Administrator typical”、“Administrator custom”）

  これらのオプションでは、PERFORCE クライアント・プログラム、PERFORCE Windows サーバ（`p4d.exe`）およびサービス（`p4s.exe`）の実行ファイルをインストールすることができ、また、Windows 環境下で動作している既存の PERFORCE サーバまたはサービスを自動的にアップグレードすることもできます。

  Windows 2000 以上で使用する場合、PERFORCE をサービスとしてインストールするには Administrator の権限、サーバとしてインストールするには Power User の権限が必要です。

- PERFORCE のアンインストール :

  このオプションにより、PERFORCE サーバ、サービス、クライアント実行ファイル、レジストリ・キー、およびサービス・エントリを削除します。サーバ・ルートに格納されている PERFORCE データベースとディポ・ファイルは保存されます。

Windows へのインストールについて詳しくは、121 ページの「PERFORCE インストーラの使用」をご覧ください。

Windows のサービスとサーバ

本書において「PERFORCE サーバ」および「p4d」という用語は、いずれも「PERFORCE クライアント・プログラムからの要求を処理するプロセス」を指し、Windows サーバ・プロセスと Windows サービス・プロセスの違いを明記することが適切な場合を除き、同義に使われています。NT サーバと NT サービスの違いが重要な場合には、その違いを明記します。

UNIX システムでは、このようなバックエンド・タスクを担当するのは PERFORCE「サーバ」プログラム（p4d）しかありません。ただし Windows では、このバックエンド・プログラムはブート時に実行される Windows サービス（p4s.exe）プロセス、あるいはコマンド・プロンプトから手動で起動するサーバ（p4d.exe）プロセスとして起動するよう構成することができます。

PERFORCE サービス（p4s.exe）と PERFORCE サーバ（p4d.exe）の実行ファイルは、相互にコピーになっています。ファイル名が異なるだけで、この 2 つは全く同じものです。実行時、これらの実行ファイルは起動時に使用した名前の最初の 3 文字（p4s または p4d）で動作を決定します。（すなわち、p4smyservice.exe という名前のコピーを起動すればサービスとして、p4dmyserver.exe という名前のコピーを起動すればサーバとして起動します。）

通常、PERFORCE はサーバではなくサービスとしてインストールすることをお勧めします。サービスとサーバの違いについて、詳しくは 123 ページの「Windows サービスと Windows サーバ」をご覧ください。

PERFORCE の起動と停止

PERFORCE を Windows 環境下でサービスとしてインストールした場合には、マシン起動時に必ずサービスが起動します。PERFORCE サービスの動作は［**コントロールパネル**］の［**サービス**］アプレットを用いて制御します。

PERFORCE を Windows 環境下でサーバとしてインストールした場合には、コマンド・プロンプトから p4d.exe を起動します。Windows 環境下で p4d に使用できるオプションは、UNIX 環境下で使用するオプションと同じです。

PERFORCE サービス（またはサーバ）を停止するときは、次のコマンドを使用します。

```
p4 admin stop
```

このコマンドは PERFORCE スーパー・ユーザだけが使えます。

これ以前のバージョンの PERFORCE では、［**コントロールパネル**］の［**サービス**］アプレットを用いて手動でサービスを停止します。コマンド・プロンプト・ウィンドウで実行しているサーバは、ウィンドウ内に CTRL-C と入力するか、コマンド・プロンプト・ウィンドウの［閉じる］アイコンをクリックして停止します。このような方法で 99.2 以前のバージョンの PERFORCE では手動停止できますが、サーバまたはサービスを急停止させるので、必ずしも「いい方法」とはいえません。99.2 以降では p4 admin stop コマンドを使用できるので、手動停止は行わないでください。

サーバをアップグレードする

PERFORCE を Windows にインストールしている場合も、UNIX にインストールしている場合も、アップグレードをするときには必ずサーバのバックアップを取らなければ*なりません*（30 ページの「バックアップの手順」参照）。

> *PERFORCE サーバのアップグレード前にはアップグレード対象のサーバに関連するリリースノートをお読みください。*

古いクライアント・プログラムを新しいサーバで使う

古いバージョンの PERFORCE クライアント・プログラムを、新しいバージョンのサーバとともに使用する場合、基本的には問題なく動作します。ただし、新しいサーバの機能には、クライアント・プログラムのアップグレードを必要とするものもあります。一般に、古いクライアント・プログラムのユーザは、そのプログラムと同じリリース・レベルのサーバの機能を利用することはできますが、それ以後のサーバのアップグレードで提供された新しいサーバ機能を利用することはできません。

ただし、PERFORCE のリモート・ディポの機能は例外です。リモート・ディポは、ご使用の PERFORCE サーバが*すべて*リリース 99.2 以降でない限り、動作を保証されていません。

リリース 2005.1 以降に関する重要な注意事項

リリース `2005.1` では PERFORCE サーバはファイル長のメタデータをリビジョン毎に追跡します。新規にファイルがサブミットされると、ファイル長のメタデータが自動的にデータベースに追加されます。ただし、2005.1 にアップグレード後も、少なくとも 1 度は `p4 verify -u` を実行し、ファイル長が保存されていない 2005.1 より前のファイルについてファイル長のメタデータを更新しなければなりません。

大規模なサイト（リビジョン数が一千万を超える規模）では、2005.1 にアップグレードした直後、管理者が一度にレポジトリ全体のファイル長メタデータを更新しようとするとメモリーの制限をオーバーしてしまうことがあります。このような場合は `-m maxRevs` オプションを使用し、一回のコマンドで更新するリビジョンの数を制限してください。

例えば

```
p4 verify -u -m 1000000 //...
```

とすると、ファイル長メタデータを一度に再計算するファイルの数を 100 万個に制限することができ、管理者は `p4 verify` を何度かに分けて実行してファイル長のメタデータの再計算を行うことができます。

リリース 2001.1 以降に関する重要な注意事項

小規模なシステム（サブミット済チェンジリスト 1000 未満）の場合、2001.1（またはそれ以降の）サーバをインストールすると、バージョン 98.2 以降で作成された基礎データベースを自動的にアップグレードします。

中規模以上のシステムの場合は、データベースを手動でアップグレードしなければなりません。アップグレードされたデータベースは、通常は 2001.1 以前のデータベースより小さくなりますが、アップグレードのプロセスで必要なファイルを保存するために既存のデータベースの約 3 倍のサイズが一時的に必要になることがあります。

> **注** *ディスク容量に制約がある場合は、リリース・ノートを見て、必要なディスク容量をより正確に予測してください。*
>
> *アップグレード中のジャーナル作成をオフ（`P4JOURNAL` をオフ）にしておけば、アップグレードに必要なディスク容量を減らすことができます。（ただし、アップグレード終了後に再度オンにするのを忘れずに！）*

リリース 97.3 以前から 2001.1 以降にアップグレードする場合には、自動アップグレードの手順も手動アップグレードの手順も適用されません。おそらく中間チェックポイントを作成する必要があるでしょう。リリース 97.3 以前のサーバをアップグレードする場合には、事前に PERFORCE テクニカル・サポートにご相談ください。

UNIX 環境下でのアップグレード

現在の PERFORCE サーバを新しいバージョンにアップグレードするには、PERFORCE ライセンス・ファイルが有効でなければなりません。満了したライセンスは、アップグレード後のサーバでは無効です。(ただし、ライセンスなしの 2 ユーザのシステムを動作させる場合には、ライセンス・ファイルの制約は問題になりません。)

アップグレードのプロセスを実行するときは、必ず 30 ページの「バックアップの手順」に従ってサーバをバックアップしてください。

より安全を期して、アップグレードを行うときには `p4 verify` を実行してください。詳しくは、46 ページの「サーバ・アップグレード中の認証」をご覧ください。

> *リリース 2001.1 以降へのアップグレードは、データベース・ファイルのアップグレードを必要とします。後でダウングレードするときはバックアップからのリストアが必要になります。2001.1 以降へのアップグレードを行いながら、2001.1 より古いサーバもフォールバック・オプションとしてとっておきたい場合には、サーバを停止した上でサーバ・ルート全体 (`db.*` ファイルを含む) をバックアップする必要があります。*

UNIX リリース 98.2 以降からのアップグレード

有効なライセンスを持っている (またはライセンスがいっさい必要ない) 場合に、リリース 98.2 以降からのアップグレードを行うには、以下の手順に従ってください。

1. プラットフォームに新しい `p4d` 実行可能プログラムをダウンロードします。
2. `p4d` のカレント・インスタンスを停止します。
3. チェックポイントを作成し、現バージョンをバックアップします。
4. 希望の場所に新しい `p4d` をインストールします。
5. `p4d -xu` を実行し、データベースをアップグレードします。

> **注** *サーバのチェンジ数が 1000 未満の場合には、アップグレードは自動的に行われます。1000 以上の場合には、手動で `p4d -xu` を実行する必要があります。アップグレードを完全に行うためには、十分なディスク容量を確保しておかなければなりません。*

6. 新しい `p4d` をサイトの通常のパラメータで再起動します。

これでユーザも新しいサーバを使用できるようになります。

Windows 環境下でのアップグレード

Windows の場合は、インストーラ (`perforce.exe`) をダウンロードし、ダイアログに従ってインストールを行います。

Windows 上のアップグレード・プロセスはきわめて堅実です。アップグレード中に何らかのエラー条件が発生した場合には、アップグレード前の PERFORCE サーバまたはサービスに戻ることができます。

> **注** *サーバのチェンジ数が 1000 未満の場合には、アップグレードは自動的に行われます。1000 以上の場合には、手動で `p4d -xu` を実行する必要があります。アップグレードを完全に行うためには、十分なディスク容量を確保しておかなければなりません。*

アップグレード中に疑問または問題が生じた場合には、PERFORCE テクニカル・サポートまでご連絡ください。

インストールと管理に関する情報

リリースおよびライセンス情報

PERFORCE サーバは、サポートするユーザの数に応じてライセンスされます。

ライセンス情報はサーバ・ルート・ディレクトリの `license` というファイルに格納されています。`license` ファイルは PERFORCE ソフトウェアによって供給されるプレーン・テキスト・ファイルです。`license` ファイルがない場合、PERFORCE サーバは 2 人のユーザと 5 つのクライアント・ワークスペースにアクセスを制限します。

`p4 license` コマンドを使用すると、PERFORCE サーバを停止せずに既存のファイルを更新することができます。詳しくは、43 ページの「新規ライセンス許可ユーザの追加」をご覧ください。

現在のライセンス情報は、`license` ファイルがあるサーバ・ルート・ディレクトリから `p4d -V` を実行するか、あるいはコマンドライン（`p4d -V -r server_root`）上または環境変数 `P4ROOT` にサーバ・ルート・ディレクトリを指定することによって、閲覧することができます。

サーバが動作しているときには、`p4 info` を用いてもライセンス情報を閲覧することができます。

サーバのバージョンは `p4d -V` または `p4 -V` を実行しても表示されます。

定期的にバックアップする

PERFORCE のデータを定期的にバックアップすることはきわめて重要です。定期的バックアップの要点は次のとおりです。

- ジャーナルを確実に作成する。
- 定期的にチェックポイントを作成する。
- 定期的に `p4 verify` を使用する。

バックアップとリストアの手順については、25 ページの「PERFORCE のサポート : バックアップとリカバリ」をご覧ください。

サーバ・ルートとジャーナルに別々の物理ドライブを使用する

UNIX であれ、Windows であれ、データベース・ファイルおよびバージョン化ファイルが格納される `P4ROOT` ディレクトリは、ジャーナル・ファイルとは別の物理ドライブに 置くことをお勧めします。

ジャーナルを別のドライブに保存しておけば、たとえディスクが破損して `P4ROOT` を含むドライブが壊れても、その破損がジャーナル・ファイルに影響を及ぼすことは*ありません*。したがって、ジャーナル・ファイルを用いて損失または損傷したメタデータをリストアすることができます。

詳しくは、25 ページの「PERFORCE のサポート : バックアップとリカバリ」をご覧ください。

プロテクションとパスワードを使用する

管理者が PERFORCE スーパー・ユーザを定義しない限り、どの PERFORCE ユーザも PERFORCE スーパー・ユーザであり、あらゆるファイルに対してあらゆる PERFORCE コマンドを実行することができます。管理者は新しい PERFORCE サーバをインストールしたら、できるだけ早く、

```
p4 protect
```

を使って PERFORCE スーパー・ユーザを定義してください。詳しくは、69 ページの「PERFORCE の管理 : プロテクション」をご覧ください。

例えば、ユーザ A がパスワードを定義していない場合は、他のユーザが `-u` オプションを使用するか、`P4USER` を A というユーザ名に設定することにより、ユーザ A として操作を行う (ユーザ A になりすます) ことができます。PERFORCE パスワードを使用すれば、この「なりすまし」を防ぐことができます。詳しくは『P4 ユーザーズ . ガイド』をご覧ください。

ユーザのパスワードを設定 (または再設定) するには、(PERFORCE スーパー・ユーザとして) `p4 passwd` *`username`* を使用し、新たに設定するユーザのパスワードを入力するか、(やはり PERFORCE スーパー・ユーザとして) `p4 user -f` *`username`* を実行し、ユーザ指定フォームに新しいパスワードを入力します。前者のコマンドは 99.1 以降のリリースでサポートされており、後者のコマンドは 97.3 以降のすべてのリリースでサポートされています。

セキュリティ意識の高い PERFORCE スーパー・ユーザであれば、`p4 protect` も使用して非特権ユーザに `list` 以上のレベルのアクセスが認められないようにし、また、どのユーザも必ず PERFORCE パスワードを持つように管理するはずです。

成長を見越した十分なディスク容量の割当

バージョン化ファイルの合計サイズは時間とともに増大するので、`P4ROOT` ディレクトリに現在のバージョン化ファイルの合計サイズの 3 倍のサイズを保持するのに十分な容量を割り当てるほかに、ディポ・ファイルのリスト、ファイルの状態、ファイル・リビジョンの履歴を保存するデータベース・ファイルを保持するために、さらに 1 ユーザ 1 ファイル当たり 0.5KB の容量を割り当てておくとよいでしょう。

さらに詳しいディスクサイズの推定方法については、110 ページの「ディスク容量の割り当て」をご覧ください。

インストール後のディスク容量の管理

PERFORCE のバージョン化ファイルはすべて、データベース・ファイルのほか、(デフォルトでは) チェックポイントやジャーナルとも同様に、サーバ・ルートのサブディレクトリにあります。ディスク容量が少なくなっているときには、ディスク容量の使用を制限するために次のような方法を考えてください。

- ジャーナル・ファイルを別の物理ディスクに保存するように PERFORCE を設定します。環境変数 `P4JOURNAL` または `p4d -J` を用いてジャーナル・ファイルの保存場所を指定します。
- チェックポイントを日常的に作成し、ジャーナル・ファイルを小さくします。
- チェックポイントを圧縮するか、`-z` オプションを用いて `p4d` にチェックポイントを作成時に圧縮するよう指示します。
- `p4d` コマンドと `-jc` *`prefix`* オプションを用いてチェックポイントを別のディスクに書き込みます。または、デフォルトのチェックポイント・ファイルを使用する場合は、チェックポイントを別のドライブにバックアップした上で、コピーしたチェックポイントをルート・ディレクトリから消去します。チェックポイントを別のドライブに移動させるのは、ディスク容量確保の面からばかりでなく、ハードウェアの故障からのリカバリを可能にするためにもよい方法です。ハードウェアの故障からのリカバリ時には古いチェックポイントが必要になり、チェックポイントとジャーナルのファイルがディポと同じディスクにあると、ハードウェアの故障時にデータベースをリストアすることができなくなる可能性があります。
- UNIX システムでは、シンボリック・リンクを用いることにより、ディポ・ディレクトリの一部または全部を他のディスクに再配置することができます。シンボリック・リンクを用いてディポ・ファイルを他のボリュームへ移すときには、PERFORCE サーバを停止させてからリンクを作成します。

- 管理しているシステムのデータベース・ファイルがチェックポイントのサイズの10倍を超えたときには、それをチェックポイントから再作成し、ファイルのサイズを小さくすることができます。119 ページの「データベース・ツリーのバランス回復のためのチェックポイント」をご覧ください。
- `p4 sizes` コマンドを使用すれば、お使いのインストール全体、またはインストール内の指定箇所で現在消費されているディスクの総容量を監視することができます。111 ページの「ディスク空き容量の監視」をご覧ください。

大容量ファイルシステムのサポート

初期のバージョンの PERFORCE サーバは、一部のオペレーティング・システムと同様、PERFORCE データベース・ファイル（サイトのメタデータを保持する `P4ROOT` ディレクトリの `db.*` ファイル）を 2GB のサイズに制限します。`db.have` ファイルは、現在クライアント・ワークスペースに同期しているファイルのリストを保持し、最も急速に成長する傾向があります。

PERFORCE データベース・ファイルのいずれかが 2GB のレベルを超えて成長することが予想されたら、PERFORCE サーバを大容量ファイルをサポートできるプラットフォームにインストールしてください。次のようなオペレーティング・システムと PERFORCE サーバ・リビジョンの組み合わせは、2GB を超える大容量のデータベース・ファイルをサポートします。

	OS バージョン	PERFORCE サーバ・リビジョン
Windows NT, 2000, XP	全バージョン、 NT は、SP6 推奨	98.2/8127 以降
FreeBSD		98.2/5713 以降
Linux x86	Kerneis 2.4.0 以降	2002.2/21749 以降
HP-UX	HP-UX11.11 以降	2001.1/26433 以降
Solaris	2.6 以降	*2.6 以降用にコンパイルされた* 98.2/7488
Tru64 UNIX (Digital UNIX, OSF/1)	全バージョン	98.2/5713 以降
SGI IRIX 6.2	全バージョン	98.2/5713 以降
SGI IRIX 5.3	SGI 供給の `xfs` アップグレード・バージョンのみ	98.2/5713 以降 OS レベルで求められている `xfs` アップグレード

UNIX および NFS のサポート

PERFORCE サーバ・プロセスは、NFS を実装した Solaris 2.6 以降でサポートされます。これはテストによって実証されています。PERFORCE クライアント・プログラムが直接 `P4ROOT` のファイルにアクセスすることはありません。`P4ROOT` へのアクセスを必要としているプロセスは `p4d` サーバのみです。

この結果、Solaris 2.6 以降では、NFS マウントされたファイル・システム上でジャーナル、ログ、ディポ、および `db.*` ファイルを保存することができます。ただし、パフォーマンスを最大にするため、サーバ・ルート（`P4ROOT`）は NFS マウントされたボリュームではなく、ローカル・ディスク上に設定してください。

非商用実装の NFS（例えば Linux、FreeBSD）上でのファイル・ロッキングについては、まだ問題が残っています。これらのプラットフォームでは、ジャーナル、ログ、ディポ、および db.* ファイルは NFS マウントされたボリューム*ではなく*、サーバ・マシンのローカル・ドライブに保存してください。

上記の問題は PERFORCE サーバ・プロセス（`p4d`）のみに影響します。PERFORCE クライアント・プログラム（`p4`、PERFORCE コマンドライン・クライアントなど）は、ユーザのホーム・ディ

レクトリに配置されたクライアント・ワークスペースなど、常に NFS マウントされたドライブ上のクライアント・ワークスペースで作業することができます。

Windows: ネットワーク・ドライブにはユーザ名とパスワードが必要

デフォルトでは、PERFORCE サービスは Windows のローカル・システム・アカウントのもとで動作します。Windows はネットワーク・ドライブ上のファイルにアクセスするのにリアル・アカウント名とパスワードを必要とするので、Windows で `P4ROOT` をネットワーク・ドライブに指定し、PERFORCE をサービスとしてインストールしようとすると、インストーラがアカウント名とパスワードを要求します。その場合、PERFORCE サービスは与えられたデータで構成され、System としてではなく指定されたユーザとして動作します。(サービスを動作させるアカウントはマシン上で `Administrator` の特権を持っていなければなりません。)

PERFORCE はルート・ディレクトリがネットワーク・ドライブ上にあっても信頼できる動作をしますが、データベースへの書き込みがすべてネットワーク経由で行われるので、その場合にはかなりのパフォーマンスが犠牲になります。最適なパフォーマンスを確保するには、Windows サービスをネットワーク・ドライブではなくローカル・ドライブで動作するようにインストールします。

詳しくは、125 ページの「ネットワーク・ドライブへの PERFORCE サービスのインストール」をご覧ください。

UNIX: P4D を非特権ユーザとして動作させる

PERFORCE サーバ・プロセスには、特権を持っていなくてもアクセスできます。したがって、セキュリティを確保するためには、`p4d` を `root` として動作させないようにするか、または別の方策として、`p4d` プロセスの所有者に `root` レベルの特権を与えます。

`p4d` を管理する非特権 UNIX ユーザ (例えば `perforce`) を作成し、(オプションで) その目的のための UNIX グループ (例えば `p4admin`) を作成します。`umask (1)` コマンドを利用して、サーバ・ルート (`P4ROOT`) とその下に生成されたすべてのファイルおよびディレクトリを UNIX ユーザ `perforce` にしか書き込めないものにして、(オプションで) UNIX グループ `p4admin` のメンバーには読み取り可能なものにします。

この設定では、UNIX ユーザ `perforce` として動作している PERFORCE サーバ (`p4d`) は、サーバ・ルートのファイルに書き込みができますが、そのファイルの読み取りまたは上書きができるユーザはいません。その結果、`p4d` によって作成されたファイル (ディポ・ファイル、チェックポイント、ジャーナルなど) に対するアクセス権を信頼のできるユーザに与えるには、それらのユーザを UNIX グループ `p4admin` に追加します。

Windows

Windows 上では、ディレクトリのパーミッションはデフォルトでしっかりと設定されています。PERFORCE をサーバとして動作させているときには、サーバをコマンド・プロンプトから起動したユーザしかサーバ・ルートにアクセスすることはできません。PERFORCE がサービスとしてインストールされている場合には、サーバ・ルートのファイルはローカル・システム・アカウントによって所有され、`Administrator` のアクセス権を持つユーザしかアクセスすることはできません。

エラー・ログ

PERFORCE サーバのエラー出力ファイルは、`p4d` に `-L` オプションを付けるか、環境変数 `P4LOG` を用いて指定することができます。エラー出力ファイルが定義されていない場合には、エラーは `p4d` プロセスの標準エラーとしてダンプされます。`p4d` はどのエラー・メッセージも必ずユーザに届ける仕組みになっていますが、エラーが発生し、そのエラーが受信される前にクライアント・プログラムが切断された場合にも、`p4d` はそのエラーをエラー出力に記録します。

PERFORCE サーバはデバッグ用のトレース・オプションもサポートしています。詳しくは、56 ページの「PERFORCE サーバ・トレースおよび追跡用オプション」をご覧ください。

ファイルへのアクセスを記録する

サイトでファイルへのユーザ・アクセスを追跡する必要がある場合は、p4d に -A オプションを付けるか環境変数 P4AUDIT を使用することにより、監査を有効にして PERFORCE サーバの監査ログ・ファイルを指定します。監査が有効になっている場合、ユーザがファイルにアクセスするたびにレコードが監査ログ・ファイルに格納されます。このオプションはアクティブなサーバ上で相当多くのディスク容量を消費する可能性があります。

詳しくは、57 ページの「ユーザによるファイル・アクセスの監査」をご覧ください。

大文字 / 小文字の区別

PERFORCE サーバが Windows または UNIX のどちらの環境で動作しているにせよ、サイトがクロスプラットフォーム開発を行っている場合には（PERFORCE クライアント・プログラムを Windows と UNIX の両方のワークステーションで使用している場合には）、ユーザは大文字 / 小文字の区別の問題についてある程度のことを知っておく必要があります。

詳しくは、53 ページの「大文字／小文字の区別とマルチプラットフォーム開発」をご覧ください。

パフォーマンス調整

PERFORCE は、ネットワークの帯域幅も CPU の能力も効率よく使用します。サーバのパフォーマンスを左右する最も重要な要素は、サーバのディスク I/O サブシステムの効率と、ユーザ独自の PERFORCE 操作で参照されるファイルの数です。

パフォーマンス調整について、詳しくは 109 ページの「PERFORCE のパフォーマンス調整」をご覧ください。

第 2 章

PERFORCE のサポート：バックアップとリカバリ

PERFORCE サーバは 2 種類のデータを保存します。*バージョン化ファイル*と*メタデータ*です。

- *バージョン化ファイル*とは、PERFORCE ユーザによってサブミットされたファイルのことです。このファイルは*ディポ*というディレクトリ・ツリーに保存されます。

  PERFORCE システムでは、ディポごとにサーバのルート・ディレクトリの下にサブディレクトリが 1 つ設けられます。各ディポでは、このサブディレクトリ直下のディレクトリ・ツリーにバージョン化ファイルが保存されます。

- *データベース・ファイル*は、チェンジリスト、作業状態のファイル、クライアント仕様、ブランチ仕様、また、バージョン化ファイルの履歴および現状に関するデータなどを含む*メタデータ*を保存します。

  データベース・ファイルは、サーバ・ルート・ディレクトリの最上階層に `db.*` ファイルとして表示されます。どの `db.*` ファイルも 1 つのバイナリコード化されたデータベース・テーブルを含んでいます。

バックアップとリカバリの考え方

ディスク容量が不足したり、ハードウェアが故障したり、システムがクラッシュしたりすると、PERFORCE サーバ上のファイルが壊れることがあります。このため、PERFORCE のルート・ディレクトリ構造 – バージョン化ファイルとデータベース – は定期的に一括してバックアップしておく必要があります。

バージョン化ファイルは PERFORCE サーバ・ルートの下のサブディレクトリに保存されており、バックアップから、完全なかたちで、直接リストアすることができます。

一方、PERFORCE のデータベースを構成しているファイルは、従来のバックアップ・プログラムによりアーカイブされている場合、トランザクションの完全性が保証されません。その結果、通常のシステム・バックアップから `db.*` ファイルをリストアしようとすると、不完全なデータベースがリストアされるかもしれません。破壊されたデータベースの完全性を確実に取り戻す方法は、PERFORCE のチェックポイント・ファイルやジャーナル・ファイルから `db.*` ファイルを再構築する以外にありません。

- *チェックポイント*は、ある特定の時点におけるデータベースのスナップショットまたはコピーのようなものです。
- *ジャーナル*は、最後にスナップショットがとられてからそのデータベースに対して行われた更新を記録するログです。

チェックポイント・ファイルは通常、元のデータベースよりはるかに小さくなり、圧縮することによって、さらに小さくすることができます。それに対して、ジャーナル・ファイルはかなり大きくなることがあり、チェックポイントが作成されるたびに、古いファイルはリネームさ

れて別ファイルとして保存されていきます。このため、古いジャーナル・ファイルをオフラインでバックアップすれば、ローカルでより多くのディスク容量を解放することができます。

チェックポイントもジャーナルもテキスト・ファイルで、フォーマットは同じです。チェックポイントがあれば PERFORCE のデータベースはリストアできますが、チェックポイント作成以後のジャーナルがあればより確実です。

> *チェックポイントとジャーナルは、あくまで PERFORCE データベース・ファイルの記録であり、ディポ・ディレクトリに格納されているバージョン化ファイルでは**ありません**!*
>
> *ディポ・ファイル（バージョン化ファイル）は、常にチェックポイント作成後に OS に標準で用意されているバックアップ・コマンドでバックアップしてください。*

PERFORCE のデータベースに保存されている情報は、バージョン化ファイルと同様、失ったら取り返しのつかないものなので、チェックポイントとジャーナルの作成は PERFORCE サーバの管理に不可欠な作業であり、通常のバックアップ・サイクルに組み込む必要があります。

チェックポイント・ファイル

*チェックポイント*は、PERFORCE のデータベースのメタデータを再生するのに必要なすべての情報が納められたファイルです。チェックポイントを作成すると、PERFORCE のデータベースはロックされ、それが 1 つの全体として調和のとれたスナップショットに納められます。

バージョン化ファイルは、チェックポイントと別にバックアップされます。つまり、チェックポイントはバージョン化ファイルの内容を*含んでいないので*、***バージョン化ファイルをチェックポイント・ファイルからリストアすることはできません***。ただし、チェンジリスト、ラベル、ジョブなどは、すべてチェックポイントからリストアできます。

データベースを完全にリストアするには、バージョン化ファイルのバックアップ以前に作成されたチェックポイントがなければなりません。つまり、バージョン化ファイルのバックアップを始めた時点のデータベースの状態が、チェックポイントに完全に記録されている必要があります。

定期的にチェックポイントを作成することは、ジャーナルが大きくなりすぎないようにするためにも重要です。チェックポイントは、システムのバックアップの直前に作成するようにするとよいでしょう。

チェックポイントの作成

チェックポイントは、自動的には作成されません。手動で、あるいは何らかの方法で PERFORCE サーバ・マシン上でチェックポイントのコマンドを実行しなければなりません。コマンドは、`p4d` の後に `-jc` (journal-create) のオプションを付けます。

```
p4d -r root -jc
```

PERFORCE サーバ（`p4d`）の動作中にチェックポイントを作成することができます。チェックポイントはサーバのルートディレクトリ（`P4ROOT`）に作成されます。

チェックポイントを作成するときには、`p4d` がデータベースをロックし、P4ROOT ディレクトリの `checkpoint.n` という名前のファイルにその内容をダンプします。*n* は連番です。`p4d` は、データベースのロックを解除する前にカレント・ジャーナルを `P4ROOT` ディレクトリの（カレント・ジャーナルが保存されているディレクトリに関係なく）`journal.n-1` という名前のファイルにコピーし、その上でカレント・ジャーナルを空にします。

連番はジャーナルのロールフォワードの性質を示しています。より古いチェックポイントのデータベースをリストアするときは、この連番を合わせます。`checkpoint.6` の時点のデータベースは `checkpoint.5` に保存されたデータベースと、それ以降の `journal.5` に記録さ

れた変更を合わせることによってリストアされます。すなわち、カレント・データベースは、最大番号のチェックポイント `checkpoint.n` とカレント・ジャーナル `journal` に記録された変更に反映されています。通常は、このカレント・データベースをリストアすれば十分です。

チェックポイントやジャーナルのプレフィクスまたはディレクトリ位置を指定するには、`-jc` オプションを使います。例えば、

```
p4d -jc prefix
```

というコマンドでチェックポイントを作成すれば、作成されるチェックポイント、ジャーナルのファイル名は `prefix.ckp.n`、`prefix.jnl.n` になります。ただし、*`prefix`* の部分はコマンドラインで指定でき、*`n`* は連番です。プレフィクスの指定がなければ、デフォルトのファイル名 `checkpoint.n` と `journal.n` が使用されます。チェックポイントおよびジャーナルはプレフィクスの一部としてディレクトリを指定すれば、任意のディレクトリに保存できます (バックアップのジャーナルは、カレント・ジャーナルの保存ディレクトリに関係なく、`P4ROOT` ディレクトリに保存されます)。s

PERFORCE サーバが稼動しているマシンにログインせずにチェックポイントを作成するには、次のコマンドを使います。

```
p4 admin checkpoint [-z] [prefix]
```

`p4 admin checkpoint` は、`p4d -jc` と実行するのと同等です。`p4 admin` は PERFORCE スーパー・ユーザしか使えません。

定期的にチェックポイントを作成する自動化プログラムを設定することができます。ただし、このプログラムの出力を常にチェックし、チェックポイントが順調に作成されていることを確認してください。作成が成功すると、チェックポイント・ファイルは圧縮、アーカイブ、ディスク移動が可能です。ほぼ同時にすぐにディポのサブディレクトリに保存されたバージョン化ファイルもバックアップする必要があります。

バックアップからリストアするとき、*チェックポイントはディポのファイルと同じかそれより古い時期に作成されたものでなければなりません。*バージョン化ファイルはチェックポイントより新しくても構いませんが、その逆はいけません。そしてこのタイミングのずれをできるだけ小さくするのが望ましいです。

チェックポイント・コマンド自体が機能しないときには、直ちに PERFORCE のテクニカル・サポートまでご連絡ください。通常、チェックポイントの不具合は資源（ディスク容量、パーミッション等）に問題があるときに発生しますが、こういった問題は適正に処理しないとデータベースを危険にさらす恐れがあります。

ジャーナル・ファイル

*ジャーナル*は、最後のチェックポイント以降のあらゆるデータベースの変更を追跡する実行トランザクション・ログです。2 つのチェックポイントの間の架け橋になります。

月曜日のチェックポイントと、それから水曜日までのジャーナルがあれば、それらの 2 つのファイル（月曜日のチェックポイント・ファイルとカレント・ジャーナル）には、水曜日に作成するチェックポイントと同じ情報が含まれています。例えば、水曜の正午に PERFORCE のデー

タベースが壊れるディスク・クラッシュが起きた場合、まだ水曜のチェックポイントが作成されていなくても、そのデータベースをリストアすることができるのです。

> *デフォルト設定では、カレント・ジャーナル・ファイルは* `journal` *という名前で* P4ROOT *ディレクトリに置かれます。ただし、ディスク故障でルート・ディレクトリが壊れたら、ジャーナル・ファイルにアクセスできなくなります。*
>
> *ジャーナルが* P4ROOT *以外のファイルシステムに書き込まれるようにシステムを設定することを強くお勧めします。これを行うには、ジャーナル・ファイルの名前を環境変数* P4JOURNAL *で指定するか、* `p4d` *の起動時に* `-J` *filename オプションを使用します。*

データベースをリストアするには、常に最新のジャーナル・ファイルにアクセスできる状態にしておくだけで十分ですが、仮に少し前のチェックポイントにリストアする必要が生じるとすれば、古いジャーナルを古いチェックポイントとともにアーカイブする方法も考えられます。

Windows 環境でジャーナル作成を有効にする

Windows 環境で、インストーラ（`perforce.exe`）を用いて PERFORCE のサーバまたはサービスをインストールした場合には、ジャーナル作成は自動的にオンになります。

インストーラを用いずに PERFORCE をインストールする場合（この方法の一例については、125 ページの「WINDOWS 環境下での複数の PERFORCE サービスの使用」をご覧ください）、Windows 環境での手動のインストールでジャーナル作成を有効にするために、`journal` という名前の空ファイルを作成する必要はありません。

UNIX 環境でジャーナル作成を有効にする

UNIX 環境の場合でも、ジャーナル作成は自動的にオンになります。

もし、P4JOURNAL が未設定のまま（さらにコマンドラインで場所が設定されていない）であっても、ジャーナルを作成するデフォルトの場所は、`$P4ROOT/journal` となります。

ジャーナル作成を有効にした後

ジャーナル作成を有効にしたら、すぐに `p4d -jc` で（必要なら `-J` *journalfile* を付け）チェックポイントを作成しましょう。ジャーナル作成を有効にしたら、ジャーナル・ファイルのサイズを抑制するため定期的なチェックポイントの作成が必要です。カレント・ジャーナルが極端に大きくなるのは、チェックポイントが必要なしるしです。

チェックポイント作成のたびに、それまでのジャーナルがリネームされ、新しいジャーナルの作成が始まります。それまでの `journal` は `journal.`*n* とリネームされ（*n* は連番）、新しいジャーナル・ファイルが作成されます。

デフォルトでは、ジャーナルはサーバ・ルート （P4ROOT）の `journal` ファイルに書き込まれます。ディスク・クラッシュを確実に防ぐ方法はないので、ジャーナル・ファイルと PERFORCE サーバ・ルートは別のファイルシステム、できれば別の物理ドライブに保存してください。ジャーナルの名前や場所は、環境変数 P4JOURNAL でジャーナル・ファイルの名前を指定するか、`p4d` に `-J` *filename* オプションを付けることで変えられます。

> **警告！** `-J` *filename オプションでジャーナル・ファイルを作成したら、以後のチェックポイントでも必ず同じファイルを使用するようにしないと、ジャーナルは正しくリネームされなくなります。*

P4JOURNAL を使う方法または p4d に -J *journalfile* オプションを付ける方法のいずれでも、ジャーナル・ファイルの名前は絶対パス、またはサーバ・ルートに対する相対パスのどちらでも指定できます。

例 : *ジャーナル・ファイルの指定*

次のコマンドでサーバを起動すると、

```
$ p4d -r $P4ROOT -p 1666 -J /usr/local/perforce/journalfile
PERFORCE Server starting...
```

次のコマンドでチェックポイントを作成するか、

```
$ p4d -r $P4ROOT -J /usr/local/perforce/journalfile -jc
Checkpointing to checkpoint.19...
Saving journal to journal.18...
Truncating /usr/local/perforce/journalfile...
```

または P4JOURNAL *を* /usr/local/perforce/journal *に設定して次のコマンドを使用する必要があります。*

```
$ p4d -r $P4ROOT -jc
Checkpointing to checkpoint.19...
Saving journal to journal.18...
Truncating /usr/local/perforce/journalfile...
```

環境変数 P4JOURNAL *（またはコマンドラインの指定）が PERFORCE サーバ起動時に使用した指定と食い違っていれば、チェックポイントは作成されますが、ジャーナルは保存されないし、更新されることもありません。これはなんとしても避けたいことです!*

ジャーナル作成を無効にする

ジャーナル作成を無効にするには、サーバを停止し、（もしあれば）既存のジャーナル・ファイルを削除し、環境変数 P4JOURNAL を off に設定した後、-J オプションを付けずに p4d を再起動します。

バージョン化ファイル

チェックポイントとジャーナル・ファイルから再構築されるのは、PERFORCE のデータベース・ファイルだけです。バージョン化ファイルは PERFORCE サーバ・ルートの下のディレクトリに保存されており、別個にバックアップする必要があります。

バージョン化ファイルのフォーマット

バージョン化ファイルは、サーバ・ルートのサブディレクトリに保存されます。テキスト・ファイルは *filename*,v の形式のファイル名を付け、RCS フォーマットで保存されます。一般に、RCS フォーマット（,v）のファイルは、テキスト・ファイルごとに 1 つあります。バイナリ・ファイルは、*filename*,d という名前の独自のディレクトリに完全な内容で保存されます。ユーザがファイルを保存する際に選んだ PERFORCE ファイル・タイプに応じて、各 *filename*,d ディレクトリに1つ以上のアーカイブされたバイナリ・ファイルが保存されます。1 つの *filename*,d ディレクトリに複数のファイルがある場合には、ディレクトリ内の各ファイルが同一のバイナリ・ファイルの異なるリビジョンを参照し、1.*n* という名前が付けられます。*n* はリビジョン番号です。

PERFORCE は Macintosh AppleSingle ファイル・フォーマットもサポートしています。これらのファイルは他のバイナリ・ファイル同様、サーバ上に圧縮されてフル・ファイル保存されます。これらは Mac の AppleSingle ファイル・フォーマットで保存されます。必要なら、これらのファ

イルはサーバ・ルートから直接コピーし、圧縮されていない状態で、そのまま Macintosh 上で使用できます。

PERFORCE はディポ・ファイルで圧縮を使用するので、システム管理者はバックアップ媒体のサイズを決めるとき、データの圧縮性をあてにしてはいけません。テキスト・ファイルもバイナリ・ファイルも、PERFORCE サーバによって圧縮されてから（拡張子 `.gz`）保存されるか、圧縮されずに保存されます。通常、ディポのサブディレクトリにバイナリ・ファイルが圧縮されずに保存される場合、そのバイナリ・ファイルは圧縮不可能なファイルだと考えられます（例えば、圧縮フォーマットで保存されている画像、ビデオ・ストリームなど）。

チェックポイント作成後のバックアップ

クラッシュによるリストア後、バージョン化ファイルが確実にデータベースの情報をすべて反映しているようにするためには、`db.*` ファイルを、少なくともバージョン化ファイルと同等またはそれ以上の古さのチェックポイントからリストアしなければなりません。このため、ディポのディレクトリに保存されているバージョン化ファイルのバックアップは、チェックポイント作成後に行ってください。

バージョン化ファイルは、チェックポイントに保存されているデータより新しくてもよいのですが、そのタイミングのずれは最小限にとどめた方がよいでしょう。一般に、バックアップ・スクリプトによりチェックポイント作成直後にバージョン化ファイルをバックアップする必要があります。

バックアップの手順

PERFORCE サーバをバックアップするには、夜間のバックアップ作業の一環として次の操作を行ってください。

1. サーバの完全性を確認し、新しいファイルの MD5 ダイジェストおよびファイル長メタデータを追加します。

   ```
   p4 verify //...
   ```

   `p4 verify` は `-q` (quiet) オプションで実行することもできます。`-q` オプションを付けると、`p4 verify` はエラーが検出されたときにのみアウトプットを出します。

   `p4 verify` は、アーカイブされたすべてのファイルの MD5 署名を確認し、最初にファイルが保存されたときのものと比較します。また、PERFORCE で認識されているすべてのファイルがディポのサブディレクトリに存在することも確認します。

   バックアップ前に `p4 verify` を実行することにより、前回バックアップ以降のディポの新しいファイルのチェックサムやファイル長メタデータを確実に作成して保存し、その情報を続くバックアップの一環として確実に保存することができます。

   `p4 verify` を日常的に使用することをお勧めします。バックアップ前にサーバの破損が発見できるだけでなく、クラッシュが起きたときに、バックアップからリストアされたファイルの状態を検出することもできるからです。

   > **注** *サイトがきわめて大きい場合には、`p4 verify` は実行に時間がかかる可能性があります。さらに、`p4 verify` の実行中はデータベースがロックされるため、他のほとんどの PERFORCE コマンドが使用できなくなります。大規模なサイトの管理者は、`p4 verify` を毎晩ではなく週1回実行するよう選択してもよいかもしれません。*
   >
   > *`p4 verify` コマンドに関する詳細は、46 ページの「署名によるファイルの認証」をご覧ください。*

2. `-jc` (journal-create) オプションで `p4d` を呼び出すか、あるいは `p4 admin` コマンドを用いることにより、チェックポイントを作成します。

```
p4d -jc
```

または

```
p4 admin checkpoint
```

`p4d` はチェックポイントを作成するときにデータベース全体をロックするので、通常はバックアップの手順の途中で PERFORCE サーバを停止する必要はありません。

> **注** *サイトがきわめて大きい場合（例えば、`db.*` ファイルが数 GB に及ぶ場合）には、チェックポイントの作成にかなり時間がかかるでしょう。*
>
> *そのような場合には、チェックポイントの作成とジャーナルの更新をシステムの活動が低下するときまで延ばしたいと思うかもしれません。例えば、夜間バックアップでは、`journal` ファイルをアーカイブするだけにし、チェックポイントの作成とジャーナルの更新は週 1 度行うようにしてもよいでしょう。*

3. ファイルをバックアップする前にチェックポイントが順調に作成されたことを確認します。（ディスク・クラッシュが起きてから、実は 3 週間前からチェックポイントが正しく作成されていなかった！――ということにならないようにしたいです。）

チェックポイント・コマンドが順調に実行されたことは `p4d -jc` から返ってくるエラー・コードを調べるか、カレント・ジャーナルが更新されたことを確認することによってわかります。

4. チェックポイントが順調に作成された後、そのチェックポイントのファイルとそれまでのジャーナルのファイル、それにバージョン化ファイルをバックアップします。（ほとんどの場合、現実にはジャーナルをバックアップする必要はありません。しかし、一般的にはジャーナルをバックアップしておくことをお勧めします。）

> **注** *まれに、チェックポイントが作られてからディポ・ファイルがバックアップ・ユーティリティによってバックアップされるまでの間に（例えば、ユーザがバックアップ中にファイルを消したり、ファイル・バックアップのプロセス中に Windows 上のファイルをサブミットしたりして）ディポ・ファイルが変更されることがあります。*
>
> *ほとんどのサイトはこうした問題の影響を受けません。したがって、上記のようなリスクは皆無ではないにしても、サーバを 1 日 24 時間 / 週 7 日、連続稼働の状態にしておくことをお勧めします。システムの活動レベルが低下する時間帯にバックアップが行われる場合には、特にそうです。*
>
> *ただし、毎回のバックアップの信頼性を最重視するならば、チェックポイント作成前に PERFORCE サーバを停止して、バックアップ・プロセス完了後に再起動することを検討してください。そうすれば、バックアップ・プロセス中にシステムの状態が変化する危険性を完全に排除することができます。*

`db.*` ファイルをバックアップする必要はありません。最新のチェックポイントとジャーナルに、それらを再生するのに必要な情報はすべて含まれています。さらに重要なのは、`db.*` ファイルからリストアされたデータベースは、チェックポイントからリストアされ

るデータベースのようにトランザクション上の完全性が保たれているとは限らないことです。

Windows

Windows では、PERFORCE サーバ動作中にシステムをバックアップする場合、バックアップ・プログラムが `db.*` *ファイルのバックアップをしないようにしておく必要があります。*

動作中のサーバで `db.*` *ファイルをバックアップしようとすると、Windows はバックアップ・プログラムがバックアップを終えるまで、それらのファイルをロックします。この間、PERFORCE サーバはファイルにアクセスできなくなります。ユーザがファイルのアップデートの操作をしようとしても、サーバは受けつけないでしょう。*

`db.*` *ファイルをバックアップの対象から除外できないソフトウェアをお使いの場合は、バックアップの前に* `p4 admin stop` *でサーバを停止し、バックアップ後に再起動してください。*

リカバリの手順

データベース・ファイルがディスク・エラー、ディスク・クラッシュ等のハードウェア故障のいずれの原因にせよ、破損したり、失われたりした場合には、データベースは保存されているチェックポイントとジャーナルで再生することができます。

システムは様々なかたちで壊れます。このマニュアルでそのあらゆるケースを扱うことはできませんが、少なくとも次の 2 つの代表的なケースを想定して、基本的なリカバリの方針を示します。

- PERFORCE データベースだけが破損し、バージョン化ファイルは壊れていない場合
- データベースもバージョン化ファイルも破損した場合

これらの場合のリカバリの手順は少し異なりますので、以下にの 2 つの項目で別々に論じることにします。

データベースまたはバージョン化ファイルに破損の疑いがある場合には、PERFORCE のテクニカル・サポートまでご連絡ください。

データベースが破損し、バージョン化ファイルは無事の場合

データベースだけが破損した場合（例えば、`db.*` ファイルを納めていたドライブがクラッシュしたが、バージョン化ファイルはシンボリック・リンクを用いて別の物理ドライブに保存していた場合）には、データベースだけを再生すれば十分です。

*必要な*もの

- 最後に作成したチェックポイント・ファイル。最新の P4ROOT ディレクトリのバックアップから入手可能です。
- カレント・ジャーナル・ファイル。P4ROOT ディレクトリとは別のファイルシステムにあるため、P4ROOT ディレクトリが保持されていたファイルシステムに対する障害の影響は受けていないはずです。

*必要ない*もの

- バージョン化ファイルのバックアップ。クラッシュの影響を受けていなければ、すでに最新の状態になっています。

データベースをリカバリするには

1. p4d サーバを停止します。
   ```
   p4 admin stop
   ```
   (p4 admin は PERFORCE スーパー・ユーザしか使えません。)

2. データベース (db.*) ファイルをリネーム (または移動) します。
   ```
   mv your_root_dir/db.* /tmp
   ```
   チェックポイントからリカバリを行った場合は db.* ファイルが $P4ROOT ディレクトリに存在しない可能性があります。古い db.* ファイルをリカバリプロセスで使用することはありませんが、リストアに成功したことが確認できるまで削除しないことをお勧めします。

3. -jr (journal-restore) オプションで最後に作成したチェックポイントとカレント・ジャーナルを指定し p4d を起動します。サーバ・ルート ($P4ROOT) を明示的に指定する場合は、-jr オプションの前に引数 -r $P4ROOT を付けます。
   ```
   p4d -r $P4ROOT -jr checkpoint_file journal_file
   ```
   これでデータベースは最後のチェックポイント作成時点の状態にリカバリし、その上でそのチェックポイント作成後のジャーナル・ファイルに記録された変更が適用されます。

> **注** *チェックポイント作成時に -z (圧縮) オプションを指定してチェックポイントを圧縮した場合には、圧縮されていないジャーナル・ファイルを圧縮されたチェックポイントと別にリストアする必要があります。*
>
> *すなわち、コマンド*
>
> `p4d -r $P4ROOT -jr checkpoint_file journal_file`
>
> *ではなく、次の 2 つのコマンドを使います。*
>
> ```
> p4d -r $P4ROOT -z -jr checkpoint_file.gz
> p4d -r $P4ROOT -jr journal_file
> ```
>
> *-z オプションを使うときには、拡張子 .gz を明示的に指定し、-jr オプションの前に引数 -r $P4ROOT を入れるのも忘れないでください。*

システムをチェックする

リストアは完了しました。35 ページの「リストア後のシステムの完全性を確認する」を参照し、リストアが成功したことを確認してください。

システムの状態

最新のチェックポイントからリカバリしたデータベースは、カレント・ジャーナルに保存されていた変更を適用すると、故障発生時点の最新の状態に戻ります。

リカバリ後はデータベースもバージョン化ファイルも、クラッシュするまでのすべての変更が反映されています。失われたデータはないはずです。

データベースとバージョン化ファイルの両方が失われるか破損した場合

データベースとバージョン化ファイルがともに破損した場合には、その両方をリストアする必要があり、バージョン化ファイルがリストアされたデータベースより古くないことも確認しておく必要があります。

*必要な*もの

- 最後に作成したチェックポイント・ファイル。最新の P4ROOT ディレクトリのバックアップから入手可能です。
- バージョン化ファイル。最新の P4ROOT ディレクトリのバックアップから入手可能なです。

*必要ない*もの

- カレント・ジャーナル・ファイル。

  ジャーナルは、最後のバックアップからクラッシュまでにメタデータとバージョン化ファイルに加えられた変更を記録しています。ここまではクラッシュの*前に行われた*バックアップからバージョン化ファイルをリストアしようとしているので、チェックポイントだけがリカバリに有用なメタデータを含み、ジャーナルの情報はほとんど役立ちません。

データベースをリカバリするには

1. p4d サーバを停止します。

   ```
   p4 admin stop
   ```

   (p4 admin は PERFORCE スーパー・ユーザしか使えません。)

2. 破損したデータベース (db.)・ファイルをリネーム (または移動) します。

   ```
   mv your_root_dir/db.* /tmp
   ```

   実際には、破損した db.* ファイルをリストアプロセスで使用することはありませんが、念のため、リストアに成功したことが確認できるまで削除しないことをお勧めします。

3. -jr (journal-restore) オプションに最新のチェックポイントのみを指定して p4d を起動します。

   ```
   p4d -r $P4ROOT -jr checkpoint_file
   ```

   これでデータベースは最後のチェックポイント作成時点の状態にリカバリしますが、ジャーナル・ファイルに記録されている変更は適用されません。(引数 -r $P4ROOT は、-jr オプションの前に指定しなければなりません。)

   ジャーナル・ファイルに記録された変更を適用せずにリカバリすると、データベースは最後にバックアップした時点の状態に戻ります。このケースでは、リストアされるバージョン化ファイルには、最後のチェックポイント作成時点までのディポの状態しか反映されないので、ジャーナル・ファイルに記録されている変更を適用する必要はありません。

バージョン化ファイルをリカバリするには

4. データベースをリカバリしたら、次に、システムのリストアの手順 (例えば UNIX の restore(1) コマンド) に従ってバージョン化ファイルをリストアする必要があります。そのバージョン化ファイルがデータベースより古くないことを確認してください。

システムをチェックする

リストアは完了しました。35 ページの「リストア後のシステムの完全性を確認する」を参照し、リストアが成功したことを確認してください。

最後のシステム・バックアップからディスク・クラッシュまでの間にディポにサブミットされたファイルは、リストア後のディポには存在しません。

注 *(ディポにサブミット済だがバックアップされていなかった) "新しい" ファイル はリストア後のディポに残っていませんが、そのようなファイルは 1 人以上のユーザがクライアント・ワークスペースに最新版のコピーを保存している可能性があります (いや、きっと保存しています!)。*

そのようなファイルを見つけるには、次のように PERFORCE コマンドを使ってユーザにクライアント・ワークスペースのファイルとディポのファイルの違いを調べてもらいます。コマンド

```
p4 diff -se
```

を実行したら、ワークスペースのファイルの中で PERFORCE が自己のシステムのファイルと認識しないファイルのリストが表示されます。そのファイルがまさにリストアしようとしているものであることを確認したら、ユーザにそのファイルを `edit` *で作業状態にしてもらい、チェンジリストとしてディポにサブミットしてもらうとよいでしょう。*

システムの状態

リストア後のディポ・ディレクトリには、最新のバージョン化ファイルは含まれていない（すなわち、最後のバックアップからディスク・クラッシュまでの間にサブミットされたファイルはサーバから失われている）可能性があります。

- 通常、そのようなファイルの最新リビジョンは、ユーザのクライアント・ワークスペースに残っているコピーからリストアすることができます。
- バージョン化ファイル*だけ*が失われた場合（つまり、データベースは別のディスクに保存されていてクラッシュの影響を受け*なかったような*場合）には、データベースのコピーを作成してジャーナルをそれに適用し、最近のチェンジリストを調べることによって、最後のバックアップからディスク・クラッシュまでの間にサブミットされたファイルを突きとめることができます。

いずれにせよ、詳細については PERFORCE テクニカル・サポートまでご連絡ください。

リストア後のシステムの完全性を確認する

リストア後は `p4 verify` を実行し、バージョン化ファイルがデータベースより古くないことを確認しておくとよいでしょう。

```
p4 verify -q //...
```

このコマンドで、バージョン化ファイルの完全性を確認します。`-q` (quiet) オプションが付いているのでアウトプットはエラー検出時のみで、アウトプットがないのが理想です。

`p4 verify` コマンドでバージョン化ファイルが `MISSING` として検出された場合は、リストアされなかったファイルに関する情報がデータベースにあることを意味します。通常、この現象は、バージョン化ファイルのバックアップ後に作成されたチェックポイントとジャーナルからリストアが行われる（つまり、バージョン化ファイルの方がデータベースより古い）ことが原因で発生します。

バックアップ・ルーチンの一環として、(お勧めしたように) `p4 verify` を用いて保存していれば、リストア後にサーバ上で `p4 verify` を実行し、リストアが成功したことを再確認することができます。

クラッシュ後のシステムリストアがうまくいかない場合には、PERFORCE テクニカル・サポートに連絡し、サポートをご依頼ください。

第 3 章

PERFORCE の管理 : スーパー・ユーザのタスク

本章では、PERFORCE の日常管理に関わる基本的なタスクのほか、クロスプラットフォーム開発課題、PERFORCE サーバのマシン間移行、およびリモート / ローカル・ディポの操作に関連する PERFORCE の高度な設定 について説明します。

本章で説明する作業のほとんどは、PERFORCE プロテクション・テーブルに定義された PERFORCE スーパー・ユーザ (アクセス・レベル `super`) または管理者 (アクセス・レベル `admin`) の権限を必要とします。PERFORCE スーパー・ユーザまたは管理者のアクセス管理、およびプロテクション全般については 69 ページの「PERFORCE の管理 : プロテクション」をご覧ください。

PERFORCE のリリース 2004.2 で、パスワード強度要求および認証方法ポリシーを管理する、新しい認証機構とサーバ構成可能なセキュリティ設定が導入されました。詳細については、37 ページの「認証方法 : パスワードとチケット」と 39 ページの「サーバ・セキュリティ・レベル」を参照してください。

PERFORCE 管理の基本

ここでは、PERFORCE スーパー・ユーザまたは管理者が日常的に行う次のような作業を説明します。

- ユーザメンテナンス作業 : パスワードの再設定、新規ユーザの作成、新規ユーザの自動作成の防止、旧ユーザが作業状態のままにしているファイルの削除。
- システム管理 : サーバ・セキュリティ・レベルの設定、ファイル削除によるディスク容量の再生、サブミット済チェンジリストの編集、サーバの完全性の確認、ファイルタイプの定義による PERFORCE のファイルタイプ検出機構の制御、`-f` オプションによる強制操作。

認証方法 : パスワードとチケット

PERFORCE では、パスワード・ベース認証とチケット・ベース認証の 2 つの認証方法がサポートされています。

> *チケット・ベース認証は、パスワード・ベース認証より安全な認証機構ですが、クライアントのワークステーションと PERFORCE サーバの間のネットワーク・トラフィックを暗号化しません。*
>
> *安全でないネットワークを経由して PERFORCE にアクセスする場合は、選択した認証方法にかかわらず、サード・パーティのトンネリング・ソリューション (ssh や VPN など) を使用してください。*

パスワード・ベース認証のしくみ

パスワード・ベース認証は、ステートレスです。つまり、パスワードを正しく設定した後は、無期限のアクセス権限が与えられます。リリース 2004.2 より前は、パスワード・ベース認証機構によってパスワードの強度または存在の要求が強要されることはありませんでした。

リリース 2004.2 で導入されたサーバ・セキュリティ・レベルの概念によって、管理者はパスワードの強度および存在の要求を強要できます。詳細については、39 ページの「サーバ・セキュリティ・レベル」をご覧ください。

パスワード・ベース認証は、レベル 0、1、および 2 でサポートされます。

チケット・ベース認証のしくみ

チケットベース認証は、時間制限のあるチケットにもとづいています。ユーザは、このチケットによって、PERFORCE サーバに接続できます。チケットは `P4TICKETS` 環境変数で指定したファイルに保存されます。この変数が設定されていない場合、Windows では `%USERPROFILE%\p4tickets.txt`、UNIX とその他の OS では `$HOME/.p4tickets` にチケットが保存されます。チケットは、リリース 2004.2 以降の PERFORCE クライアント・プログラムによって自動的に管理されます。

すべてのチケットの有効期間は有限で、この期間が終了した後はチケットが無効になります。デフォルトでは、チケットは 12 時間 (43200 秒) 有効です。ユーザのグループごとに異なるチケットの有効期間を設定するには、各グループの `p4 group` フォームの `Timeout:` フィールドを編集します。複数のグループに属するユーザのタイムアウト値は、そのユーザが属するすべてのグループの タイムアウト値のうちの最大の値です。無期限のチケットを作成するには、[`Timeout:`] を `0` に設定します。

チケットはパスワードではありませんが、ユーザが PERFORCE パスワードを指定できれば、PERFORCE サーバは有効なチケットを受け入れます。この動作によって、チケット・ベース認証のセキュリティ上の利点に、パスワード・ベース認証によって提供されるスクリプト作成の容易さが加わります。チケット・ベース認証は、すべてのサーバ・セキュリティ・レベルでサポートされ、セキュリティ・レベル 3 では必須です。

PERFORCE へのログイン

チケット・ベース認証を使用するには、`p4 login` コマンドを使用してログインして、チケットを取得します。

```
p4 login
```

ユーザはパスワードの入力を求められ、そのユーザのチケット・ファイルにチケットが自動的に作成されます。ログイン中に `p4 login` を実行して、チケットの有効期間を延長できます。ログイン中に p4 login を実行すると、チケットの有効期間は初期タイムアウト設定値の 1/3 だけ延長されます (最大でチケットの初期タイムアウト設定値まで)。

デフォルトでは、PERFORCE チケットはそのユーザの IP アドレスに対してのみ有効です。複数のマシン上で使用される共有ホーム・ディレクトリがある場合は、以下のコマンドを使用すると、どのマシンからでも PERFORCE にログインできます。

```
p4 login -a
```

このコマンドは、すべての IP アドレスから使用できるホーム・ディレクトリにチケットを作成します。

PERFORCE からのログアウト

チケットを削除して 1 つのマシン上で PERFORCE からログアウトするには、以下のコマンドを使用します。

```
p4 logout
```

チケット・ファイルのエントリが削除されます。同じ PERFORCE サーバに対して有効なチケットが複数あり、それらのチケットが他のマシン上にある場合は、他のマシン上のチケットは削除されません (それらのマシン上でログインしたままになります)。

複数のマシンから PERFORCE にログインしている場合、以下のコマンドを使用して、ログインしていたすべてのマシンで PERFORCE からログアウトすることができます。

```
p4 logout -a
```

これにより、すべての PERFORCE チケットが無効化され、ユーザはログアウトします。

チケット・ステータスの設定

現在のチケット (つまり、IP アドレス、ユーザ名、および P4PORT の設定) がまだ有効であるかどうかを確認するには、以下のコマンドを使用します。

```
p4 login -s
```

チケットが有効である場合は、有効期間の残り時間が表示されます。

ユーザが現在持っているすべてのチケットを表示するには、以下のコマンドを使用します。

```
p4 tickets
```

チケット・ファイルの内容が表示されます。

サーバ・セキュリティ・レベル

PERFORCE スーパー・ユーザは、security カウンタを設定して、サーバ全体にわたるパスワード使用要求、パスワード強度の強要、およびサポートされているユーザ / サーバ認証の方法を設定できます。security カウンタを変更するには、以下のコマンドを実行します。

```
p4 counter -f security seclevel
```

この場合、*seclevel* は 0、1、2、または 3 です。カウンタを設定した後、サーバを再起動します。

サーバ・セキュリティ・レベルの選択

デフォルト のセキュリ ティ・ レベルは 0 (パスワード は不要、パスワード 強度は強要されない) です。

すべてのユーザがパスワードを持つようにするには、セキュリティ・レベル 1 を使用します。旧リリースのクライアント・プログラムのユーザは引き続き弱いパスワードを入力することができます。

すべてのユーザが強いパスワードを持つようにするには、セキュリティ・レベル 2 を使用します。旧リリースの PERFORCE ソフトウェアは引き続き動作しますが、旧リリースの PERFORCE クライアント・ソフトウェアのユーザは、リリース 2003.2 またはそれ以降の PERFORCE クライアント・プログラムを使用してパスワードを強いパスワードに変更する必要があります。

すべてのユーザに強いパスワードを持つよう要求する場合やセッション・ベースの認証の使用を要求する場合は、セキュリティ・レベル 3 と最新の PERFORCE クライアント・ソフトウェアを使用します。

レベル 0 は、2003.2 より前のリリースのサーバの動作に相当します。レベル 1 および 2 は、以前のクライアント・ソフトウェアのサポートを目的として設計されました。レベル 3 は、最も高度なセキュリティを提供します。

PERFORCE サーバ・セキュリティ・レベルと、PERFORCE クライアント・プログラムの動作に対するその影響を以下に説明します。

	サーバの動作
0 （または未設定）	以前のクライアントをサポート。パスワードは不要です。パスワードを使用する場合、パスワード強度は強要されません。 パスワードを持つユーザは、チケット・ベース認証のために P4PASSWD 設定または p4 login コマンドを使用することができます。 旧リリースの PERFORCE クライアント・プログラムのユーザは影響を受けません。
1	リリース 2003.2 以降の PERFORCE クライアント・プログラムのユーザの場合は強いパスワードが必要ですが、既存のパスワードは再設定されません。 リリース 2003.2 より前の PERFORCE クライアント・プログラムは、p4 passwd を使用してパスワードを設定するか、p4 user フォームにパスワードを設定することができますが、パスワード強度は強要されません。 パスワードを持つユーザは、チケット・ベース認証のために P4PASSWD 設定または p4 login コマンドを使用することができます。
2	すべての未検証の強度のパスワードを変更する必要があります。 リリース 2003.2 より前のクライアント・プログラムのユーザはパスワードを設定できません。 リリース 2003.2 以降のクライアント・プログラムのユーザは、p4 passwd を使用して、プロンプトでパスワードを入力する必要があります。p4 user フォームまたは p4 passwd -O *oldpass* -P newpass コマンドによるパスワードの設定は禁止されます。 Windows では、パスワードがレジストリに保存されることも、レジストリから読み取られることもなくなります（P4PASSWD を環境変数として保存することはできますが、p4 set P4PASSWD で設定されたパスワードは無視されます）。 リリース 2003.2 以降の PERFORCE クライアント・プログラムで強いパスワードを設定しているユーザは、P4PASSWD 設定（ パスワード・ベース認証の場合 ）または p4 login コマンド（チケットベース認証の場合）を使用することができます。
3	すべてのパスワード・ベース認証が拒否されます。 ユーザは、チケット・ベース認証を使用する必要があります（p4 login）。 パスワードに依存するスクリプトがある場合は、p4 login を使用して、そのスクリプトを実行するユーザに対して有効なチケットを作成するか、p4 login -p を使用して、パスワードと同じように PERFORCE コマンドに渡すことのできる（つまり、コマンドラインから渡すか、または P4PASSWD を有効なチケットの値に設定して渡すことのできる）チケットの値を表示します。

パスワードの強度

サーバのセキュリティ・レベルと PERFORCE クライアント・ソフトウェア・リリースの特定の組み合わせでは、ユーザは「強い」パスワードを設定することを求められます。パスワードが 8 文字以上の長さで、以下の記述のうちの少なくとも 2 つが真である場合に、そのパスワードは強いと見なされます。

- パスワードに大文字が含まれている
- パスワードに小文字が含まれている
- パスワードにアルファベット以外の文字が含まれている

例えば、a1b2c3d4、A1B2C3D4、aBcDeFgH などのパスワードは強いと見なされます。

ユーザ・パスワードの再設定

PERFORCE スーパー・ユーザは、次のコマンドを使用して PERFORCE ユーザのパスワードを再設定することができます。

- リリース 99.1 以降

  ```
  p4 passwd username
  ```

  入力を要求されたら、新しいユーザ用パスワード *username* を入力します。

- 99.1 より前のリリース

  ```
  p4 user -f username
  ```

  ユーザ仕様フォームの `Password:` フィールドにパスワードを入力します。

ユーザの作成

デフォルトでは、存在しないユーザ名によってコマンドが発行されるたびにデータベースに新規ユーザ・レコードが作成されます。また、PERFORCE スーパー・ユーザは次のように `-f` (force) オプションを使って新規ユーザを生成することができます。

```
p4 user -f username
```

フォーム・フィールドに作成したいユーザに関する情報を入力します。

`p4 user` コマンドには、フォーム・エディタを使わずに標準入力できるオプション（`-i`）もあります。多数のユーザをすばやく作成するには、ユーザ・データを読み取り、`p4 user` フォームによって使用されるフォーマットで出力を生成し、生成された各フォームを `p4 user -i -f` にパイプで送るスクリプトを作成します。

ユーザ（自動）作成の防止

デフォルトでは、PERFORCE サーバは存在しないユーザによってコマンドが発行されるたびに新規ユーザ・レコードを作成します。

PERFORCE グループを作成し、権限を持つユーザ全員を 1 つのグループに割り当て、権限を持つユーザのグループに属さないユーザには書き込み権限を認めないようにすれば、最も簡単にユーザの自動作成を防げます。

例 : *グループ内のユーザを設定する*

PERFORCE スーパー・ユーザが、サーバに新しいユーザを作らせないようにしたいと考えます。そこで、次のように `p4 users` *を実行し、現時点で自サイトにいる 3 人のユーザを登録します。*

```
p4 group p4users
```

表示されるフォームには、次のように記述します。

```
# A Perforce Group Specification.
# Group:      The name of the group.
# MaxResults: Limits the rows (or 'unlimited') any one operation
#             can return to the client. See 'p4 help maxresults'.
# MaxScanRows:Limits the rows (or 'unlimited') any one operation
#             can scan from any one database table...
# MaxLockTime:Limits the time (in milliseconds, or 'unlimited')
#             any one operation can lock any database table when
#             scanning data...
# Timeout:    Time in seconds which determines how long a 'p4 login'
#             session ticket remains valid (default is 12 hours).
# Subgroups:  Other groups automatically included in this group.
# Users:      The users in the group.  One per line.
Group:  p4users
MaxResults:     unlimited
MaxScanRows:    unlimited
MaxLockTime:    unlimited
Timeout:        43200
Subgroups:
Users:
        edk
        lisag
```

さらに `p4 protect` *を実行し、プロテクション・テーブルを編集します。デフォルト時、テーブルの書き込み権限に関する行は次のように表示されます。*

```
write user * * //...
```

*これは、あらゆるユーザ（最初の *）に対して、あらゆるホスト（2 番目の *）から、ディポ内のすべての領域（//... が示すファイル）への書き込み権限を認めています。*

最後に、スーパーユーザは `p4 protect` *を使ってプロテクション・テーブル内のこの行を次のように変更します。*

```
write group p4users * //...
```

これで、 `write` *アクセスは Perforce グループ* `p4users` *のユーザだけに限定されるようになりました。* `p4users` *のメンバーは、あらゆるホスト（*）から Perforce を使用することができ、ディポ内のすべての領域（//...）に対して書き込みを行うことができます。*

プロテクション・テーブルに別の行を追加しない限り、すべてのユーザ（今では `p4 protect` *の中で定義されています）に権限が認められており（または認められておらず）、新しいユーザが Perforce の使用を試みようとしても、サーバは自動的に新しいユーザを作成することはありません。*

プロテクションのさらに詳しい取り扱いについては 69 ページの「PERFORCE の管理 : プロテクション」をご覧ください。

旧ユーザの削除

ユーザはそれぞれ PERFORCE のライセンスを 1 つずつ使用しています。PERFORCE 管理者は、次のコマンドを使ってユーザを削除し、空きのライセンスを作ることができます。

```
p4 user -d username
```

ユーザを削除する前に、チェンジリスト上でユーザが作業状態にしているファイルを取り消す（またはサブミットする）必要があります。ファイルを作業状態にしているユーザを削除しようとすると、PERFORCE はその旨のエラー・メッセージを表示します。

ユーザを削除すると PERFORCE のライセンスが解放されますが、グループやプロテクション・テーブルは自動的には更新されません。`p4 group` または `p4 protect` を使用してこれらのテーブルからユーザを削除してください。

新規ライセンス許可ユーザの追加

PERFORCE のライセンスは、`license` というファイルにより制御されています。このファイルはサーバのルート・ディレクトリに格納されています。

ライセンス・ファイルを追加または更新するには、PERFORCE サーバを停止し、`license` ファイルをサーバのルート・ディレクトリにコピーして PERFORCE サーバを再起動します。

リリース 2006.2 では、`p4 license -i` を使用して標準入力から新しいライセンス・ファイルを読み込むことにより、PERFORCE サーバを停止することなく既存の `license` ファイルを更新することができます。

PERFORCE から取得した新しいライセンス・ファイルのほとんどは、サーバの IP アドレスが変更されている場合を除き、`p4 license` を使用してインストール可能です。サーバの IP アドレスが変更されている場合、または現時点で `license` ファイルが存在しない場合は、PERFORCE サーバを停止し、ライセンス・ファイルを手動で所定の場所にコピーし、サーバを再起動する必要があります。

旧ユーザによる作業中のファイルの取り消し

不在または廃止されたユーザ（例えば退職者）によって作業状態のままになっているファイルを元に戻すには、作業状態のクライアント・ワークスペース仕様を PERFORCE 管理者が削除します。

例えば、`p4 opened` のアウトプットが次のように表示されたとき、

```
//depot/main/code/file.c#8 - edit default change (txt) by
jim@stlouis
```

クライアント・ワークスペース仕様 “`stlouis`” は次のコマンドで削除できます。

```
p4 client -d -f stlouis
```

クライアント・ワークスペース仕様を削除すると、そのワークスペースで作業状態にあるファイルもすべて取り消され、ワークスペースに関連付けられた作業中チェンジリストと、ワークスペースに関連付けられた修正レコードが削除されます。クライアント・ワークスペース仕様を削除しても、ワークスペースの所有者が実際に使っていたワークスペースのファイルには影響*しません*。それらのファイルは引き続き他のユーザによってアクセス可能です。

ファイル削除によるディスク容量の再生

> `p4 obliterate` *は注意して使ってください。これは PERFORCE の中で実際にファイルのデータを削除する唯一のコマンドです。*

ディポは常に拡張しています。これは必ずしも望ましいことではありません。ユーザが不注意なブランチやサブミットによって数百の不要なファイルを作成していたり、使用されなくなった古いファイルのディレクトリがあることもあります。`p4 delete` は単に最新リビジョン内でファイルを削除扱いにするだけなので、サーバのディスク容量に空きができるわけではありません。

ディスク容量に空きを作るには、PERFORCE 管理者が `p4 obliterate` を使います。`p4 obliterate` ***filename*** は、ディポからあるファイルのすべての履歴を削除し、それを最初から存在しなかったも同然にすることができます。

注 *ソフトウェア構成管理システムの目的は、どのファイルにどのような操作が行われたかという履歴をサイトに保持させることにあります。*`p4 obliterate` *コマンドはこの目的に反します。このコマンドは、本来、ディポ内に存在しなくなったファイルの情報をデータベースから削除することだけを目的としたコマンドであり、通常のソフトウェア開発プロセスの一部として使用するためのコマンドではありません。*

`p4 obliterate` *がコンピュータの能力のかなりの部分を使用することにもご注意ください。ファイルを削除するには、メタデータ全体をファイル引数ごとにスキャンする必要があります。コンピュータの使用がピークに達する期間に、*`p4 obliterate` *を実行することは避けてください。*

デフォルトでは、`p4 obliterate` ***filename*** は何もしません。何をするかを表示するだけです。実際にファイルを破壊するには、`p4 obliterate -y` ***filename*** を使います。

あるファイルの 1 つのリビジョンだけを破壊するには、コマンドラインに消したいリビジョンの番号（例えばリビジョン 5）だけを指定します。

```
p4 obliterate -y file#5
```

リビジョン範囲の指定も可能です。あるファイルのリビジョン 5 ～ 7 を破壊するには、次のように指定します。

```
p4 obliterate -y file#5,7
```

> *リビジョンを指定するときには、指定に間違いがないかどうかを確かめてください。リビジョンを指定していないと、**すべてのリビジョンが消去されます!***
>
> `p4 obliterate` *を使うときは注意が必要です。ファイルとリビジョンの指定に間違いがないことを確認するまでは、必ず* `-y` *オプション**なしで**使ってください。*

`p4 obliterate` コマンドにはもう 1 つ、`-z` オプションもあります。ディポの 1 つの領域から別の領域へファイルのブランチを作成するときには“怠惰なコピー”が作成されます。ファイルそのものはコピーされず、ブランチ作成だけが記録されます。事情により“怠惰なコピー”を取り消し、ディポのサブディレクトリにブランチしたファイルの新しいコピーを作成する場合は、`p4 obliterate -y -z filename` で怠惰なコピーを“消去”し、新しいコピーを作成できます。

`-z` オプションを使って怠惰なコピーを削除すると、一般的に使用可能なディスク容量が*増えます*。削除したファイルを指しているいずれのメタデータも壊すことなく、アーカイブ・ファイルを手動で削除できるようにするということが、一般的には `p4 obliterate -y -z` の唯一の使用目的です。

ファイルにアクセスしようすると、次のエラー・メッセージが表示されることがあります。

```
Operation:user-sync
Librarian checkout path failed
```

この場合の *path* は事前に消去されたファイルのパスです。以前に古い PERFORCE サーバ (98.2/10314 より前のリリース) で `p4 obliterate` を使用していたことが原因と思われます。PERFORCE テクニカル・サポートまで対策をご相談ください。

チェンジリストの削除とチェンジリストの記述の編集

`p4 change` に `-f` (force) オプションを付けると、PERFORCE 管理者はサブミット済チェンジリストの記述、日付、ユーザ名を変更できます。`p4 change -f changenumber` と記述します。このコマンドにより標準のチェンジリストのフォームが表示されるだけでなく、スーパーユーザが、チェンジリストの時刻、コメント、日付および関連するユーザ名を編集できます。

`-f` オプションは `p4 obliterate` でファイルを消去されたサブミット済チェンジリストの削除にも使えます。`p4 change -d -f changenumber` と記述します。

例 : *チェンジリスト 123 の更新とチェンジリスト 124 の削除*

`p4 change` *に* `-f` *(force) オプションを付けます。*

```
p4 change -f 123
p4 change -d -f 124
```

チェンジリスト `123` *の* `User:` *と* `Description:` *のフィールドが編集され、チェンジリスト* `124` *は削除されます。*

署名によるファイルの認証

`p4 verify` *`filenames`* コマンドにより、PERFORCE 管理者はファイルの各リビジョンについて保存されている MD5 ダイジェストを検証できます。ユーザがディポにファイルを保存したとき作成された署名は、クラッシュ時にリカバリが正しく行われたことを後で確認するために利用できます。リカバリしたファイルの署名が保存されていた署名と一致すれば、そのファイルは正しくリカバリされたことになります。新しい署名が PERFORCE データベースの該当するファイル・リビジョンの署名と一致しなければ、PERFORCE は署名の後ろに `BAD!` と表示します。

毎晩システム・バックアップを行うときには、事前に `p4 verify` を実行し、不一致がないことを確認してからバックアップに移るとよいでしょう。

`p4 verify` 実行中に `BAD!` が表示されるときは、データベースかバージョン化ファイルが破損している可能性があります。PERFORCE テクニカル・サポートまでご連絡ください。

サーバ・アップグレード中の認証

サーバ・アップグレードの前、およびアップグレード後も以下のように `p4 verify` を実行することをお勧めします。

1. アップグレードの前に、コマンド、
   ```
   p4 verify -q //...
   ```
   を実行し、サーバの完全性を確認します。
2. チェックポイントを取り、そのチェックポイントとバージョン化ファイルを安全な場所にコピーします。
3. サーバのアップグレードを実行します。
4. アップグレード後に、コマンド、
   ```
   p4 verify -q //...
   ```
   を実行し、新しいシステムの完全性を確認します。

p4 typemap でファイルタイプを定義する

デフォルトでは、PERFORCE は自動的にファイルの先頭 8192 バイトを解析し、`text` か `binary` かのタイプを判断します。先頭 8192 バイトのすべてのバイトについて最上位ビットが立っていなければ `text` と見なし、そうでなければ、`binary` と見なします。

このデフォルトの動作は `-t` *`filetype`* オプションを用いてオーバライドすることができますが、ファイルタイプがおおむね（常にではない !）正しく検出される場合、ユーザは容易にこのオプションを指定し忘れてしまいます。RTF（Rich Text Format）や Adobe PDF（Portable Document Format）のように確実なファイル・フォーマットは、一連のコメント・フィールドまたはテキスト形式のデータで始まります。もしコメントがある程度長いと、このようなファイルは PERFORCE サーバによって誤って `text` として検出されます。

`p4 typemap` コマンドにより、システム管理者が PERFORCE のファイルタイプとファイル名の仕様をリンクさせるテーブルを設定できるようになったため、この問題は解決されました。追加されているファイルが設定したタイプマップ・テーブルの入力と一致すると、ファイルタイプがオーバーライドされ、一致しない場合には、PERFORCE クライアント・プログラムがファイルタイプを割り当てます。例えば、すべての PDF ファイルと RTF ファイルを `binary` として処理するには、`p4 typemap` を使用してタイプマップ・テーブルを以下のように修正します。

```
Typemap:
        binary //....pdf
        binary //....rtf
```

最初の 3 個のピリオド（“...”）はルート・ディレクトリ下の全ファイルをマッピングの対象に含める PERFORCE のワイルドカードです。4 個めのピリオドとファイル拡張子は、その指定をファイル名の最後が“.pdf”（または“.rtf”）で終わるファイルに適用する指定です。以下の表に、一般的なファイル拡張子と、それに対する PERFORCE ファイルタイプおよび修飾子を示します。

	PERFORCE ファイルタイプ	説明
.asp	text	アクティブサーバページ・ファイル
.avi	binary+F	Windows ビデオ・ファイル
.bmp	binary	Windows ビットマップ・ファイル
.btr	binary	Btrieve データベース・ファイル
.cnf	text	カンファレンス・リンク・ファイル
.css	text	カスケード・スタイルシート・ファイル
.doc	binary	Microsoft Word ファイル
.dot	binary	Microsoft Word テンプレート
.exp	binary+w	エクスポート・ファイル（Microsoft Visual C++）
.gif	binary+F	GIF 画像ファイル
.htm	text	HTML ファイル
.html	text	HTML ファイル
.ico	binary	アイコン・ファイル
.inc	text	アクティブサーバ・インクルード・ファイル
.ini	text+w	アプリケーション初期設定ファイル
.jpg	binary	JPEG 画像ファイル
.js	text	Java スクリプト言語ソースコード・ファイル
.lib	binary+w	ライブラリファイル（複数のプログラミング言語）
.log	text+w	ログファイル
.mpg	binary+F	MPEG ビデオ・ファイル
.pdf	binary	Adobe PDF ファイル
.pdm	text+w	Sybase Power Designer ファイル
.ppt	binary	Microsoft PowerPoint ファイル
.xls	binary	Microsoft Excel ファイル
.zip	binary+F	ZIP アーカイブファイル

注 *ファイルの内容に日本語（複数バイト文字）を含むテキストファイルは、ファイルタイプを* `text` *ではなく* `unicode` *とする必要があります。*

上記の表で推奨されたすべてのファイル拡張子を PERFORCE のファイルタイプにマッピングするには、次のように `p4 typemap` テーブルを使用します。

```
# PERFORCE File Type Mapping Specifications.
#
#  TypeMap:     a list of filetype mappings; one per line.
#               Each line has two elements:
#               Filetype: The filetype to use on 'p4 add'.
#               Path:     File pattern which will use this filetype.
# See 'p4 help typemap' for more information.

TypeMap:

        text //....asp
        binary+F //....avi
        binary //....bmp
        binary //....btr
        text //....cnf
        text //....css
        binary //....doc
        binary //....dot
        binary+w //....exp
        binary+F //....gif
        text //....htm
        text //....html
        binary //....ico
        text //....inc
        text+w //....ini
        binary //....jpg
        text //....js
        binary+w //....lib
        text+w //....log
        binary+F //....mpg
        binary //....pdf
        text+w //....pdm
        binary //....ppt
        binary //....xls
        binary+F //....zip
```

詳しくは、『PERFORCE コマンド・リファレンス』の `p4 typemap` のページをご覧ください。

p4 typemap でサイト全体の悲観的ロックを実施する

デフォルトでは、PERFORCE は同時並行開発をサポートしていますが、ファイルを編集目的で作業状態にするのは一度に一人のユーザだけであることが想定される環境では、部分ファイルタイプで修飾子 `+l`（排他的作業状態）を使用して悲観的ロックを実施することができます。次のように[`typemap:`]を定義すると、`+l`修飾子がディポ内にあるすべての新規追加されたファイルに対し自動的に適用されます。

```
TypeMap:
        +1 //depot/...
```

このタイプマップを使用すると、タイプマップの更新後、ユーザがディポに追加したファイルには自動的に `+l` 修飾子が適用されるので以降、そのファイルを編集目的で作業状態にできるのは一度に一人のユーザに限られます。タイプマップ・テーブルはディポへの新規追加分にのみ適用されます。したがって、タイプマップ・テーブルをサイト全体で排他的作業状態に更新した後、それ以前に `+l` オプションを付けずにサブミットしたファイルは `p4 edit -t+l` *`filename`* コマンドを使って編集目的で作業状態にし、再度サブミットしなければなりません。同様に、すでにファイルを編集目的で作業状態にしているユーザは、`p4 reopen -t+l` *`filename`* コマンドでファイルタイプを更新しなければなりません。

-f オプションによる強制動作

PERFORCE スーパー・ユーザや管理者は、特定のコマンドに `-f` オプションを指定することによって、一般ユーザは実行できないような特定の操作を強制的に実行できます。PERFORCE 管理者がこのオプションを使用できるのは、`p4 branch`、`p4 change`、`p4 client`、`p4 job`、`p4 label`、`p4 unlock` です。PERFORCE スーパー・ユーザはさらに、`p4 user` にも使用できます。このオプションの用法と意味は次のとおりです。

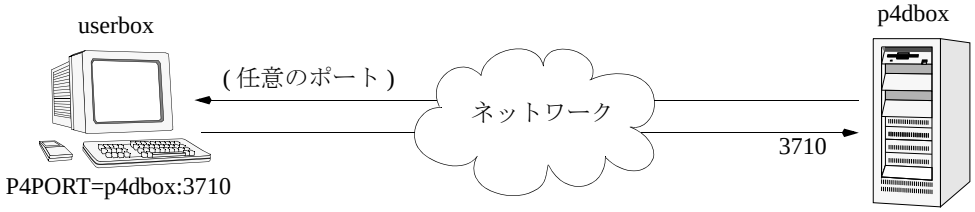
	シンタックス	機能
`p4 branch`	`p4 branch -f branchname`	ブランチ仕様編集時に更新日の変更を可能にします。
		所有権を無視し、ブランチを削除します。
`p4 change`	`p4 change -f [changelist#]`	チェンジリスト仕様編集時に更新日の変更を可能にします。
		対象となるチェンジリストのコメント・フィールドとユーザ名の編集を可能にします。
		対象となる空のチェンジリストを削除します。
`p4 client`	`p4 client -f clientname`	クライアント仕様編集時に更新日の変更を可能にします。
		所有権を無視し、クライアントを削除します。そのクライアントによる作業中のファイルがある場合も含めます。
`p4 job`	`p4 job -f [jobname]`	読み取り専用フィールドの手作業による更新を可能にします。
`p4 label`	`p4 label -f labelname`	ラベル仕様編集時に更新日の変更を可能にします。
		所有権を無視し、ラベルを削除します。
`p4 unlock`	`p4 unlock -c changelist -f file`	所有権を無視し、作業中の番号付チェンジリストにある作業状態のファイルに対して、(`p4 lock` で設定された) ロックを解除します。
`p4 user`	`p4 user -f username`	所有権を無視し、すべてのフィールドの更新を可能にします。このコマンドを実行するには super 権限が必要です。
		所有権を無視し、ユーザを削除します。このコマンドを実行するには super 権限が必要です。

高度な PERFORCE 管理

ファイアウォールの利用

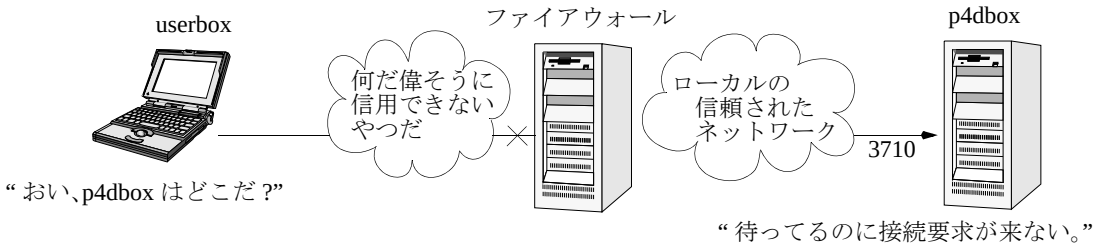
PERFORCE クライアントは、TCP/IP を利用して PERFORCE サーバと通信します。サーバはそれが動作しているマシン上の指定のポートで接続待ちし、クライアントはそのポートに接続要求を出します。

サーバが接続待ちするポートはサーバ起動時に指定されています。ポート番号は、他のネットワーク・サービスと衝突せず、かつ 1024 より大きければ、任意の番号を使用できます。クライアント・マシン上のポート番号は動的に割り当てられます。



*ファイアウォール*は（信頼された）ローカル・ネットワークの外部から来たパケットがローカル・ネットワークに到達するのを防止するネットワークの構成要素です。この防止動作はネットワーク・プロトコルの低いレベルで行われ、信頼された IP アドレス以外から来たパケットはすべて完全に無視されます。

次の図では、クライアントがネットワークの 信頼されていない部分にいます。その接続要求は PERFORCE サーバ搭載マシンに到達しません。結果的に、ファイアウォール越しにクライアントを操作しようとしているユーザは PERFORCE を利用できないことになります。



セキュア・シェル

上記の問題を解決するには、信頼されたネットワークの中からサーバに接続要求を出す必要があります。このためには、*セキュア・シェル*（ssh）というパッケージを用いると、安全です。

セキュア・シェル （ssh）は、リモート・システムにログインし、そこでコマンドを実行することを可能にする UNIX の rsh（remote shell）コマンドに代わるものです。“セキュア（安全な）” という語は、接続が暗号化され、信頼されていないネットワークを通過してもデータを読み取られない、という意味で使われています。rsh のような単純なユーティリティでは、すべてのトラフィックが -- パスワードさえも -- 暗号化されず、中間にあるすべてのホストで読み取ることができるので、セキュリティ上、非常に危険です。

セキュア・シェルは、UNIX の多数のプラットフォームに対応したソース形式で、http://www.openssh.com から無料で入手できます。このページは、Data Fellows の Windows 、Macintosh 用の商用実装のページ（http://www.datafellows.com）や SSH のページ（http://www.ssh.com）と同様に、OS/2 用および Amiga 用の ssh のポートにもリンクしています。

OpenSSH の FAQ も、メインサイトにおいてオンラインで見ることができます。

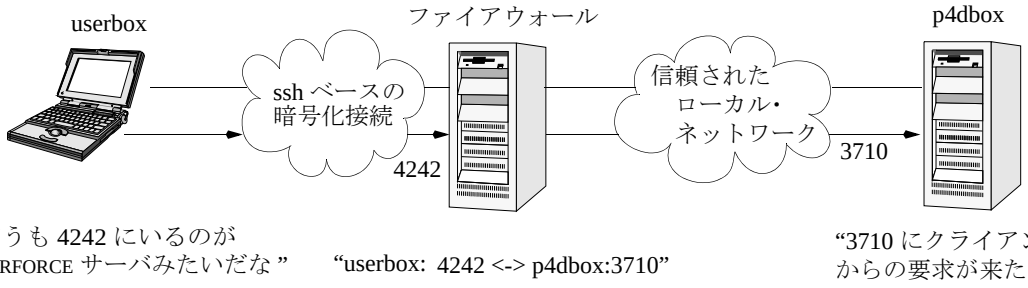
問題の解決法

ssh のインストール後に即可能なことは、ssh を使ってファイアウォール・マシンにログインし、ファイアウォールから PERFORCE クライアントを動作させることです。これは、簡単ではありますが、十分な解決法ではありません。通常はローカル・マシン上でクライアント・ファイルをアクセス可能にするべきで、もちろんファイアウォール・マシンが開発用プラットフォームに適合するという保証もありません。

解決法としてよいのは任意の TCP/IP 接続を*転送する* ssh の機能を利用する方法です。ssh を使えば、クライアントがファイアウォール・マシンから（信頼された）ローカル・ネットワー

クを介してサーバに接続しようとしているように見せかけることができます。現実には、クライアントはローカル・マシン上にいますが、そこから送られるパケットは、すべてまずは ssh によって設定された安全な通信経路を介してファイアウォールへ送られます。

今、サーバが p4dbox.bigcorp.com、ファイアウォール・マシンが firewall.bigcorp.com にあるとして、ローカル・ポートは任意に 4242 を選択し、PERFORCE サーバはポート 3710 で接続待ちしていると想定します。



クライアントのポート 4242 を最終目的地とするパケットはまずファイアウォールに送られ、ssh がそれを安全にクライアントへ転送します。同様に、ファイアウォール・マシンのポート 4242 に対して行われた接続は、最終的に PERFORCE サーバのポート 3710 につながります。

UNIX では、TCP/IP 接続を設定し、転送する ssh コマンドを次のように記載します。

```
ssh -L 4242:p4dbox.bigcorp.com:3710 firewall.bigcorp.com
```

この時点で、firewall.bigcorp.com にログインするパスワードを発行しておく必要があります。接続が確立されたら、ssh はローカル・マシンのポート 4242 で接続待ちし、受け取ったすべての情報を、暗号化された接続を介して firewall.bigcorp.com に転送し、ファイアウォールはさらにそれを通常の通信経路で p4dbox.bigcorp.com のポート 3710 に転送します。

あと必要なのは、環境変数 P4PORT を 4242 に設定し、ポート 4242 を使用するように、PERFORCE クライアントに 対して伝えることだけです。

通常、P4PORT=4242 という設定は、ポート 4242 で接続待ちしているマシン上の PERFORCE サーバに接続しようとすることを意味します。しかし、この場合には、ssh がそのサーバの役目を果たします。クライアントがローカル・マシンのポート 4242 に送ったものはすべて、ssh によってファイアウォールに送られ、それがさらに p4dbox.bigcorp.com の本当の PERFORCE サーバに転送されます。この手順はすべて、クライアントにとって透過的なので、クライアントがローカル・マシンのポート 4242 からのトラフィックを転送しようとしている ssh と通信しているか、本当の PERFORCE サーバと通信しているかは、問題ではありません。

唯一問題なのは、本来ならファイアウォール・マシン上で発生すべきではないログイン・セッションがあることです。

これはリモート・システム上で次のコマンドを実行することにより解決できます。

```
ssh -L 4242:p4dbox.bigcorp.com:3710 firewall.bigcorp.com -f sleep
9999999 -f
```

上記は firewall.bigcorp.com 上の ssh に対して、パスワードのプロンプトの後、長時間の sleep コマンドをバックグラウンドで生成させるためのコマンドです。実質的にはこれで ssh リンクが設定されて維持され、終了するログイン・セッションはなくなります。

最後に、ssh は“正しいことをする”ように構成できるので、上記のような長いコマンドをセッションのたびに打ち込む必要はありません。例えば、Windows 版 ssh には、その構成を行うための GUI があります。

留意すべきことが 1 つあります。トラフィックがローカル・マシンのポート 4242 から、安全と想定されるサーバに転送されると、ローカル・マシンも信頼されたネットワークの一部になります。そのローカル・マシンが本当に*安全かどうか*を慎重に確認してください。Windows 版 ssh には、転送に使用されるポートへの*ローカル接続のみ*を許可するオプションがあり、これはいい予防策になります。ローカル・マシンはポート 4242 を使えますが、第三者のマシンは無視されることになるわけです。

P4PORT で IP アドレスを指定する

ほとんどの場合、PERFORCE サーバの P4PORT は、ポート番号だけで構成されています。p4d 起動時に P4PORT で IP アドレス*および*ポート番号の両方を指定すれば、PERFORCE サーバはそれ以外の IP アドレスからの要求はいっさい無視するようになります。

これはデフォルト動作ではありませんが、例えば、特定のネットワーク・インターフェイスまたは IP アドレスに対してのみ接続待ちをするように p4d を構成したいときには有用です。例えば、P4PORT=localhost:*port* と設定すると、PERFORCE サーバはすべての非ローカル接続の要求を無視します。

UNIX の inetd で起動する

通常のインストールを行った PERFORCE サーバは、UNIX 上でバックグラウンド・プロセスとして動作し、クライアントからの接続要求を待ちます。inetd と p4d -i を使用して、接続要求があったときに初めて p4d を起動するようにする /etc/inetd.conf の後に次の行を記述します。

```
p4dservice stream tcp nowait username /usr/local/bin/p4d p4d -i -
rp4droot
```

また、/etc/services の後には次の行を記述します。

```
p4dservice nnnn/tcp
```

ただし、

- *p4dservice* はこの PERFORCE サーバ用に選んだサービス名です。
- /usr/local/bin は p4d のバイナリ・ファイルを保持しているディレクトリです。
- *p4droot* はこの PERFORCE サーバ用に使用するルート・ディレクトリ（P4DROOT）です。（例えば、/usr/local/p4d）
- *username* はこの PERFORCE サーバを動作させるために使用する UNIX ユーザ名です。
- *nnnn* はこの PERFORCE サーバが使用するポート番号です。

/etc/inetd.conf の行に“2 つめ”の p4d を必ず記述してください。inetd はこれを argv[0] として OS に渡します。そのとき、最初の引数は -i オプションで、これによって p4d は、デーモンとしてバックグラウンドで動作するのではなく、stdin/stdout 上で接続された単一のクライアントに対してサービスを行います。（これが inetd によって起動されるサービスの規定です。）

これは起動スクリプトから p4d を動作させる 1 つの方法です。これは、特別なサービスの提供にも利用できます。例えば、PERFORCE 社では、UNIX 上で動作するテスト・サーバが多数あり、それぞれが独自のポート番号を持つ inetd サービスとして定義されています。

この方法には、注意すべき点があります。

- inetd は過度な接続を許可しないことがあるので、数千の p4 コマンドを呼び出し、そのそれぞれが inetd を介して p4d のインスタンスを生じるようなスクリプトによって、inetd のサービスが一時的に使用不可になることがあります。システムによっては、この限度を無視するか、引き上げるために inetd の設定を変える必要があるでしょう。

- 毎回 p4d 実行可能プログラムが実行されるため、簡単にサーバを使用不可にする方法がありません。サーバを無効にするには、/etc/inetd.conf を修正し、inetd を再起動する必要があります。

大文字／小文字の区別とマルチプラットフォーム開発

初期（97.2 以前）のリリースの PERFORCE サーバは UNIX 環境、Windows 環境を問わず、すべてのファイル名、パス名、およびデータベースのエンティティ名を、大文字 / 小文字の区別をつけて扱っていました。

例えば、//depot/main/file.c と //depot/MAIN/FILE.C は、完全に異なる 2 つのファイルとして扱われました。このため、UNIX 上のユーザが Windows 上で動作している PERFORCE サーバに接続している場合に、様々な問題が生じました。すなわち、Windows サーバの基本的なファイルシステムでは、UNIX ユーザによってサブミットされる、大文字／小文字が区別される名前のファイルを保存できなかったからです。

リリース 97.3 で、この状況は変わりました。UNIX サーバだけが、大文字／小文字が区別される名前をサポートしています。とはいえ、UNIX と Windows の両方にまたがる環境で開発プロジェクトを行っている場合には、いまだにユーザは大文字／小文字区別の問題にぶつかることがあります。

97.2 以前のサーバを Windows でご使用の場合には、support@perforce.com に連絡して、サーバやデータベースのアップグレードをご相談ください。

現在リリースされているサーバでは、

- UNIX 上の PERFORCE サーバは、大文字／小文字が区別される名前をサポートしています。
- Windows 上の PERFORCE サーバは大文字 / 小文字の区別を無視します。
- プラットフォームに関係なく、キーワードによるジョブ検索では、常に大文字 / 小文字の区別を無視します。

以下の表は、これらの規則を要約しています。

	UNIX サーバ	Windows サーバ
パス名、ファイル名	する	しない
データベースのエンティティ名（ワークスペース名、ラベル名等）	する	しない
ジョブ検索のキーワード	しない	しない

PERFORCE サーバが動作しているプラットフォームは p4 info で確認できます。

PERFORCE サーバが UNIX 上にある場合

PERFORCE サーバが UNIX 上にあり、UNIX 上にも Windows 上にもユーザがいる場合には、UNIX ユーザは大文字と小文字が違うだけの名前のファイルをサブミットしないように十分に注意しなければなりません。UNIX サーバはそれらのファイルをサポートできますが、Windows ユーザがワークスペースを同期させれば、互いにファイルを上書きすることになります。

逆に、Windows ユーザは新しいファイルを追加するときにファイル名やパス名の大文字 / 小文字を一貫させる必要があります。彼らは //depot/main/one.c と //depot/MAIN/two.c のファイルが UNIX ユーザのワークスペースに同期された場合には 2 つの異なるディレクトリに格納されることを知らないかもしれません。

UNIX サーバは常にクライアント名、ラベル名、ブランチ・ビュー名等の大文字 / 小文字の違いを認識します。接続する Windows ユーザは、新しいワークスペース名がデフォルトでは小文字で登録されることを知っておく必要があります。例えば、新規ユーザが ROCKET という名前で Windows マシン上にクライアント仕様を生成すると、その名前はデフォルトでは rocket となります。後でそのユーザが P4CLIENT を ROCKET （または Rocket）と設定すれば、サー

バはそのクライアントは未定義と言ってきます。自分の定義したワークスペースを使うには、P4CLIENT を rocket と設定する（またはそれを解除する）必要があります。

PERFORCE サーバが Windows 上にある場合

PERFORCE サーバが Windows 上で動作していれば、UNIX ユーザは大文字と小文字が違うだけのファイルが同じネームスペースに保存されることを知っておかなければなりません。

例えば、次のような名前で 3 つのファイルを保存しようとすると、

```
p4 add dir/file1
p4 add dir/file2
p4 add DIR/file3
```

すべて同じディポ・ディレクトリに保存されることを知っておく必要があります。Windows サーバに割り当てられるディポのパス名とファイル名は最初に参照されたものになります。（この場合は、ディポのパス名は DIR ではなく dir になります。）

サーバの動作を監視する

p4 monitor コマンドを使用して、PERFORCE サーバ上で実行中の PERFORCE 関連プロセスに関する情報を取得します。

プロセスの監視を有効にする

サーバ・プロセスの監視には最小限のシステム・リソースしか必要としませんが、p4 monitor でプロセスの監視が動作できるように、設定を行う必要があります。プロセスの監視を有効にするには、次のようにカウンタ monitor を設定します。

```
p4 countor -f monitor 1
```

monitor カウンタを設定した後、その変更を有効にするには、PERFORCE サーバを再起動しなければなりません。

休止中プロセスの監視を有効にする

デフォルトでは、IDLE プロセス（多くの場合、PERFORCE API に基づくカスタム・アプリケーションに関連付けられている）は p4 monitor の出力に含まれていません。p4 monitor の出力に休止中プロセスを含めるには、監視レベル 2 を使用してください。

```
p4 counter -f monitor 2
```

変更を有効にするには、PERFORCE サーバを再起動しなければなりません。

実行中のプロセスを表示する

PERFORCE サーバで実行中のプロセスを表示するには、コマンド

```
p4 monitor show
```

を使用します。

デフォルトでは p4 monitor の出力の各行は、次のようになります。

```
pid status owner hh:mm:ss command [args]
```

ここで、*pid* は UNIX のプロセス ID（Windows のスレッド ID）です。*status* は R もしくは T で、実行中か終了しているかを示します。*owner* は、そのコマンドを実行しているユーザの PERFORCE ユーザ名です。*hh:mm:ss* は、そのコマンドを実行してから経過した時間です。*command* と *args* は、PERFORCE が受け取ったコマンドと引数です。例えば、次のようになります。

```
$ p4 monitor show
74612 R qatool     00:00:47 job
78143 R edk        00:00:01 filelog
78207 R p4admin    00:00:00 monitor
```

実行時、コマンドと共に指定された引数を表示するには、-a（引数）オプションを使用します。

```
$ p4 monitor show -a
74612 R qatool     00:00:48 job job004836
78143 R edk        00:00:02 filelog //depot/main/src/proj/file1.c //dep
78208 R p4admin    00:00:00 monitor show -a
```

ユーザ環境に関する追加情報を取得するには、-e オプションを使用します。-e オプションは、以下のようなフォームの出力を生成します。

pid client IP-address status owner workspace hh:mm:ss command [args]

この場合、*client* は PERFORCE クライアント・プログラム (およびバージョン文字列または API プロトコル・レベル)、IP-address はユーザの PERFORCE クライアント・プログラムの IP アドレス、workspace はユーザの現在のクライアント・ワークスペース設定です。例えば、以下のようになります。

```
$ p4 monitor show -e
74612 p4        192.168.10.2   R qatool    buildenvir 00:00:47 job
78143           192.168.10.4   R edk       eds_elm    00:00:01 filelog
78207 p4        192.168.10.10  R p4admin   p4server   00:00:00 monitor
```

デフォルトでは、すべてのユーザ名とクライアント・ワークスペース名 (該当する場合) は 10 文字で切り捨てられ、行は 80 文字で切り捨てられます。この切り捨てを無効にするには、-l (長いフォーマット) オプションを使用します。

```
$ p4 monitor show -a -l
74612 R qatool     00:00:50 job job004836
78143 R edk        00:00:04 filelog //depot/main/src/proj/file1.c //dep
ot/main/src/proj/file1.mpg
78209 R p4admin    00:00:00 monitor show -a -l
```

-a、-l、–e の各オプションを使用できるのは、PERFORCE 管理者とスーパーユーザのみです。

プロセスを終了させる

PERFORCE サーバ上のプロセスが大量のリソースを消費しているとき、管理者やスーパーユーザは p4 monitor terminate によって、そのプロセスを終了させることができます。

いったん終了が予約されると、50000 行または列をスキャンするまでの間に、そのプロセスは PERFORCE サーバによって終了されます。終了を予約することができるプロセスは、少なくとも 10 秒間実行し続けているプロセスのみとなります。プロセスを終了させられたユーザに対しては、次のメッセージが通知されます。

```
Command has been canceled, terminating request
```

対話形式のフォームを使用するプロセス（p4 job、p4 user など）も終了を予約することができますが、ユーザによってフォームに入力されたデータは保持されます。p4 obliterate など、一部のコマンドは終了させることができません。

プロセス・テーブルのエントリをクリアする

ある状況下（例えば、PERFORCE のプロセスがまさに実行中のときに Windows マシンがリブートされる）においては、プロセスが終了した後であっても、プロセス・テーブル内にエントリが残る場合があります。

PERFORCE の管理者とスーパー・ユーザは、p4 monitor clear *pid* を実行することによって、これらの誤ったエントリをプロセス・テーブルから削除することができます。このとき、*pid* はその誤ったプロセス ID です。(実行中であるか否かにかかわらず) テーブルからすべてのプロセスをクリアするには、p4 monitor clear all を実行します。

`p4 monitor clear` によってプロセス・テーブルから削除された実行中プロセスは、終了処理を実行します。

PERFORCE サーバ・トレースおよび追跡用オプション

コマンド・トレーシングまたはパフォーマンス追跡を実行可能にするには、`p4d` 起動コマンドに適切な `-v var=value` オプションを指定します。ログ・ファイルの名前は `P4LOG` または `-L logfile` オプションで指定します。例えば次のように記述します。

```
p4d -r /usr/perforce -v server=1 -p 1666 -L /usr/perforce/logfile
```

ログ記録を開始する前に、十分なディスク容量を確保しておく必要があります。

Windows *Windows のサービスとして Perforce を起動している場合、`p4 set` コマンドを使用して `P4DEBUG` をレジストリ変数に設定してください。これらのトレース・オプションは、コマンド・ラインから `p4d.exe` をサーバ・プロセスとして起動して設定することもできます。*

PERFORCE サーバ (p4d) でサーバ・デバッグ・レベルを設定しても、PERFORCE プロキシ (p4p) プロセスのデバッグ・レベルには何の影響もありません。この逆も同様です。

通常、PERFORCE サーバ・コマンドのトレーシングおよび追跡のオプションは、システム管理者が PERFORCE テクニカル・サポートと共同で問題の診断・調査を行う場合にのみ有用です。

コマンドのトレーシング

サーバ・コマンド・トレース・オプションおよびその意味は次のとおりです。

トレース・オプション	意味
`server=1`	サーバ・コマンドをサーバのログファイルに記録します。 (サーバは、98.1 以降である必要があります。)
`server=2`	レベル 1 で記録されるデータのほか、サーバ・コマンドの完了、CPU 使用時間の基本情報を記録します。経過時間は秒単位でレポートされます。UNIX では、CPU 使用 (システム、ユーザ時間) は、`getrusage()` により、ミリ秒でレポートされます。 (サーバは、2001.1 以降である必要があります。)
`server=3`	レベル 2 で記録されるデータのほか、`p4 sync`、`p4 flush` (`p4 sync -k`) コマンドのコンピュート・フェーズについての使用情報を追加します。 (サーバは、2001.2 以降である必要があります。)

コマンド・トレーシングでは、出力は特定のログ・ファイルに記録され、日付、時間、ユーザ名、IP アドレス、およびサーバにより処理される各リクエストのコマンドを示します。

パフォーマンス追跡

ユーザ・コマンドが事前定義のリソース使用量のしきい値を超えると、Perforce サーバによりサーバ・ログに診断結果が生成されます。

tracking オプション	意味
`track=0`	追跡を無効化
`track=1`	すべてのコマンドを追跡
`track=2`	ユーザ数が 10 未満のサーバについて使用量の超過を追跡
`track=3`	ユーザ数が 100 未満のサーバについて使用量の超過を追跡

tracking オプション	意味
`track=4`	ユーザ数が 1000 未満のサーバについて使用量の超過を追跡
`track=5`	ユーザ数が 1000 を超えるサーバについて使用量の超過を追跡

追跡の出力の厳密なフォーマットは変更される場合があり、ここでは記述しません。

PERFORCE サーバ (`p4d`) でサーバ・デバッグ・レベルを設定しても、PERFORCE プロキシ (`p4p`) プロセスのデバッグ・レベルには何の影響もありません。この逆も同様です。

通常、PERFORCE サーバのトレース・オプションは管理者が PERFORCE テクニカル・サポートと共同で問題の診断・調査にあたっているときにのみ有用です。

ユーザによるファイル・アクセスの監査

リリース 2006.1 では、PERFORCE サーバは個々のファイルへのアクセスを監査用ログファイルに記録することができます。

デフォルトでは監査機能は無効にされており、これを有効にするには、`P4AUDIT` を監査用ログファイルの場所を示すよう設定するか、サーバ起動時に `-A auditlog` オプションを指定しなければなりません。

監査が有効にされていると、サーバからクライアントへファイル・コンテンツが転送されるたびに、サーバが監査用ログファイルに一行追加します。アクティブなサーバ上では、監査用ログファイルは非常に急速に増大します。

監査用ログ内の行は次の形式で出力されます。

日付　時間　ユーザ @ クライアント　クライアント IP アドレス　コマンド　ファイル # リビジョン

例を示します。

```
$ tail -2 auditlog
2006/05/09 09:52:45 karl@nail 192.168.0.12 diff //depot/src/x.c#1
2006/05/09 09:54:13 jim@stone 127.0.0.1 sync //depot/inc/file.h#1
```

コマンドが PERFORCE サーバを起動しているマシン上で実行されると、*クライアント IP アドレス*は `127.0.0.1` と表示されます。

PERFORCE サーバを新しいマシンに移動する

既存の PERFORCE サーバを別のマシンに移行する手順は次の点によって変わってきます。

- 同一のアーキテクチャのマシンへの移動か
- テキスト・ファイル（CR/LF）のフォーマットは同じだが、アーキテクチャの異なるマシンへの移動か
- アーキテクチャもテキスト・ファイルのフォーマットも異なるマシンへの移動か

新しいマシンの IP アドレス / ホスト名が変わる場合には、さらに考慮すべきことがあります。

PERFORCE サーバはルート・ディレクトリに 2 種類のデータを保存しています。*バージョン化ファイル*とそれを記述する*メタデータからなるデータベース*です。バージョン化ファイルはユーザが作成・保管するファイルです。データベースは PERFORCE が保管しているバイナリ・ファイルで、バージョン化ファイルの履歴および現在の状態を保持しています。PERFORCE サーバを別のマシンに移すときには、必ずこれらのバージョン化ファイルとデータベースも新しいマシンに移さなければなりません。

バージョン化ファイルとデータベースのさらに細かい違いや、バックアップ、リストアの手順については、25 ページの「バックアップとリカバリの考え方」をご覧ください。

PERFORCE サーバの別のマシンへの移行については、次の PERFORCE Tech Note もご覧ください。

```
http://www.perforce.com/perforce/technotes/note010.html
```

同一アーキテクチャのマシン間の移動

新旧マシンのアーキテクチャが同じ（例えば SPARC/SPARC、x86/x86）なら、バージョン化ファイルもデータベースも 2 台の間で直接コピーでき、必要なのはサーバ・ルートのディレクトリ・ツリーを新しいマシンに移すことだけです。`tar`、`cp`、`xcopy.exe` など、あらゆる方法が使えます。`P4ROOT` ディレクトリにあるもの – `db.*` ファイル（データベース）とディポ・サブディレクトリ（バージョン化ファイル）– をすべてコピーしてください。

1. サーバのバックアップを作成し （事前に `p4 verify` も行うこと）、チェックポイントを取ります。
2. 古いマシン上で `p4d` を終了します。
3. 古いマシン上の古いサーバ・ルート（`P4ROOT`）とそのすべてのサブディレクトリの内容を新しいマシンの新しいサーバ・ルート・ディレクトリにコピーします。
4. 新しいマシン上で、希望のオプションを使って、`p4d` を起動します。
5. 新しいマシンで `p4 verify` を実行し、データベースとバージョン化ファイルが新しいマシンに正しく移されたことを確認します。

（同一アーキテクチャ間の移動の場合、バックアップ、チェックポイント、移行後の `p4 verify` は、厳密には必要ありませんが、移行の前後でシステムの完全性確認、チェックポイント、バックアップを行っておくのはよいことです。）

異なるアーキテクチャで同一テキスト・フォーマットのマシン間の移動

2 台のマシン間で内部データ表現（ビッグエンディアンかリトルエンディアン）の規定に違いがあるが（例えば、Linux-on-x86/SPARC、NT-on-Alpha/NT-on-x86）、オペレーティング・システムはともに同じ CR/LF テキスト・ファイル規定を採用している場合も、簡単にサーバ・ルートのディレクトリ・ツリーを新しいマシンに移すことができます。

バージョン化ファイルはアーキテクチャ間で移動可能ですが、`db.*` ファイルに保存されているデータベースは違います。この問題を解決するには、PERFORCE サーバのデータベースのチェックポイントを作成し、そのチェックポイントを利用して、新しいマシン上にデータベースを再構成します。チェックポイントはテキスト・ファイルであるため、あらゆるアーキテクチャの PERFORCE サーバで読み取ることができます。詳しくは、26 ページの「チェックポイントの作成」をご覧ください。

チェックポイントを作成したら、`tar`、`cp`、`xcopy.exe` などを用いてそのチェックポイント・ファイルとディポ・ディレクトリを新しいマシンにコピーします。（`db.*` ファイルはチェックポイントから再生されるので、コピーする必要はありません。）

1. 古いマシン上で `p4 verify` を使用し、データベースの完全性を確認します。
2. 古いマシン上で `p4d` を終了します。
3. 古いマシン上でチェックポイントを生成します。
   ```
   p4d -jc checkpointfile
   ```
4. 古いマシン上の古いサーバ・ルート（`P4ROOT`）とそのすべてのサブディレクトリの内容を新しいマシン上の新しいサーバ・ルート・ディレクトリにコピーします。

(厳密には db.* ファイルをコピーする必要はありません。必要なのはチェックポイントとディポ・ディレクトリだけです。db.* ファイルは、チェックポイントから再生されます。すべてをコピーする方が都合がよければ、すべてをコピーして構いません。)

5. db.* ファイルもコピーした場合、必ず新しいマシン上で新しい P4ROOT から切り離してください。
6. 作成したチェックポイントから新しいマシンのアーキテクチャに適した新しい db.* ファイルのセットを再構成します。

   ```
   p4d -jr checkpointfile
   ```

7. 新しいマシン上で、希望のオプションを使って、p4d を起動します。
8. 新しいマシンで p4 verify を実行し、データベースとバージョン化ファイルが新しいマシンに正しく移されたことを確認します。

Windows と UNIX との間の移行

この場合は、システムのアーキテクチャも CR/LF テキスト・ファイル規定も異なります。同様にチェックポイントを作成してコピーし、新しいプラットフォーム上でデータベースを再構成する必要がありますが、バージョン化ファイルを格納しているディポのサブディレクトリを移すときは、プラットフォーム間の異なる改行規定にも対処しなければなりません。

ディポのサブディレクトリには、テキスト・ファイルもバイナリ・ファイルも格納されます。テキスト・ファイル (RCS フォーマットなら最後が ",v") とバイナリ・ファイル (個々にディレクトリを持ち、各ディレクトリの最後が ",d") は、テキスト・ファイルでは行末を変換するため (バイナリ・ファイルはそのまま)、別々の方法で移す必要があります。

前述の他の移行の場合と同様、移行後は必ず p4 verify を実行してください。

> *Windows は大文字 / 小文字の区別をしないオペレーティング・システムです。UNIX 上で大文字 / 小文字の違いによって別ファイルと認識されているファイルは、Windows マシンに移されると、同じネームスペースを占めます。例えば、*Makefile *というファイルと* makefile *というファイルは、UNIX 上では別ファイルとして認識されますが、Windows マシン上では同じファイルとして扱われます。大文字 / 小文字の区別がなくなることによるデータ損失のリスクがあるため、UNIX サーバから Windows への移行はお勧めしません。*

PERFORCE サーバを Windows から UNIX へ、またはその逆に移行するときは、PERFORCE テクニカル・サポートまでご相談ください。

サーバの IP アドレスの変更

新しいマシンの IP アドレスが古いマシンのときと異なる場合は、プロテクション・テーブルの IP アドレスに基くプロテクションを更新する必要があります。PERFORCE サーバのプロテクションの設定については、69 ページの「PERFORCE の管理 : プロテクション」をご覧ください。

ライセンスをお持ちの場合でも、新 IP アドレスを反映した新しいライセンス・ファイルが必要になります。PERFORCE テクニカル・サポートに連絡して更新されたライセンスを取得してください。

サーバのホスト名の変更

新しいサーバ・マシンのホスト名が旧マシンのときと異なる場合は、P4PORT の設定を変更する必要があります。旧マシンが廃棄またはリネームされる場合は、旧マシンに適合するエイリアスを新しいマシンに設定すれば、P4PORT の設定変更が不要になります。

複数のディポを利用する

新規のディポを定義するには p4 depot depotname コマンドを使います。ディポのタイプは local、remote、spec のうちいずれかになります。

PERFORCE サーバには複数のディポを置くことができ、クライアント・プログラムも複数のディポのファイルにアクセスできます。これらのディポは、通常クライアントがアクセスする PERFORCE サーバに置くことも、他の PERFORCE サーバ、すなわち*リモート* PERFORCE サーバに置くこともできます。

ローカル・ディポはユーザの PERFORCE クライアント・プログラムが通常アクセスする PERFORCE サーバに存在します。ローカル・ディポ使用時には、PERFORCE クライアント・プログラムはユーザの環境変数 P4PORT または同等の設定で指定された PERFORCE サーバと通信します。

リモート・ディポ使用時には、ユーザの PERFORCE クライアント・プログラムは、ユーザの環境変数 P4PORT または同等の設定で指定された PERFORCE サーバを、第 2 の PERFORCE サーバ、すなわち*リモート* PERFORCE サーバに対する中継点として利用します。ローカルの PERFORCE サーバは、リモートの PERFORCE サーバにあるファイルの一部にアクセスするため、リモートの PERFORCE サーバと通信します。リモート・ディポは、主に独立した組織間におけるコードの共有（「コード・ドロップ」と言います）を容易にするために使用されます。これについては、62 ページの「リモート・ディポと分散開発」に記載されています。

リモート・ディポは、負荷バランスやネットワーク・アクセスの問題の総合的な解決策ではありません。共有開発や、負荷分散、ネットワーク・アクセスの問題をサポートするには、129 ページの「第 9 章 PERFORCE プロキシ」をご覧ください。

スペック・ディポは特殊なディポです。スペック・ディポが存在する場合、クライアント・ワークスペース仕様、ジョブ、ブランチ仕様などのユーザ編集フォームに対する変更を追跡します。スペック・ディポは一つのサーバに対して一つしか作成できません。

ディポのネーミング

ディポ名は、ブランチ名、クライアント・ワークスペース名、ラベル名とネームスペースを共用します。例えば、//rel2 は ディポ rel2、ワークスペース rel2、ブランチ rel2、またはラベル rel2 のうちいずれかを一意に指すものです。同時に rel2 というディポとラベルを両方持つことはできません。

新しいローカル・ディポを定義する

新しいローカル・ディポを定義する（すなわち、カレント PERFORCE サーバのネームスペースに新しいディポを作成する）には、新しいディポの名前を付けて p4 depot を呼び出し、画面に表示されるフォーム内の Map: のフィールドのみを変更します。

例えば、ローカル PERFORCE サーバのネームスペースに保存されたファイルを持つ book という新しいディポを、book というルート・サブディレクトリ（すなわち、$P4ROOT/book）に

生成するには、コマンド `p4 depot book` を入力し、表示されるフォームを次のように埋めます。

```
Depot:      book
Type:       local
Address:    subdir
Suffix:     .p4s
Map:        book/...
```

`Address:` フィールドと `Suffix:` フィールドはローカル・ディポには適用されないため、無視されます。

デフォルトでは、ローカル・ディポ上の `Map:` フィールドは、ディポ名と一致しているディポ・ディレクトリを指し示し、サーバ上で設定されているサーバルート（`P4ROOT`）からの相対パスとなります。ディポでバージョン管理するファイルを別のボリュームやドライブに保持するには、`Map:` フィールドに絶対パスを指定します。このパスは `P4ROOT` 下にある必要はありません。Windows 上で `Map:` フィールドに絶対パスを指定する場合は、ディポ・フォームでスラッシュ（`d:/newdepot/` など）を使用する必要があります。

スペック・ディポを利用してバージョン化仕様を有効にする

スペック・ディポを利用してバージョン化仕様を有効にするユーザがユーザ編集フォームに対する変更履歴を取得するには、バージョン化仕様を有効にする必要があります。スペック・ディポにはどのような名前でも付けられますが、タイプを `spec` にする必要があります。また、一つのサーバに対して一つしか作成できません。（スペック・ディポが既に存在する場合、別のスペック・ディポを作成しようとするとエラーが出力されます）。

スペック・ディポを作成してバージョン化仕様を有効にすると、すべてのユーザ生成フォーム（クライアント・ワークスペース仕様、ジョブ、ブランチ仕様など）は自動的にスペック・ディポにテキストファイルとしてアーカイブされます。スペック・ディポでのファイル名はサーバにより自動的に生成され、以下の PERFORCE 構文で表示されます。

```
//specdepotname/formtype/objectname[suffix]
```

スペック・ディポを作成する

`//spec` という名前のスペック・ディポを作成するには、`p4 depot spec` と入力し、その結果表示されるフォームに次のように入力します。

```
Depot:      book
Type:       local
Address:    subdir
Map:        spec/...
Suffix:     .p4s
```

`Address:` フィールドはスペック・ディポには適用されないため、無視されます。

`Suffix:` は任意選択ですが、スペック・ディポにあるオブジェクトにファイル拡張子を指定することによって、P4V や P4Win などのグラフィカル・クライアント・プログラムを使用するユーザの利便性が向上します。それは、PERFORCE 仕様で使われる拡張子と、各自が使用したいテキスト・エディタとを関連付けることができるからです。生成されるファイルのデフォルトの拡張子は `.p4s` です。

例えば、`spec` という名前のスペック・ディポを作成し、デフォルトの拡張子 `.p4s` を使用する場合、以下のコマンドを使用して `job000123` に対する変更履歴を調べることができます。

```
p4 filelog //spec/job/job000123.p4s
```

または P4V や P4Win を使用し、ワークステーション上でファイル拡張子 `.p4s` と関連付けられている任意のエディタを使って `job000123.p4s` への変更を確認することもできます。

スペック・ディポに現在のフォームからデータを読み込む

スペック・ディポを作成したら、`p4 admin updatespecdepot` コマンドでそこにデータを読み込むことができます。このコマンドを実行すると、PERFORCE サーバは保管フォーム（具体的には、`client`、`depot`、`branch`、`label`、`typemap`、`group`、`user`、および `job` の各フォーム）をスペック・ディポにアーカイブします。

現在のフォームをすべてアーカイブするには、`-a` オプションを以下のように使用します。

```
p4 admin updatespecdepot -a
```

特定のフォーム・タイプを持つスペック・ディポにデータを読み込むには（例えば、非常に大規模なサイトで 1 度に 1 つのテーブルだけを更新したい場合など）、コマンドラインで `-s` オプションを使用してフォームの *タイプ* を指定します。例 :

```
p4 admin updatespecdepot -s job
```

どちらの場合でも、まだアーカイブされていないフォームのみがスペック・ディポに追加されます。スペック・ディポの作成後は `p4 admin updatespecdepot` を 1 度実行するだけで済みます。

ディポのリスト表示

カレント PERFORCE サーバが認識しているすべてのディポのリストは `p4 depots` コマンドで表示することができます。

ディポの削除

ディポを削除するには、`p4 depot -d` *`depotname`* コマンドを使います。

ディポを削除するには、事前に `p4 obliterate` を実行し、ディポ内のすべてのファイルを削除しておかなければなりません。

ローカル・ディポおよびスペック・ディポにおいて、`p4 obliterate` は関連するメタデータとともにバージョン化ファイルを削除します。リモート・ディポにおいて、`p4 obliterate` はローカルに保持しているクライアントおよびラベルの記録*のみ*を消去します。リモート・サーバ上にあるファイルとメタデータは、そのまま残ります。

`p4 obliterate`を実行する前、特にディポ内のすべてのファイルを削除するようなときには、44 ページの「ファイル削除によるディスク容量の再生」に記述されている警告を読んで理解してください。

リモート・ディポと分散開発

リモート・ディポは、共用*開発*ではなく共用*コード*をサポートするように設計されています。独自の PERFORCE システムを持つ独立した組織が、他のシステムのディポに保存されたファイルを参照したり、その変更を反映したりすることを可能にします。簡潔にまとめると、次のようになります。

- 「リモート・ディポ」とは、ローカルの PERFORCE サーバ上に存在するディポであり、タイプが `remote` です。
- リモート・ディポのユーザは多くの場合、独立した組織間でソフトウェアの統合を担当しているビルド・エンジニアか、もしくは提供資料の管理者です。
- リモート・サーバの管理者と一緒にいるリモート・ディポのユーザに対して、どのファイルが有効であるかを管理します。ローカル・ディポのユーザに対してではありません。
- セキュリティの要求については、65 ページの「リモート・ディポへのアクセスを制限する」をご覧ください。

いつリモート・ディポを使用するか

PERFORCE は大規模なネットワークの待ち時間に対処するように設計されており、本来、遠隔地にクライアント・ワークスペースを持つユーザをサポートします。PERFORCE は、参加者の地理的な分散に関係なく、1 台のサーバで共用開発プロジェクトをサポートできるのです。

共同開発プロジェクトを分割し、複数の PERFORCE システムで行っても、スループットが改善されることはなく、通常は管理が複雑になるだけでしょう。サイトが分散型の開発に参加している場合には、通常は単一の PERFORCE サーバ上のディポ内にすべてのコードを置き、個々の開発サイトの頻繁にアクセスされるファイルを PERFORCE プロキシによってキャッシュするようにした方がよいでしょう。

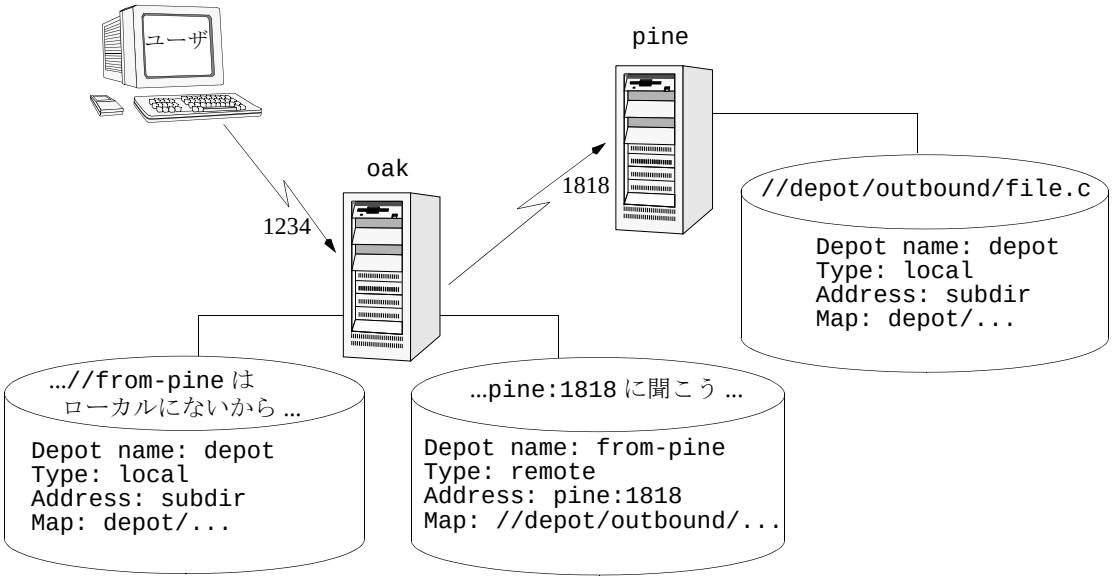
ただし、他の組織の成果物を定期的にインポートしたりエクスポートしたりするような場合には、コード・ドロップの手順を合理化するために PERFORCE のリモート・ディポは利用価値があると言えるでしょう。

リモート・ディポはどのように動作するか

別のサーバのディポにあるファイルに対して、PERFORCE クライアントがどのようにユーザのデフォルト PERFORCE サーバにアクセスするかを、次の図に示します。

この例では、`oak`:1234 の PERFORCE サーバにおける管理者は、`pine:1818` の PERFORCE サーバからファイルを取り出そうとしています。

```
P4PORT=oak:1234
p4 integ //from-pine/file.c //depot/codedrops/file.c
```



個々の開発者がリモート・ディポからクライアント・ワークスペースへファイルを同期することは可能ですが、一般的にリソースの使い方が効率的ではありません。

組織におけるビルドや提供資料の管理者にとって、リモート・ディポからローカル・ディポの領域へファイルを反映することが、効率的なリモート・ディポの使い方と言えます。反映した後、開発者は、ローカル・ディポ内にある反映済みのファイルのコピーにアクセスすることができます。

リモート・ディポからのコード・ドロップを承諾するには、ローカル・ディポ内にリモート・ディポ内のファイルからのブランチを作成し、そのローカル・ブランチの中へリモート・ディポからの変更を反映します。この反映は、一方通行の操作となります。すなわち、ローカル・ブランチで実施した変更をリモート・ディポへ反映することはできません。自サイトの

PERFORCE 環境の中へ反映したファイルのコピーに対しては、自サイトの開発チームが責任を持ちます。ディポ上のファイルは、他の PERFORCE 環境における開発チームの管理下に残ります。

リモート・ディポの制約

リリース 99.2 以前は、リモート・ディポは、同じリリース・レベルで動作している PERFORCE サーバによってのみアクセスできました。リリース 99.2 またはそれ以降のリリースでは、リモート・ディポはリリース・レベル間で相互運用が可能です。

リモート・ディポは、単一組織での共同開発ではなく、組織間でのコード共用を容易にするものです。結果的に、リモート・ディポへのアクセスは一方通行の操作に制限され、リモート・ディポを使ってサーバのメタデータ（クライアント・ワークスペース、チェンジリスト、ラベルなどの情報）をアクセスすることはできません。

コード・ドロップのためにリモート・ディポを使用する

コード・ドロップを実現するには、2 つの組織間に役割を持たせる必要があります。すなわち、コード・ドロップを受領するサイトと、コード・ドロップを提供するサイトです。多くの場合、次の 3 点が設定されていなければなりません。

- コード・ドロップを受領するサイトの PERFORCE 管理者は、コード・ドロップを提供するサイトを参照するリモート・ディポを、自分のサイト内に構築しなければなりません。

  これは、64 ページの「リモート・ディポを定義する」に説明されています。

- コード・ドロップを提供するサイトの PERFORCE 管理者は、受領側のサイトにあるリモート・ディポから、自分のサイトの PERFORCE サーバへのアクセスを許可するよう環境設定すべきです。

  これは、65 ページの「リモート・ディポへのアクセスを制限する」に説明されています。

- 受領サイトの構成管理者または統合管理者は、彼らの管理下において、リモート・ディポからローカル・ディポへファイルを反映しなければなりません。

  これは、66 ページの「コード・ドロップを受領する」に説明されています。

リモート・ディポを定義する

新しいリモート・ディポを定義するには、次のように実行します。

1. `p4 depot` *`depotname`* によってディポを作成します。
2. `Type:` を `remote` に設定します。
3. `Address:` フィールドにリモート・サーバの名前と接続待ちのポート番号を設定することによって、ローカルの PERFORCE サーバからリモートの PERFORCE サーバに接続できるようにします。

   リモート・サーバのホストとポートを `Address:` フィールドに設定する書式は、ちょうど `P4PORT` を設定するのと同じ書式です。

4. リモート・サーバの空間において参照したい部分へのマッピングを `Map:` フィールドに設定します。

   リモート・ディポに対して、そのマッピングはリモート・ディポの空間を相対参照するサブディレクトリを含みます。例えば、`//depot/outbound/...` というマッピングは、リモート・サーバ上にある `depot` という名前のディポにおける `outbound` サブディレクトリを指します。

   `Map:` フィールドはこのサブディレクトリを指す単一の行で構成され、ディポ・シンタックスで指定され、最後にはワイルドカード“`...`”を含んでいなければなりません。

もしクライアント・ビューとマッピングについてよくご存知ない場合は、『P4 ユーザーズ・ガイド』をご覧ください。PERFORCE のマッピング機能に関する一般的な情報が記載されています。

5. `Suffix:` フィールドはリモート・ディポには適用されませんので無視してください。

ローカル・サイト内の誰かがリモート・ディポのファイルをアクセスする場合は、リモート・ディポの管理者が、リモート・ディポおよび `Map:` フィールドに指定されたディポ内のサブディレクトリへの `read` 権限を、ユーザ `remote` に対して設定しなければなりません。

例 : *リモート・ディポを定義する*

ライザはあるプロジェクトに取り組んでおり、サード・パーティ開発会社からのライブラリ・セットを、彼女のサイトの開発者に対して提供しようとしています。そのサード・パーティ開発会社はホスト `pine` *に PERFORCE サーバを設置していて、ポート* `1818` *で接続待ちをしています。彼らの方針としては、*`depot` *という名前のディポにある* `outbound` *というサブディレクトリに、彼らのライブラリのリリースを置くこととします。*

ライザは新しいディポを作成し、そのディポからコード・ドロップをアクセスできるようにします。彼女はこのディポを `from-pine` *と名付け、*`p4 depot from-pine` *と入力し、次のようにフォームを埋めます。*

```
Depot:      from-pine
Type:       remote
Address:    pine:1818
Map:        //depot/outbound/...
```

これでライザの PERFORCE サーバに `from-pine` *というリモート・ディポが作成されます。このディポ（*`//from-pine`*）は、サブディレクトリ* `outbound` *配下にあるサード・パーティのディポのネームスペースをマッピングします。*

リモート・ディポへのアクセスを制限する

リモート・ディポは、常に `remote` という仮想ユーザによってアクセスされます。この仮想ユーザには、PERFORCE のライセンスは不要です。

デフォルトでは、PERFORCE サーバ上のすべてのファイルがリモートからアクセス可能です。特定のサーバに対するリモート・アクセスを制限または排除するには、`p4 protect` を使って、そのサーバ上でユーザ `remote` へのパーミッションを設定します。

`p4 protect` のテーブルに次のようなパーミッション行を追加することによって、すべてのファイルおよびすべてのディポについて、ユーザ `remote` のアクセスを拒否することをお勧めします。

```
list user remote *  -//...
```

`remote` ディポは読み取りの目的でしか利用できないので、書き込みまたは特権のアクセスを排除する必要はありません。

セキュリティ構成の例

66 ページの「コード・ドロップを受領する」で説明された 2 つの組織を使って、それぞれの組織におけるセキュリティ構成の基本例を示します。

ローカル・サイト（`oak`）において

- すべてのユーザに対して、`//from-pine` へのアクセスを拒否します。`oak` サイト開発者は、リモート・ディポのメカニズムによって `pine` サーバ上のファイルにアクセスする必要がありません。

- 統合またはビルド管理者に対して、`//from-pine` への `read` アクセスを許可します。`oak` サイトにおいてリモート・ディポ `//from-pine` へのアクセスを要求する唯一のユーザは、リモート・ディポからローカル・ディポへの反映を実行するユーザ（この例では "`adm`"）です。

  これを実現するために `oak` の PERFORCE 管理者は、`p4 protect` のテーブルに次の行を含める必要があります。

```
list user ** -//from-pine/...
read user adm * //from-pine/...
```

リモート・サイト（`pine`）では、`pine` 上にあるコードへのアクセスは、完全に `pine` の PERFORCE 管理者の管理下にあります。少なくともこの PERFORCE 管理者は、次のことをする必要があります。

- あらかじめ、ユーザ `remote` に対して、すべての IP アドレスからのすべてのディポに対するアクセスを拒否します。

```
list user remote * -//...
```

  この行を `p4 protect` テーブルに追加することは、システム管理者がリモート・ディポを使用しようとしているかどうかとは関係なく、あらゆる PERFORCE 環境に対して手堅い運用です。

- サーバ `pine` 上でコード・ドロップを配置する領域に対して、ユーザ `remote` に対する `read` アクセスを許可します。この例では、提供されるコード・ドロップはサーバ `pine` 上の `//depot/outbound/...` サブディレクトリにおいて公開されます。
- コード・ドロップを受領することを認可された PERFORCE サーバの IP アドレスに対してのみ、ユーザ `remote` の `read` アクセスを許可します。`oak` の IP アドレスが `192.168.41.2` の場合、`pine` の PERFORCE 管理者は `p4 protect` テーブルに次の行を追加しなければなりません。

```
read user remote 192.168.41.2 //depot/outbound/...
```

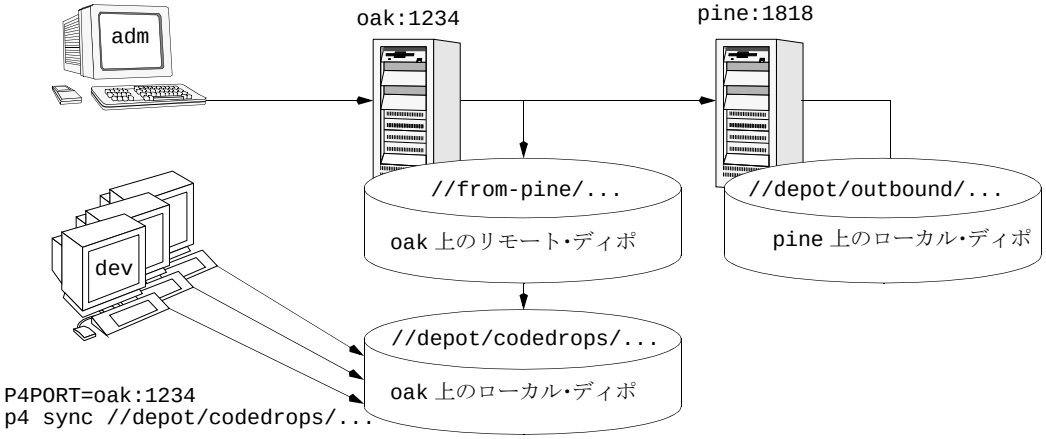
コード・ドロップを受領する

2 つの PERFORCE 環境の間で提供物のやり取りやコード・ドロップの実施をするには :

1. `pine:1818` で作業する開発者は、外部提供するためのコードの内容を書き終えます。
2. `pine:1818` のビルド管理者またはリリース管理者は、コード・ドロップの外部提供を意図した `pine:1818` 上の領域に、提供可能となったコードをブランチします。この例では、リリースされるコードは `//depot/outbound/...` にブランチされます。
3. `oak:1234` の PERFORCE 管理者は、`oak` サーバ上に `//from-pine` という名前のリモート・ディポを構築します。このリモート・ディポの `Map:` フィールドでは、`oak` サーバにおいて `pine:1818` の `//depot/outbound/...` 配下を参照するための指定を行います。
4. リリース可能の通知を受けると、`oak:1234` のビルド管理者またはリリース管理者は、リモート・ディポ `//from-pine/...` 配下のファイルをローカル・ディポの適当な領域（`//depot/codedrops/pine` のような）に反映することによって、コード・ドロップを実施します。
5. `oak:1234` の開発者は、今やローカルの `//depot/codedrops/pine` 配下にある `pine` のコードを使用できるようになりました。`pine` のコードに対してパッチが必要であるな

らば、oak の開発者は //depot/codedrops/pine 配下でそのようなパッチを作成することができます。pine グループは、引き続きそのコードを管理します。

```
P4PORT=oak:1234
p4 integrate //from-pine/... //depot/codedrops/pine/...
```



第 4 章

PERFORCE の管理 : プロテクション

PERFORCE はディポに対して権限のないアクセス、または不注意によるアクセスを防ぐプロテクション・スキームを提供します。プロテクション・スキームにより、どのコマンドが、どのファイルについて、誰によって、どのホストから実行できるかが限定されます。プロテクションの設定は `p4 protect` コマンドを使って行います。

いつプロテクションを設定するか？

`p4 protect` は、PERFORCE を初めてインストールした直後に実行してください。`p4 protect` を実行するまでは、すべての PERFORCE ユーザはスーパー・ユーザであり、ディポ内のあらゆるものにアクセスしてそれらを変更することができます。`p4 protect` を初めて実行すると、プロテクション・テーブルが作成され、そのユーザにすべての IP アドレスからのスーパー・ユーザ権限を提供し、その他のユーザのアクセス・レベルをすべての IP アドレスからの全ファイルに対する `write` 権限に下げます。

PERFORCE プロテクション・テーブルは、サーバ・ルート・ディレクトリの `db.protect` ファイルに保存されます。`p4 protect` が権限のないユーザによって最初に実行されたときには、このファイルを取り除くことによってディポをプロテクトされていない状態に戻すことができます。

“p4 protect” によるプロテクションの設定

`p4 protect` の設定フォームは、下図のように `Protections:` という複数行から成る単一のフォーム・フィールドで構成されており、各行にサブフィールドが設けられています。

例 : *プロテクション・テーブル例 :*

```
Protections:
read        user        emily       *             //depot/elm_proj/...
write       group       devgrp      *             //...
write       user        *           195.3.24.*    -//...
write       user        joe         *             -//...
write       user        lisag       *             -//depot/...
write       user        lisag       *             //depot/doc/...
super       user        edk         *             //...
```

（実際に表示される 5 つのサブフィールドは、縦にきれいにならんでいないかもしれません。ここでは、見やすくするために、縦にならべています。）

設定フォームの 5 つのサブフィールド

各行では、個々のパーミッションを 5 つのサブフィールドで記述して指定します。

これらのサブフィールドの意味は次のとおりです。

	意味
アクセス・レベル	list、read、open、write、review、admin、super のどのアクセス・レベルを認めるかを指定します。各レベルについては次表をご覧ください。
ユーザ / グループ	指定対象がユーザかグループかを指定します。user または group 以外は記述できません。
名前	その行のアクセス・レベルが定義されているユーザまたはグループの名前を記述します。ワイルドカード“*”も使えます。“*”とだけ記述すれば、全員にアクセス・レベルの指定が適用されます。“*e”とすれば、名前の最後が“e”で終わるすべてのユーザ（またはグループ）にアクセス・レベルの指定が適用されます。
ホスト	その行のアクセス・レベルが許可されているホストの TCP/IP アドレスを記述します。4 オクテットをピリオドで区切った表記法の数値アドレスとして記述しなければなりません（例えば、192.168.41.2）。 ワイルドカード“*”も使えます。“*”とだけ記述すれば、その行に指定されたアクセス・レベルがすべてのホストに認められます。ワイルドカードはアドレスの一部にも使え、“192.168.41.*”と記述すると、192.168.41 内のあらゆるサブネット、“*3*”は“3”を含むすべての IP アドレスを指します。 ホストの IP アドレスはインターネット・プロトコルそのもので指定するので、このフィールドはネットワークと同等のセキュリティがあります。 もし PERFORCE プロキシを使用している場合は、131 ページの「P4P のオプション」を参照し、PERFORCE プロキシ経由で PERFORCE サーバへ接続しているユーザに対して、どのようにホスト・ベースのプロテクションを使用するのか確認してください。
ファイル	アクセス・レベルを定義するディポのファイルを指定します。指定には、PERFORCE のワイルドカードも使えます。 “//...”はすべてのディポの中のすべてのファイルを意味します。

アクセス・レベル

アクセス・レベルは各行の先頭に記述します。次の 7 段階のアクセス・レベルがあります。

アクセス・レベル	意味
list	ファイルのメタデータを表示する PERFORCE コマンド（例えば p4 filelog）を実行することを認めます。ファイルの内容を参照または変更することは認めません。
read	p4 client、p4 sync など、ファイルを読むのに必要な PERFORCE コマンドの実行を認めます。read 権限には list 権限が含まれます。
open	クライアント・ワークスペースからディポのファイルを読み取り、それを開いて編集することを認めます。ただし、書き込みをしてファイルをディポに戻すことは認められていません。open は write と似ていますが、open の場合、ユーザは p4 submit や p4 lock の実行は認められていません。 open 権限には、read と list の権限も含まれます。
write	ファイルを編集、削除、追加するコマンドの実行を認めます。write 権限には、read、list、open の権限も含まれます。 この権限は、protect、depot、obliterate、verify 以外のすべての PERFORCE コマンドの実行を認めます。
review	デーモンをレビューすることを認めます。この権限には、list と read の権限のほか、p4 review コマンドの使用も認められます。このコマンドはデーモンをレビューするときにのみ必要です。

アクセス・レベル	意味
admin	PERFORCE 管理者用です。メタデータに影響する PERFORCE コマンドの実行を認めますが、サーバの動作にかかわるものは認めません。PERFORCE の write と review の権限のほかに、タイプマップ・テーブルの更新、ファイルの検証 (verify) および抹消 (obliterate)、ジョブ仕様のカスタマイズが実行でき、他のユーザのブランチ仕様、クライアント仕様、ジョブ、ラベルおよびチェンジリストをオーバライドすることができます。
super	PERFORCE スーパー・ユーザ用です。すべての PERFORCE コマンドの実行を認めます。write、review、admin の権限のほかに、ディポやトリガの生成、プロテクションやユーザ・グループの編集、ユーザの削除、パスワードのリセット、サーバの停止などの権限も認められます。

各 PERFORCE コマンドには、最低限必要なアクセス・レベルが決まっています。例えば、あるファイルに p4 sync を実行するには、ユーザは少なくともそのファイルへの read 権限を認められていなければなりません。

個々のコマンドを実行するのに必要なアクセス・レベルは、通常はそのコマンドが何をするかでわかります。例えば、p4 print が read 権限を必要とすることは、誰でも理解できるでしょう。各 PERFORCE コマンドの実行に最低限必要なアクセス・レベルの全リストについては、74 ページの「プロテクション実装のしくみ」をご覧ください。

どのユーザにどのアクセス・レベルを認めるべきか ?

最も単純なアクセス・レベルの割り当て方は、PERFORCE システムを管理する必要のないユーザ全員に write 権限、管理するユーザに super 権限を与える方法ですが、このような単純な割り当て方では不十分な場合があります。

個々のファイルについて編集する必要のないユーザは、特定ファイルに対する Read 権限を認めてもらう 方がよいでしょう 。例えば、エンジニアなら 、ソース・ ファイルに対する write 権限はあっても 、文書ファイルに対して は、read 権限だけでよいかもしれません。また、コードを扱わないマネージャも すべてのファイルについて read 権限にした方がよいかもしれません。

open 権限はファイルのローカル編集を可能にしますが、書き込みとディポへのサブミットは認めないので、特殊な状況でのみ指定します。ユーザがローカルで変更をテストしても、その変更を他のユーザに見せる必要がない場合には、write 権限より open 権限を与えます。例えば、バグのテスタは特定のバグがなぜ発生するかの理論をテストするためにコードを変更したいでしょうが、その変更がディポに書き込まれることはありません。おそらく、コードラインはそのままにしておき、開発チームによる入念なレビューを経て初めてローカルの変更はディポにサブミットされます。このような場合には、コード変更が承認されるまでは open 権限にしておき、承認後にアクセス・レベルを write に上げ、変更がサブミットされるようにするとよいでしょう。

デフォルト設定

p4 protect が呼び出されるまでは全ユーザがスーパー・ユーザの特権をもっています。p4 protect が最初に実行されると、デフォルトで 2 つのパーミッションが設定されます。デフォルトのプロテクション・テーブルは次のように表示されます。

```
write        user        *        *        //...
super        user        edk      *        //...
```

これは、全ホストの全ユーザに全ファイルへの write 権限を認め、最初に p4 protect を呼び出したユーザ（この場合は edk）にスーパー・ユーザの特権を認めることを示しています。

重複設定の解釈

複数行に設定されたユーザのアクセス・レベルは、ユーザ名やクライアント IP アドレスが合致する行の設定の組み合わせで定義されます。(排他的プロテクションはやや事情が異なりますが、それについては次のセクションで説明します。)

例 : *重複設定の解釈*

ライザは PERFORCE ユーザ名が `lisag` *で、IP アドレス* `195.42.39.17` *のワークステーションを操作しています。今、プロテクション・ファイルが次のように表示されたとします。*

```
read        user    *       195.42.39.17    //...
write       user    lisag   195.42.39.17    //depot/elm_proj/doc/...
read        user    lisag   *               //...
super       user    edk     *               //...
```

ライザには最初の 3 行が適用されます。ユーザ名は `lisag` *で、現在は 1 行目と 2 行目に指定されたホストのクライアント・ワークスペースを使用しています。つまり、ディポのサブディレクトリ* `elm_proj/doc` *のファイルには書き込みできますが、他のファイルは読むことしかできません。ライザは以下を試みます。*

`p4 edit //depot/elm_proj/doc/elm-help.1` *と入力すると、受け付けられます。*

`p4 edit //depot/elm_proj/READ.ME` *と入力すると、正規のパーミッションがないと言われます。*`read` *権限しかないファイルに書き込みをしようとしているのです。*`p4 sync //depot/elm_proj/READ.ME` *と入力すると、受け付けられます。必要なのは* `read` *権限だけで、これは 1 行目で認められているからです。*

ライザはさらに IP アドレス `195.42.39.13` *の別のマシンに乗り換えます。*`p4 edit //depot/elm_proj/doc/elm-help.1` *と入力すると、拒否されます。そのマシンを使用中には、3 行目の設定しか該当しないので、*`read` *権限しかないのです。*

排他的プロテクション

ファイルのフィールドの先頭にマイナス (-) を記述すれば、ユーザのアクセスを拒否することができます。これは特定のファイルへのアクセスを大半のユーザには認めながら、ごく一部のユーザがそのファイルにアクセスするのは拒否するような場合に有用です。

排他的マッピングを正しく活用するには、次のようなその特性を理解する必要があります。

- プロテクション・テーブルの中に排他的プロテクションが含まれている場合には、上下の順番が意味を持ちます。排他的プロテクションは、テーブル内のその行より上の関連する設定をすべて無効にします。
- 排他的プロテクションの設定行には、いかなるアクセス・レベルが指定されていようと、指定したファイルおよび IP アドレスに関するすべてのアクセスが拒否されます。排他的プロテクションの設定行のアクセス・レベルの指定は意味がありません。詳細については 74 ページの「プロテクション実装のしくみ」を参照してください。

例 : *排他的プロテクション*

管理者は `p4 protect` *を用いて次のようにプロテクションを設定しました。*

```
write       user        *           *       //...
read        user        emily       *       //depot/elm_proj/...
super       user        joe         *       -//...
list        user        lisag       *       -//...
write       user        lisag       *       //depot/elm_proj/doc/...
```

1 行目は一見、全ユーザに全ディポの全ファイルへの write 権限を認めているようですが、下の排他的プロテクションによって、一部のユーザは除外されています。

3 行目はジョーについて、どのホストからのどのファイルへのアクセスも拒否します。以下の行もジョーには何らのアクセスも認めていません。このため、ジョーは実質的に PERFORCE からロック・アウトされていることになります。

4 行目はライザのあらゆるホストからのあらゆるファイルへのアクセスを禁じていますが、5 行目が新たに彼女に特定のディレクトリの全ファイルへの `write` *権限を認めています。4 行目と 5 行目が入れ替わると、ライザは PERFORCE のどのコマンドも実行できなくなります。*

どの行がどのユーザまたはファイルに適用されるのか

`p4 protects` コマンドを使用して、あるユーザやグループまたはファイル・セットに適用されるプロテクション・テーブルの行を表示することができます。

オプションを指定しない場合、`p4 protects` は現在のユーザに適用されるプロテクション・テーブル内の行を表示します。*`file`* 引数が指定された場合、そのファイルに適用されるプロテクション・テーブル内の行のみが表示されます。`-m` オプションを使用すると、適用可能な最大アクセスレベルの概要が一語で表示されます。

PERFORCE スーパーユーザは `p4 protects -a` を使用して全ユーザの行を表示したり、`p4 protects -u` *`user`*、`-g` *`group`*、または `-h` *`host`* の各オプションを使用して特定のユーザ、グループ、ホスト IP アドレスの行を表示したりすることができます。

グループのアクセス権の設定

PERFORCE *グループ*はプロテクション・テーブルの管理を容易にします。同じような権限を必要とするユーザの名前を 1 つのグループに保存できます。その上でそのグループ名をテーブルに記述すれば、そのグループのユーザ全員が指定の権限をもちます。

グループは `p4 group` で管理し、プロテクションは `p4 protect` で割り当てます。これらのコマンドは、PERFORCE スーパー・ユーザだけが使えます。

グループの生成と編集

`p4 group` *`groupname`* を未使用の *`groupname`* で呼び出すと、*`groupname`* のついた新しいグループが作成されます。`p4 group` を既存の *`groupname`* で呼び出すと、そのグループのユーザ・リストの編集が可能になります。

グループにユーザを追加するには、`p4 group` *`groupname`* コマンドによって生成されたフォームの `Users:` フィールドにユーザ名を追加します。ユーザ名は `Users:` フィールドのヘッダの下に入力します。各ユーザ名は別々の行にインデントを付けて入力します。1 つのユーザ名をいくつのグループに登録しても構いません。

グループは個別のユーザだけでなく他のグループを含むこともできます。既存のグループのユーザ全員を作業中のグループに追加するには、`p4 group` のフォームの `Subgroups:` フィールドにそのグループ名を入れます。ユーザ名とグループ名は別々のネームスペースを占めるため、グループとユーザは同じ名前を持つことができます。

グループとプロテクション

`p4 protect` のフォームでグループを使用するときは、プロテクション・テーブルの当該行にユーザ名ではなくグループ名を指定し、その行の 2 番目のサブフィールドを `user` ではなく `group` にします。そのグループのユーザ全員に指定の権限が認められます。

例 : *PERFORCE グループの権限の設定*

下のプロテクション・テーブルはグループ `devgrp` *のメンバー全員に* `list`、*ユーザ* `edk` *に* `super` *の権限を認めています。*

```
list        group       devgrp      *       //...
super       user        edk         *       //...
```

ユーザが複数のグループに属していれば、設定と設定がかち合うことがあります。例えば、除外マッピングを使用して、`group1` のメンバーに対してディポのある領域へのアクセスを拒否し、`group2` のメンバーに対してはディポの同じ領域へのアクセスを許可した場合、`group1` と `group2` の両方のメンバーであるユーザは、プロテクション・テーブル内でどちらの行が後に記述されているかによってアクセスの可否が決まります。現実にユーザに認められる権限は、そのユーザが属するすべてのグループの名前をそのユーザの名前に置き換え、本章で先に説明した複数設定の解釈の仕方を当てはめることによって判断できます。

グループの削除

グループを削除するには、次のコマンドを使います。

```
p4 group -d groupname
```

`p4 group` *`groupname`* で特定のグループのユーザ・リストを呼び出し、すべてのユーザを削除することもできます。そのグループは、ユーザ・リストを閉じると削除されます。

プロテクション実装のしくみ

このセクションでは、PERFORCE がそのプロテクション・スキームを実行するために従うアルゴリズムを説明します。プロテクションはこのセクションを読まなくてもを適切に使用できます。ここで紹介するのはプロテクション実行の背後にあるロジックです。

ファイルに対するユーザの権限は以下の手順により決定されます。

- あるコマンドを実行するのに最低限必要なアクセス・レベルを決定するには、アクセス・レベル・テーブル（75 ページの「PERFORCE コマンドが必要とするアクセス・レベル」参照）でそのコマンドを検索します。下記の例では、`p4 print` が実行するコマンドで、そのコマンドを実行するために最低限必要なアクセス・レベルは `read` です。
- PERFORCE はプロテクション・テーブルに対する 2 回の走査のうちの 1 回目を行います。どちらの走査でもプロテクション・テーブルを下から上へ見ていき、最初の該当行をさがします。

  最初の走査はユーザがファイルの有無を知ることを許されているかどうかを判断するために行われます。この検索では、単純にユーザ名、ホスト IP アドレス、およびファイルの引数が合致する最初の行をさがします。最初に見つかった合致行が包含的なプロテクションであれば、ユーザは少なくともファイルを list するパーミッションを持っているので、PERFORCE は 2 回目の走査に移ります。逆に、最初に見つかった合致行が排他的マッピングになっていた場合、または合致する行が見つからないままプロテクション・テーブルの一番上まで行った場合には、そのユーザはファイルの list 権限もなく、`File not on client` のようなメッセージが表示されます。

例 : *プロテクション・テーブルの走査*

プロテクション・テーブルが次のようになっているものとします。

```
write       user        *       *       //...
read        user        edk     *       -//...
read        user        edk     *       //depot/elm_proj/...
```

エドが `p4 print //depot/file.c` *を実行すると、PERFORCE はプロテクション・テーブルを下から上へ調べ始め、まず最終行にぶつかります。ここに指定されたファイルはエ*

ドが印刷したいファイルとは異なるため、この行は関係ありません。次に、下から 2 行目を調べます。この行はエドのユーザ名、IP アドレス、印刷したいファイルと合致していますが、排他的マッピングなので、エドはファイルを list することを許されません。

- 1 回目の走査が成功したら、2 回目の走査が行われます。この走査も 1 回目の走査と同じですが、今回はアクセス・レベルが考慮されます。

  最初にユーザ名、IP アドレス、ファイルの引数が合致した行が包含的なプロテクション行で、指定されたアクセス・レベルが与えられたコマンドの実行に必要なレベル以上であれば、ユーザにはそのコマンドを実行するパーミッションが与えられます。

  最初にユーザ名、IP アドレス、ファイルの引数が合致した行が排他的なプロテクション行の場合、あるいは合致する行が見つからないままプロテクション・テーブルの一番上まで行った場合には、そのユーザにはコマンド実行のパーミッションは与えられず、“`You don't have permission for this operation`”などのメッセージが表示されます。

PERFORCE コマンドが必要とするアクセス・レベル

下の表は各コマンドの実行に最低限必要なアクセス・レベルを示しています。例えば、`p4 add` は少なくとも `open` 権限を必要とするので、`open`、`write`、`admin` または `super` 権限があれば、`p4 add` を実行できます。

コマンド	アクセス・レベル	コマンド	アクセス・レベル
add	open	jobspec [b]	admin
admin	super	label [a e]	open
annotate	read	labels [a b]	list
branch [e]	open	labelsync	open
branches	list	license	super
change [e]	open	lock	write
changes [a]	list	login	list
client [e]	list	logout	list
clients	list	monitor	list [f]
counter [c]	review	obliterate	admin
counters	list	opened	list
delete	open	passwd	list
depot [a b]	super	print	read
depots [a]	list	protect [a]	super
describe	read	protects [g]	list
describe -s	list	reopen	open
diff	read	resolve	open
diff2	read	resolved	open
dirs	list	revert	open
edit	open	review [a]	review
filelog	list	reviews [a]	list
files	list	set	list
fix	open	sizes	list

コマンド	アクセス・レベル	コマンド	アクセス・レベル
fixes [a]	list	submit	write
fstat	list	sync	read
group [b]	super	tag	open
groups [a]	list	tickets	none
have	list	triggers	super
help	none	typemap	admin [b]
info	none	unlock [e]	open
integrate [d]	open	user [a b]	list
integrated	list	users [a]	list
job [b e]	open	verify	admin
jobs [a]	list	where [a]	none

a このコマンドは特定のファイルに作用するものではありません。つまり、ユーザがディポの少なくとも 1 つのファイルに対して権限を指定されていれば、コマンド実行のパーミッションは認められています。
b このコマンドの -o オプションは、フォームの編集ではなく読み取りを可能にするもので、list 権限さえあれば十分です。
c 既存のカウンタの値を参照するには、list 権限が必要です。カウンタの値を変更したり、新しいカウンタを生成したりするには、review 権限が必要です。
d p4 integrate を実行するには、反映先ファイルに対する open 権限と反映元ファイルに対する read 権限が必要です。
e 既存のメタデータや他のユーザのデータをオーバライドするための -f オプションには、admin 権限が必要です。
f プロセスを終了または消去するには super 権限が必要です。引数を参照するには admin 権限が必要です。
g p4 protects で -a オプションおよび -u オプションを使用するには、super 権限が必要です。

p4 describe など、ファイルをリストするコマンドは、ユーザが少なくとも list 権限を持つファイルだけをリストします。

一部のコマンド（例えば、事前にサブミットされたチェンジリストを編集するときの p4 change）は、PERFORCE スーパー・ユーザにしか実行できない -f オプションを伴います。詳しくは 49 ページの「–f オプションによる強制動作」をご覧ください。

第 5 章

PERFORCE のカスタマイズ：ジョブ仕様

PERFORCE のジョブでは、チェンジリストを機能拡張要求、問題報告、その他ユーザ定義タスクにリンクさせることができます。また、PERFORCE はサード・パーティの欠陥追跡ツールを PERFORCE と連携させる手段として、P4DTI（PERFORCE Defect Tracking Integration）も提供します。詳しくは 84 ページの「サード・パーティの欠陥追跡システムとの連携」をご覧ください。

PERFORCE ユーザによる `p4 job` の使用方法については、『コマンドライン・ユーザーズ・ガイド』をご覧ください。本章では、管理者によるジョブ・システムの変更について説明します。

PERFORCE のデフォルト・ジョブ・テンプレートには、ジョブ追跡用のフィールドが 5 つあります。PERFORCE で管理するプロジェクトの規模が小さいうちは、これらのフィールドだけで十分ですが、プロジェクトの規模が大きくなってくると、これらのフィールドの情報だけでプロジェクトを管理することが困難になる場合があります。ジョブ・テンプレートの内容は、`p4 jobspec` コマンドを使用して変更することができます。`p4 jobspec` は、PERFORCE 管理者だけが使用できるコマンドです。

本章では、PERFORCE ジョブ・テンプレートの変更のしくみを説明します。

> *PERFORCE ジョブ・テンプレートへの不適切な変更は、サーバ・データベース破損の原因になる恐れがあります。本章の章末に記載されたジョブ・テンプレートへの変更に関する警告、注意、勧告をご覧ください。*

PERFORCE のデフォルト・ジョブ・テンプレート

PERFORCE のジョブがどのように指定されるかを理解するために、PERFORCE のデフォルト・ジョブ・テンプレートについて考えてみましょう。この章で示されている例は、このテンプレートを適宜修正して作成されています。

PERFORCE のデフォルト・ジョブ・テンプレートを使用すると、以下のフォーマットのジョブが生成されます。

```
# A PERFORCE Job Specification.
#
#  Job:         The job name.  'new' generates a sequenced job number.
#  Status:      Either 'open', 'closed', or 'suspended'. Can be changed.
#  User:        The user who created the job. Can be changed.
#  Date:        The date this specification was last modified.
#  Description: Comments about the job.  Required.
Job:    new
Status: open
User:   edk
Date:   1998/06/03 23:16:43
Description:
        <enter description here>
```

このジョブの生成に使用したテンプレートは、`p4 jobspec` により表示して編集することができます。デフォルト・ジョブ・テンプレートは、以下のように表示されます。

```
# A PERFORCE Job Specification.
#
#  Updating this form can be dangerous!
#  See 'p4 help jobspec' for proper directions.
Fields:
        101 Job word 32 required
        102 Status select 10 required
        103 User word 32 required
        104 Date date 20 always
        105 Description text 0 required
Values:
        Status open/suspended/closed
Presets:
        Status open
        User $user
        Date $now
        Description $blank
Comments:
        # A PERFORCE Job Specification.
        #
        # Job: The job name. 'new' generates a sequenced job number.
        # Status: Either 'open', 'closed', or 'suspended'. Can be changed.
        # User: The user who created the job. Can be changed.
        # Date: The date this specification was last modified.
        # Description: Comments about the job. Required.
```

ジョブ・テンプレートのフィールド

`p4 jobspec` によって表示されるジョブ・テンプレートには、以下の 4 つのフィールドがあります。サーバに保存されているすべての PERFORCE ジョブのテンプレートは、これらのフィールドにより定義されます。以下の表に、フィールドおよびフィールド・タイプを示します。

	意味
`Fields:`	各ジョブに含めるフィールドのリストです。 各フィールドは ID 番号、名前、データタイプ、長さ、属性で構成されます。
`Values:`	データタイプが `select` のフィールドのリストです。 各 `select` データタイプのフィールドについて、フィールド名、スペース、およびスラッシュで区切った選択肢のリストからなる 1 行を追加しなければなりません。
`Presets:`	各フィールドと、そのデフォルト値のリストです。 デフォルト値は、PERFORCE によってサポートされている変数または固定文字列のいずれかです。
`Comments:`	`p4 job` フォームの先頭に表示されるコメントです。 このコメントは PERFORCE の Windows クライアント P4Win によっても使用されます。

Fields: フィールド

`p4 jobspec` の `Fields:` フィールドには、ジョブで追跡するフィールドのリストが表示され、`p4 job` のフォーム上に出現する順序を定義します。

デフォルトの `Fields:` フィールドには、以下のフィールドが含まれます。

```
Fields:
        101 Job word 32 required
        102 Status select 10 required
        103 User word 32 required
        104 Date date 20 always
        105 Description text 0 required
```

警告！ *101 ～ 105 のフィールドに対しては、変更、リネーム、再定義はしないでください。これらのフィールドは PERFORCE が使用するので、削除も変更もしてはいけません。`p4 jobspec` は、ジョブに新しいフィールド（すなわち、106 以降）を追加する目的のみに使用してください。*

- フィールド 101 は PERFORCE に必要です。リネームすることも削除することもできません。
- フィールド 102 ～ 105 は、PERFORCE のクライアント・プログラム専用です。これらのフィールドをリネームまたは削除することは可能ですが、望ましくありません。PERFORCE クライアント・プログラムは、チェンジリストがサブミットされるたびにフィールド 102（`Status:` フィールド）の値を `closed` に設定します。これは、システム管理者がフィールド 102 を、`closed` を許容値として含まないフィールドとして使用するように再定義しても変わりません。むしろ、このような再定義は予測不可能な混乱を生じる危険性があります。

各追跡対象フィールドは、1 つずつ別の行に、以下の 5 つの記述子で記述します。

	意味
ID 番号	各追跡対象フィールドのインデックスとなる固有の整数識別子です。フィールドが生成されジョブがシステムに入力された後も、フィールド名は変更可能です。ただし、ID 番号を変更すると、そのフィールドのデータにアクセスすることができなくなります。 指定できる ID 番号の範囲は、`106` ～ `199` です。
名前	`p4 job` フォームに表示する追跡対象フィールドの名前です。
データタイプ	次表に記述した 5 つのデータタイプ（`word`、`text`、`line`、`select`、`date`）のいずれかを示します。
長さ	PERFORCE の Windows クライアント P4Win に表示される追跡対象フィールドのテキスト・ボックスの推奨サイズです。複数行入力できるテキスト・ボックスを表示する場合は、`0` と指定します。1 行のテキストボックスにする場合は、最大入力文字数を指定します。 この値は PERFORCE のコマンドラインから編集するジョブには影響せず、サーバに保存された実際の値の長さとも関係ありません。
属性	追跡対象フィールドが読み取り専用か否か、デフォルト値が使用されているか否かなどを指定します。以下の指定が可能です。 ・ `optional`: 任意の値を指定することができます。削除も可能です。 ・ `default`: デフォルト値が与えられます。変更または消去が可能です。 ・ `required`: デフォルト値が与えられます。変更はできますが、空白にしておくことはできません。 ・ `once`: 読み取り専用です。最初にデフォルト値に設定され、以後変更はできません。 ・ `always`: 読み取り専用です。ジョブを保存するたびに、値がデフォルト値にリセットされます。変数 `$now` でジョブ更新日を変更するとき、変数 `$user` でジョブを最後に更新したユーザの名前を変更するときにのみ有用です。 PERFORCE バージョン 98.2 と本バージョンとでは、属性の指定形式が異なります。バージョン 98.2 から本バージョンにアップグレードした場合には、`p4 jobspec` によりバージョン 98.2 の属性指定は自動的に新しい指定形式に変換されます。

前記のフィールドデータタイプの意味は、以下のとおりです。

データタイプ	説明	例
`word`	1 語です。	ユーザ ID: `edk`
`text`	1 行または複数行のテキストブロックです。	ジョブの記述。
`line`	1 行以内のテキストです。	ユーザの実名 : `Ed K.`
`select`	複数の選択肢のうちの 1 つです。 このデータタイプのフィールドは、対応する選択肢が`Values:`フィールドに指定されていなければなりません。	ジョブの状態。次のいずれか : `open/suspended/closed`
`date`	日付です。	ジョブ生成の日付と時刻 : `1998/07/15:13:21:46`

Values: フィールド

データタイプが `select` のフィールドごとに、選択肢を 1 行で記述します。各行には、追跡対象フィールドの名前とスペース 1 個に続けて、選択肢をスラッシュで区切って記述します。

PERFORCE のデフォルト・ジョブ仕様では、Status: フィールドのデータタイプだけが select になっていて、その選択肢は以下のように設定されています。

```
Values:
        Status open/suspended/closed
```

> **注** *PERFORCE バージョン 2000.1 とそれより前のバージョンとでは、Values: フィールドと Presets: フィールドの指定形式が異なります。*
>
> *旧ジョブ仕様に基づくスクリプトは修正を要する場合があります。(ジョブに対するスクリプトが、ジョブ仕様には関わりのない場合は修正不要です。)*

Presets: フィールド

属性が optional 以外のフィールドは、すべてデフォルト値を必要とします。フィールドにデフォルト値を割り当てるには、ジョブ・テンプレートの Presets: の下にフィールド名と対応するデフォルト値を 1 行で記述します。いかなる 1 行の文字列もデフォルト値として使用することができます。

デフォルト値として、以下の 3 つの変数を使用することができます。

変数	値
$user	ジョブを生成しようとしている PERFORCE ユーザです。P4USER の環境設定またはレジストリ変数で指定するか、p4 -u *username* ジョブでオーバーライド指定します。
$now	ジョブを保存するときの日付と時刻です。
$blank	テキスト <enter description here> です。 ユーザによるジョブ入力時、ジョブ・テンプレートでプリセット値が $blank に設定されているフィールドがある場合、ジョブをシステムに追加する前に、そのフィールドに必要な記述を入力しておかなければなりません。

標準的なジョブ・テンプレートの Presets: フィールドは、以下のようになります。

```
Presets:
        Status open
        User $user
        Date $now
        Description $blank
```

Comments: フィールド

Comments: フィールドには、p4 job フォームの先頭に表示されるコメントを記述します。p4 job は、データタイプが select になっているフィールドに指定できる値や、どのフィールドに記述が必要かなどの情報をユーザに通知しません。したがって、ユーザが必要とするすべての情報を Comments: フィールドに記述しておく必要があります。

Comments: フィールドの各行は左端から少なくとも 1 タブはインデントさせてコメント・キャラクタの # で始めます。

`p4 job` のデフォルト・ジョブ・テンプレートの `Comments:` フィールドは、以下のように表示されます。

```
Comments:
        # A PERFORCE Job Specification.
        # Job: The job name. 'new' generates a sequenced job number.
        # Status: Either 'open', 'closed', or 'suspended'. Can be changed
        # User: The user who created the job. Can be changed.
        # Date: The date this specification was last modified.
        # Description: Comments about the job. Required.
```

PERFORCE の Windows クライアント P4Win を使用しているユーザがいる場合、PERFORCE サーバの管理においてコメントの記述に特別な注意が必要です。

P4Win では、以下の 2 つの方法でコメントを表示します。

- P4Win ユーザがジョブの生成時または編集時にジョブ・ダイアログの Form Info ボタンを押すと、ポップアップ・ウィンドウにコメントが表示されます。
- (Windows の) カーソルが各フィールドの上に行くと、ジョブ仕様に記述された該当フィールド名とコロンに続くコメントの 1 行目がツール・チップに表示されます。実際にツール・チップとして表示されるのは、各行の先頭のフィールド名を除く残りの部分です。

  例えば、先のジョブ仕様の `Status:` フィールドのツール・チップは、以下のように表示されます。

  ```
  Either 'open', 'closed', or 'suspended'. Can be changed.
  ```

警告、注意、勧告

このセクションの内容は、すでに本章のほかの箇所でもお伝えしていますが、重要なことなので、繰り返しお伝えしておきます。`p4 jobspec` でジョブ仕様を編集するときは、ここに示す指針に従ってください。

> **警告！** *ジョブ仕様の編集を開始する前に、必ずこのセクションの内容を読んで理解してしてください。*

- 101 ～ 105 のフィールドに対しては、変更、リネーム、再定義はしないでください。これらのフィールドは PERFORCE が使用するので、削除も変更もしてはいけません。`p4 jobspec` は、ジョブに新しいフィールド（すなわち、106 以降）を追加する目的のみに使用してください。
  - フィールド 101 は PERFORCE に必要です。リネームすることも削除することもできません。
  - フィールド 102 ～ 105 は、PERFORCE のクライアント・プログラム専用です。これらのフィールドをリネームまたは削除することは可能ですが、望ましくありません。PERFORCE クライアント・プログラムは、チェンジリストがサブミットされるたびにフィールド 102 (`Status:` フィールド）の値を `closed` に設定します。これは、システム管理者がフィールド 102 を、`closed` を許容値として含まないフィールドとして使用するように再定義しても変わりません。むしろ、このような再定義は予測不可能な混乱を生じる危険性があります。
- フィールドが生成され、ジョブが入力された後は、そのフィールドの ID 番号を変更しないでください。`p4 job` でそのフィールドに入力したデータにアクセスできなくなります。
- フィールドの名前はいつでも変更可能です。フィールド名を変更するときは、`p4 jobspec` の他のフィールドに記述された同じフィールド名も忘れずに変更してください。例えば、`severity` と名付けたフィールド `106` を生成した後、フィールド名を `bug-severity` に変

更する場合は、ジョブ仕様の `Presets:` フィールドの対応する行に記述された同じフィールド名も、`bug-severity` に変更します。

- `Comments:` フィールドに記述するコメントは、ユーザに各フィールドの要件を知らせる唯一の手段です。これらのコメントはわかりやすく、完全なものにしてください。特に、PERFORCE の Windows クライアント P4Win では、このコメントがツール・チップに利用されます。各フィールドのコメントの1行目は、それだけで読んで分かるものにしてください。

例 : カスタム・テンプレート

より複雑なジョブ仕様と、その仕様に基づき表示されるジョブ・フォームを以下に示します。

```
# A Custom Job Specification.
#
#  Updating this form can be dangerous!
#  See 'p4 help jobspec' for proper directions.
Fields:
        101 Job word 32 required
        102 Status select 10 required
        103 User word 32 required
        104 Date date 20 always
        111 Type select 10 required
        112 Priority select 10 required
        113 Subsystem select 10 required
        114 Owned_by word 32 required
        105 Description text 0 required
Values:
        Status open/closed/suspended
        Type bug/sir/problem/unknown
        Priority A/B/C/unknown
        Subsystem server/gui/doc/mac/misc/unknown
Presets:
        Status open
        User setme
        Date $now
        Type setme
        Priority unknown
        Subsystem setme
        Owned_by $user
        Description $blank
Comments:
        # Custom Job fields:
        # Job:    The job name. 'new' generates a sequenced job number.
        # Status: Either 'open', 'closed', or 'suspended'. Can be changed
        # User:   The user who created the job. Can be changed.
        # Date:   The date this specification was last modified.
        # Type:         The type of the job. Acceptable values are
        #               'bug', 'sir', 'problem' or 'unknown'
        # Priority:     How soon should this job be fixed?
        #               Values are 'a', 'b', 'c', or 'unknown'
        # Subsystem:    One of server/gui/doc/mac/misc/unknown
        # Owned_by:     Who's fixing the bug
        # Description:  Comments about the job. Required.
```

ジョブ・フォームで表示されるフィールドの順番は、`p4 jobspec` フォームの `Fields:` フィールドでのフィールド記述順によって決まります。フィールドは ID 番号順に並んでいる必要はありません。

例のカスタム・ジョブ仕様で p4 job を実行すると、以下のようなジョブ・フォームが表示されます。

```
# Custom Job fields:
# Job:    The job name. 'new' generates a sequenced job number.
# Status: Either 'open', 'closed', or 'suspended'. Can be changed
# User:   The user who created the job. Can be changed.
# Date:   The date this specification was last modified.
# Type:          The type of the job. Acceptable values are
#                'bug', 'sir', 'problem' or 'unknown'
# Priority:      How soon should this job be fixed?
#                Values are 'a', 'b', 'c', or 'unknown'
# Subsystem:     One of server/gui/doc/mac/misc/unknown
# Owned_by:      Who's fixing the bug
# Description: Comments about the job. Required.

Job:    new
Status: open
User:   setme
Type:   setme
Priority:       unknown
Subsystem:      setme
Owned_by:       edk
Description:
       <enter description here>
```

サード・パーティの欠陥追跡システムとの連携

P4DTI を使用し、PERFORCE をサード・パーティの欠陥追跡システムまたはプロセス管理ソフトウェアと連携させることができます。

PERFORCE ディポでの作業 – 機能拡張、バグ修正、リリース・ブランチへの変更の伝達などは、P4DTI によって自動的に欠陥追跡システムに入力することができます。また、欠陥追跡システムに入力された発行物や状態（バグ報告、変更命令、作業割当など）は、自動的に PERFORCE メタデータに変換し、PERFORCE ユーザによってアクセスできるようにすることができます。

P4DTI は容易に他の製品へ拡張できるように設計されています。TeamShare および Bugzilla が最初に発表された 2 つの連携製品です。

P4DTI はオープン・ソース連携であり、FreeBSD と同様のライセンスで入手可能です。

P4DTI の使用 – PERFORCE 欠陥追跡統合

PERFORCE と自社の欠陥追跡システム、またはサード・パーティの欠陥追跡システムを連携させる場合、手始めに P4DTI を起用するのが最善の策でしょう。

P4DTI 製品に関しては、下記のホームページをご覧ください。

```
http://www.perforce.com/perforce/products/p4dti.html
```

このホームページから、TeamShare および Bugzilla への実装に関する情報、P4DTI の機能の概略説明、他の製品や社内システムとの連携を行うときに必要なキット（ソース・コード、開発文書など）などを入手することができます。

> **注** *日本語版 PERFORCE は、P4DTI をサポートしておりませんのでご注意ください。*

独自の連携システムを構築する

最初に P4DTI キットを使用しない場合でも、PERFORCE ジョブ・システムを PERFORCE と欠陥追跡システムとのインターフェイスとして利用することができます。インターフェースは、アプリケーションに応じて、以下の 1 つまたは複数で構成されます。

- 欠陥追跡システム内でバグが追加、更新、削除されるたびに PERFORCE のジョブを追加、更新、削除する欠陥追跡システム側のトリガまたはスクリプト。

  サード・パーティのシステムはデータを生成し、それを `p4 job` の手動（インタラクティブな）起動用フォームに合わせて再フォーマットするスクリプトに渡します。スクリプトは、そこで生成されたフォームを `p4 job -i` コマンドの標準入力として送出することができます。

  （`p4 job` に標準入力から直接ジョブ・フォームを読み込ませた場合は、`p4 job` コマンドの `-i` オプションを使用します。これは、通常ユーザが行う対話形式の「フォーム・アンド・エディタ」方式の操作とは異なります。`-i` オプションを用いた PERFORCE の自動化に関する詳細については、『PERFORCE コマンド・リファレンス』をご覧ください。）

- 必要なバグ修正完了情報を収集するために、サブミットされたチェンジリストをチェックする PERFORCE 側のトリガ。
- ジョブ修正完了のチェンジリストが無事にサブミットされたことをチェックし、欠陥追跡システムの対応データを更新するための適切なコマンドを発行する PERFORCE レビュー・デーモン。

トリガおよびレビュー・デーモンについての詳細や使用例については、87 ページの「PERFORCE のスクリプト : トリガとデーモン」をご覧ください。

さらに情報を得るには

P4DTI を基本にした TeamaShare や Bugzilla との連携以外にも、PERFORCE カスタマは PERFORCE と自社製の欠陥追跡システムや、Remedy、Scopus、ClearTrack などのサード・パーティのシステムとの連携も行っています。

ほかの PERFORCE ユーザの実績に関心がある場合は、PERFORCE の Web サイトにアクセスし、`perforce-user` のメーリング・リストの過去ログをお調べください。Documentation ページにあります。

また、連携検討中のサード・パーティ・ツールをすでに連携させたことのあるユーザがいるかどうか、`perforce-user` に質問を送ることもできます。貴社で必要となる設定作業をすでに行った他のユーザが見つかるかも知れません。

第 6 章

PERFORCE のスクリプト：トリガとデーモン

PERFORCE のスクリプトには、主として次の 2 種類があります。

- PERFORCE *トリガ*は、ユーザが記述するスクリプトであり、特定の操作（チェンジリストのサブミット、フォームの変更、ユーザによるログイン試行、パスワード変更など）が実行されるたびに PERFORCE サーバによって呼び出されます。スクリプトが値 0 を返した場合は操作を続行し、スクリプトがそれ以外の値を返した場合は操作を失敗させます。失敗時、エラー・メッセージとしてスクリプトの標準出力（エラー出力ではない）が PERFORCE クライアント・プログラムに送信されます。
- *デーモン*は事前に設定された時刻に実行され、PERFORCE メタデータの変更を調べます。デーモンは、ディポの状態が何らかの指定されたかたちに変化したと判断すると、ほかのコマンドを実行します。例えば、新たにサブミットされたチェンジリストを調べ、それらのファイルに対する変更を追跡することに関心のあるユーザに、電子メールを送ることもできます。PERFORCE はデーモンを書くのを容易にする様々なツールを提供します。

本章の説明は、読者がスクリプトの書き方を知っていることを前提にしています。

トリガ

トリガはさまざまな状況で利用できます。よく使用されるのは次のような場合です。

- PERFORCE のプロテクション・テーブルによって提供される機構の範囲を超えてチェンジリストの内容を検証する場合。例えば、サブミット前トリガを使用して、チェンジリスト内の `file1` が サブミットされるたび `file2` もサブミットされるようにすることができます。
- チェンジリスト・サブミットの一部としてファイルの内容を検証する場合。例えば、サブミット中トリガを使用して、`file1` と `file2` がサブミットされた場合に両方のファイルが同じヘッダ・ファイルのセットを参照するようにすることができます。
- チェンジリスト・サブミットが成功した後にビルド・プロセスを開始する場合
- フォームを検証する場合や、カスタマイズされたバージョンの PERFORCE フォームを提供する場合。例えば、フォーム・トリガを使用して、カスタマイズされたデフォルトのワークスペース・ビューを p4 client コマンド実行時に生成したり、適切なワークスペース説明を必ず入力させるようにしたりできます。
- LDAP や Active Directory などの外部の認証メカニズムに対応するように PERFORCE を構成する場合。

- `p4 user` や `p4 job` などのコマンドによるデータの変更や削除を他のユーザに通知する場合や、PERFORCE メタデータへの更新の後にプロセス制御ツールをトリガする場合。

> *トリガ・スクリプトを使用する場合は、**ディポにデータを書き込む PERFORCE コマンドは危険であるため、避けるべきであることに注意してください。**特に、`p4 submit` をトリガ・スクリプトから**実行することは避けてください。***

例 : *基本的なトリガ*

開発グループでは、`.exe` ファイルが含まれているチェンジリストがディポにサブミットされるときは必ず、そのプログラムに対するリリース・ノートが同時にサブミットされるようにしたいと考えています。

この場合は、チェンジリストの番号を唯一の引数とし、チェンジリストに対して `p4 opened` を実行し、その結果を解析してチェンジリストに含まれるファイルを探し、サブミットされたすべての実行可能ファイルについて同じディレクトリの `RELNOTES` ファイルもサブミットされているかどうかを確かめるトリガ・スクリプトを書きます。チェンジリストに `RELNOTES` ファイルが 1 つ含まれていれば、スクリプトは完了コード `0` で終了し、それ以外の場合は、完了コードが `1` に設定されます。

スクリプトが書けたら、`p4 triggers` フォームを以下のように編集し、書いたスクリプトをトリガ・テーブルに追加します。

```
Triggers:
   rnotes  submit  //depot/....exe   "/usr/bin/test.pl %changelist%"
```

(他の PERFORCE フォームと同様に、`[Triggers:]` フィールドの下の各行をタブでインデントしてください。)

このトリガは `.exe` で終わる名前のファイルがサブミットされるたびに実行されます。失敗したら 0 以外の完了コードを返し、ユーザのサブミットは失敗します。

トリガ・テーブル

トリガ・スクリプトを書いた後、`p4 triggers` コマンドを実行してトリガを作成します。`p4 triggers` フォームは次のようになります。

```
Triggers:
   relnotes_check  submit  //depot/bld/... "/usr/bin/relcheck.pl %user%"
   verify_jobs     submit  //depot/...     "/usr/bin/job.py %change%"
```

フォームを使用するすべての PERFORCE コマンドと同様に、フィールド名（`Triggers:` など）は左端で始め（インデントしない）、最後にコロンを付ける必要があり、フィールド値（トリガごとに 1 行ずつ追加する、行の並び）はフィールド名の下の行に空白またはタブでインデントする必要があります。

`p4 triggers` を実行するユーザは、PERFORCE スーパー・ユーザでなければなりません。

トリガ・テーブルのフィールド

トリガ・テーブルの各行には以下の 4 つのフィールドがあります。

フィールド	意味
`name`	ユーザ定義のトリガ名。 連続する行に同一のトリガ名がある処理は、複数の ***`path`*** が指定できるように単一のトリガとして扱われます。この場合、そのようなトリガ行の最初の ***`command`*** のみが使用されます。
`type`	トリガ・タイプは 12 種類あり、4 つのサブタイプ（チェンジリスト・サブミット・トリガ、修正トリガ、フォーム・トリガ、認証トリガ）に分類されます。 チェンジリスト・サブミット・トリガ ・ `change-submit` : チェンジリストの作成後、ファイル転送の前にチェンジリスト・トリガを実行します。トリガ・スクリプトはファイルの内容にアクセスできません。 ・ `change-content` : チェンジリストの作成およびファイルの転送後、ファイルのコミット前にチェンジリスト・トリガを実行します。 ファイルの内容を取得するには、リビジョン指定子 @=change を指定して `p4diff2`、`p4 files`、`p4 fstat`、`p4 print` などのコマンドを使用します。この場合、`change` は、`%changelist%` 変数でスクリプトに渡される作業中チェンジリストのチェンジリスト番号です。 ・ `change-commit` : チェンジリスト作成、ファイル転送、およびファイル・コミットの後に、チェンジリスト・トリガを実行します。 修正トリガ 特殊変数 `%jobs%` を使用して拡張が可能です。この変数は `p4 fix` コマンドライン（あるいは `p4 change` フォームや `p4 submit` フォームの `[Jobs:]` フィールド）にリスト表示されるすべてのジョブに対して 1 つずつ引数を拡張します。そのため、トリガ・スクリプトに指定される最後の引数でなければなりません。 ・ `fix-add` : 修正を追加する前に修正トリガを実行します。 ・ `fix-delete` : 削除を追加する前に修正トリガを実行します。

フィールド	意味
	フォーム・トリガ ・ `form-save` : フォームの内容が解析された後、その内容が PERFORCE データベースに保存される前に、フォーム・トリガを実行します。トリガは、`%formfile%` 変数で指定されたフォームを変更できません。 ・ `form-out`: エンド・ユーザへのフォームの生成時にフォーム・トリガを実行します。トリガはフォームを変更できます。 ・ `form-in`: PERFORCE サーバによって内容が解析され、検証される前に、編集済みフォームに対してフォーム・トリガを実行します。トリガはフォームを変更できます。 ・ `form-delete`: フォームの内容が解析された後、その内容が PERFORCE データベースから削除される前に、フォーム・トリガを実行します。トリガはフォームを変更できません。 ・ `form-commit`: フォーム・トリガがコミットされた後それを実行し、ジョブ名、日付などの自動生成フィールドへのアクセスを有効にします。ジョブ・フォームの場合、このトリガは `p4 fix` コマンドと同様 `p4 job` コマンドでも実行されます（ステータス更新後）。`form-commit` トリガは `p4 job` によって作成された新しいジョブ名へのアクセス権を持っています。`form-in` トリガと `form-save` トリガはジョブ名が作成される前に実行されます。 ジョブ・フォームの場合、このトリガは `p4 change` コマンド（ジョブがチェンジリストの `[Jobs:]` フィールドの編集によって追加または削除されている場合）、および `p4 submit` コマンド（チェンジリストの `[Jobs:]` フィールドに存在するすべてのジョブに対して）によっても実行されます。これらの場合は、ジョブの `form-commit` トリガ・コマンドラインで特殊変数 `%action%` を使用した拡張が可能です。このトリガはフォームを変更することはできません。 認証トリガ : ・ `auth-check` : 認証チェック・トリガを実行し、ログイン時または新規パスワードの設定時にパスワードを外部パスワード管理機構に対して検証します。`auth-check` トリガが存在する場合、認証はトリガ・スクリプトにより制御されるため、PERFORCE`security` カウンタ（および関連するパスワードの長さについての必要条件）は無視されます。 ・ `auth-set` : 認証セット・トリガを実行し、新規パスワードを外部パスワード管理機構に送信します。 `auth-check` トリガを追加したら、PERFORCE サーバを再起動しなければなりません。
`path`	チェンジリスト・サブミット・トリガ (`change-submit`、`change-content`、および `change-commit`) の場合は、ディポ・シンタックスのファイル・パターンです。このファイル・パターンと一致するファイルが含まれているチェンジリストをユーザがサブミットしたときに、このトリガにリンクされたスクリプトが実行されます。除外マッピングを使用すると、指定したファイルに対してトリガが実行されるのを防ぐことができます。 修正トリガ (`fix-add` または `fix-delete`) の場合は、修正をパスの値として使用します。 フォーム・トリガ (`form-save`、`form-out`、`form-in`、`form-commit` または `form-delete`) の場合は、フォームのタイプの名前 (`branch`、`change`、`client`、`depot`、`group`、`job`、`label`、`protect`、`spec`、`triggers`、`typemap`、または `user` のうちどれか) です。`p4 triggers` コマンドに対して起動するトリガは無視されます。 認証トリガ (`auth-check` または `auth-set`) の場合は、`path` の値に `auth` を指定してください。

フィールド	意味
command	一致する *path* がトリガ・タイプに適用されたときに、PERFORCE サーバが実行するコマンドです。PERFORCE サーバのアカウントがコマンドを見つけて実行できるように、コマンドを指定します。コマンドは、引用符で囲む必要があります。また、91 ページの「トリガ・スクリプトの変数」に示した変数を引数に取ることができます。 change-submit トリガと change-content トリガの場合、トリガ・スクリプトが 0 で終了したときはチェンジリストのサブミットが続行され、スクリプトがゼロ以外の値で終了したときはチェンジリストのサブミットは失敗します。change-commit トリガの場合、トリガ・スクリプトの終了コードに関係なくチェンジリストのサブミットは成功します。しかし、スクリプトがゼロ以外の値で終了したときは、後続の change-commit トリガは起動しません。 form-in トリガ、form-out トリガ、form-save トリガ、form-delete トリガの場合、スクリプトが 0 で終了したときに仕様内のデータが PERFORCE データベースに組み込まれます。ゼロ以外の値で終了したときは、データベースは更新されません。 form-commit トリガ・タイプは絶対に変更を拒否しません。これは最初から存在するため、ジョブ作成処理中にスクリプトがジョブ番号に (%formname% 値から) アクセスできます。 fix-add トリガと fix-delete トリガでは、トリガ・スクリプトが 0 で終了した場合は修正の追加や削除が続行し、スクリプトがゼロ以外の値で終了したときは失敗します。 auth-check トリガ（p4 login により起動）の場合、ユーザが入力したパスワードは標準入力からトリガ・コマンドに渡されます。トリガの実行に成功すると、PERFORCE チケットが発行されます。ユーザ名はコマンドラインに %user% を渡すことにより入手できます。 auth-set トリガ（p4 passwd により auth-check トリガによる検証成功後に起動）の場合、ユーザの古いパスワードと新しいパスワードは標準入力でトリガに渡されます。ユーザ名はコマンドラインに %user% を渡すことにより入手できます。

トリガ・スクリプトの変数

command フィールドに以下の変数を使用して、トリガ・スクリプトにデータを渡します。

引数	解説	引数を使用できるタイプ
%action%	ジョブ・フォームにおいて、ヌルまたはジョブが追加または削除される（作業中またはサブミット済）チェンジリストの状態を表します。	form-commit
%changelist% %change%	サブミットするチェンジリストの番号（短縮形の %change% は %changelist% と同等です）。	change-submit、 change-content、 change-commit fix-add, fix-delete, form-commit
%client%	トリガを行うユーザのクライアント･ワークスペース名	すべて
%clienthost%	クライアントのホスト名	すべて
%clientip%	クライアントの IP アドレス	すべて
%jobs%	p4 fix コマンドで指定されたジョブ番号ごとに、または p4 submit フォームか p4 change フォームの [Jobs:] フィールドに追加（削除）されたジョブ番号ごとに、引数を 1 つずつ拡張したジョブ番号の文字列。	fix-add, fix-delete
%serverhost%	PERFORCE サーバのホスト名	すべて
%serverip%	サーバの IP アドレス	すべて

引数	解説	引数を使用できるタイプ
%serverport%	PERFORCE サーバの IP アドレスとポート。形式は *ip_address:port*	すべて
%serverroot%	PERFORCE サーバの P4ROOT ディレクトリ	すべて
%user%	トリガを実行するユーザの PERFORCE ユーザ名	すべて
%formfile%	一時フォーム仕様ファイルのパス。in トリガまたは out トリガからフォームを変更するには、このファイルを上書きします。タイプ save のトリガおよび delete のトリガの場合、このファイルは読み取り専用です。	form-save、form-out、form-in、form-delete
%formname%	フォームの名前 (ブランチ名やチェンジリスト番号など)	form-commit、form-save、form-out、form-delete
%formtype%	フォームのタイプ (branch や change など)	form-commit、form-save、form-out、form-in、form-delete

チェンジリストに対するトリガ

ユーザがチェンジリストをサブミットしたときにトリガ・スクリプトを実行するように PERFORCE を設定するには、チェンジリスト・サブミット・トリガ (タイプ change-submit、change-content、および change-commit のトリガ) を使用します。

チェンジリスト・サブミット・トリガの場合、各トリガ行の *path* 列はディポ・シンタックスのファイル・パターンです。サブミットされたチェンジリストにこのパス内のファイルが含まれている場合に、トリガが起動します。ファイルへの変更によってトリガが起動されるのを防ぐには、そのパスで除外マッピングを使用します。

チェンジ – サブミット・トリガ

change-submit トリガ・タイプを使用して、チェンジリスト作成の後、ファイルがサーバに転送される前に起動するトリガを作成します。チェンジ – サブミット・トリガは、ファイルがサーバに転送される前に起動されるので、ファイルの内容にアクセスできません。チェンジ – サブミット・トリガは、システムファイルの内容にアクセスする必要のないレポート・ツールやシステムによる反映操作で役に立ちます。

例 : *以下のチェンジ – サブミット・トリガは、サブミットを行ったユーザがチェンジリストにジョブを割り当てていない場合にそのチェンジリストを拒否する MS-DOS バッチ・ファイルです。このトリガは、*//depot/qa *ブランチ内の少なくとも 1 つのファイルに影響するチェンジリスト・サブミット試行に対してのみ実行されます。*

```
@echo off
if not x%1==x goto doit
echo Usage is %0[change#]
:doit
p4 describe -s %1|findstr "Jobs:\n\n\t" > nul
if errorlevel 1 echo No jobs found for changelist %1
p4 describe -s %1|findstr "Jobs:\n\n\t" > nul
```

このトリガを使用するには、トリガ・テーブルに以下の行を追加します。

```
sample1   submit //depot/qa/...   "jobcheck.bat %changelist%"
```

//depot/qa *の下の任意のファイルに影響を与えるチェンジリストがサブミットされるたびに、*jobcheck.bat *ファイルが呼び出されます。"*Jobs: fixed...*" という文字列*

(後ろに 2 つの改行とタブ文字が続く) が検出されると、スクリプトはジョブがチェンジリストに割り当てられていると想定し、チェンジリストのサブミットを続行することを許可します。この文字列が検出されないと、サブミットは拒否されます。

2 つめの `findstr` *コマンドは、トリガ・スクリプトの最終エラー・レベルが、エラー・メッセージを出力するかどうかを決定するエラー・レベルと同じであることを確認します。*

チェンジ – コンテンツ・トリガ

`change-content` トリガ・タイプを使用して、チェンジリスト作成とファイル転送の後、データベースにサブミットをコミットする前に、起動されるトリガを作成します。チェンジ – コンテンツ・トリガは、`@=`*change* リビジョン指定子を指定して `p4 diff2`、`p4 files`、`p4 fstat`、および `p4 print` の各コマンドを使用してファイルの内容にアクセスします。この場合、`change` は、`%changelist%` 変数でトリガ・スクリプトに渡される作業中チェンジリストの番号です。

チェンジ – コンテンツ・トリガを使用して、チェンジリスト・サブミットの一部としてファイルの内容を検証し、検証に失敗した場合はチェンジリスト・サブミットを中断します。

例 : *以下のチェンジ – コンテンツ・トリガは、すべてのチェンジリスト内のすべてのファイルに現年度の著作権情報が含まれていることを確認する Bourne シェル・スクリプトです。*

このスクリプトは、`//depot/src` *のすべてが含まれた* `copychecker` *と呼ばれるクライアント・ワークスペースが存在することを前提とします。このワークスペースは、同期されている必要はありません。*

```
#!/bin/sh
# Set target string, files to search, location of p4 executable...
TARGET="Copyright `date +%Y` Example Company"
DEPOT_PATH="//depot/src/..."
CHANGE=$1
P4CMD="/usr/local/bin/p4 -p 1666 -c copychecker"
XIT=0
echo ""
# For each file, strip off #version and other non-filename info
# Use sed to swap spaces w/"%" to obtain single arguments for "for"
for FILE in `$P4CMD files $DEPOT_PATH@=$CHANGE | \
  sed -e 's/\(.*\)\#[0-9]* - .*$/\1/' -e 's/ /%/g'`
do
  # Undo the replacement to obtain filename...
  FILE="`echo $FILE | sed -e 's/%/ /g'`"
# ...and use @= specifier to access file contents:
  # p4 print -q //depot/src/file.c@=12345
  if $P4CMD print -q "$FILE@=$CHANGE" | grep "$TARGET" > /dev/null
  then
  else
      echo "Submit fails: '$TARGET' not found in $FILE"
      XIT=1
  fi
done
exit $XIT
```

このトリガを使用するには、トリガ・テーブルに以下の行を追加します。

```
sample2 change-content //depot/src/... "copydate.sh %change%"
```

このトリガは、`//depot/src` *内のファイルを少なくとも 1 つ持つチェンジリストがサブミットされたときに起動します。スクリプトに定義された対応する* `DEPOT_PATH` *によって、トリガ対象のチェンジリスト内のすべてのファイルのうちで、実際に* `//depot/src` *の下にあるファイルだけがチェックされるようになります。*

チェンジ – コミット・トリガ

change-commit トリガ・タイプを使用して、チェンジリスト作成、ファイル転送、およびデータベースへのチェンジリストのコミットの後に起動されるトリガを作成します。チェンジリストのサブミットの成功を前提とする (または必要とする) プロセスの場合は、チェンジ – コミット・トリガを使用します。

例 : *以下のチェンジ – コミット・トリガは、サブミットしたチェンジリスト内のファイルを作業状態にしている他のユーザに電子メールを送信します。*

```
#!/bin/sh
# mailopens.sh - Notify users when open files are updated
changelist=$1
workspace=$2
user=$3
p4 fstat @$changelist,@$changelist | while read line
do
  # Parse out the name/value pair.
  name=`echo $line | sed 's/[\. ]\+\([^ ]\+\) .\+/\1/'`
  value=`echo $line | sed 's/[\. ]\+[^ ]\+ \(.\+\)/\1/'`
  if [ "$name" = "depotFile" ]
  then
    # Line is "... depotFile <depotFile>". Parse to get depotFile.
    depotfile=$value
  elif [ "`echo $name | cut -b-9`" = "otherOpen" -a \
    "$name" != "otherOpen" ]
  then
    # Line is "... ... otherOpen[0-9]+ <otherUser@otherWorkspace>".
    # Parse to get otherUser and otherWorkspace.
    otheruser=`echo $value | sed 's/\(.\+\)@.\+/\1/'`
    otherworkspace=`echo $value | sed 's/.\+@\(.\+\)/\1/'`
    # Get email address of the other user from p4 user -o.
    othermail=`p4 user -o $otheruser | grep Email: \
      | grep -v \# | cut -b8-`
    # Mail other user that a file they have open has been updated
    mail -s "$depotfile was just submitted" $othermail <<EOM
The PERFORCE file: $depotfile
was just submitted in changelist $changelist by PERFORCE user $user
from the $workspace workspace.  You have been sent this message
because you have this file open in the $otherworkspace workspace.
EOM
  fi
done
exit 0
```

このトリガを使用するには、トリガ・テーブルに以下の行を追加します。

```
sample3  commit //... "mailopens.sh %changelist% %client% %user%"
```

ユーザがチェンジリストをサブミットするたびに、そのチェンジリストの影響を受ける作業状態のファイルを持つすべてのユーザが電子メール通知を受信します。

修正に対するトリガ

ユーザがチェンジリストに修正を追加または削除したときにトリガ・スクリプトを実行するように PERFORCE を設定するには、修正トリガを使用します。修正トリガは、タイプ fix-add および fix-delete のトリガです。

フィックス – アド・トリガおよびフィックス – デリート・トリガ

例 : *次のスクリプトは、*fixadd.sh *および* fixdel.sh *にコピーされると、ユーザが* p4 fix *コマンドを使用するか* p4 change *コマンドおよび* p4 submit *コマンドによって表*

示されるフォームの [Jobs:] *フィールドを修正することにより、修正レコードを追加または削除したときに実行されます。.*

```
#!/bin/bash
# fixadd.sh, fixdel.sh - illustrate fix-add and fix-delete triggers

COMMAND=$0
CHANGE=$1
NUMJOBS=(($# - 1 ))

echo $COMMAND: fired against $CHANGE with $NUMJOBS job arguments.
echo $COMMAND: Arguments were: $*
```

例: fix-add *トリガおよび*fix-delete *トリガは、ユーザがチェンジリストに修正レコードを追加（または削除）しようとするたびに実行されます。これらのトリガを使用するには、トリガ・テーブルに以下の行を追加します。*

```
sample4   fix-add    fix "fixadd.sh %change% %jobs%"
sample5   fix-delete fix "fixdel.sh %change% %jobs%"
```

*両方のスクリプトのコピーを使用して、*fixadd.sh *が* p4 fix *によって起動され、*fixdel.sh *スクリプトが*p4 fix -d *によって起動されること、またどちらかのスクリプトがチェンジリスト・フォームの* [Jobs:] *フィールド内に手動でジョブ番号を追加（または削除）することにより、*p4 change *の処理または* p4 submit *処理の一部によって起動される可能性があることを確認してください。*

%jobs% *変数は、*p4 fix *コマンドライン（または* p4 change *あるいは* p4 submit *のフォームの* [Jobs:] *フィールド）にリストされたジョブごとに 1 つずつ引数を拡張するため、fix-add トリガまたは* fix-delete *トリガのスクリプトに渡される最後の引数でなければなりません。*

フォームに対するトリガ

ユーザがフォームを編集したときにトリガ・スクリプトを実行するように PERFORCE を設定するには、フォームトリガを使用します。フォームトリガは、タイプ `form-save`、`form-in`、`form-out`、`form-delete`、`form-commit` のトリガです。

フォームトリガを使用して、ユーザ用にカスタマイズされたフォームを生成したり、フォームを検証したり、フォームに対する変更を他のユーザに通知したり、あるいはプロセス制御ツールおよびプロセス管理ツールと通信します。

フォーム - セーブ・トリガ

`form-save` トリガタイプを使用して、ユーザが変更したフォームをサーバに送信したときに起動するトリガを作成します。また、フォームがサーバによって解析された後、変更したフォームが PERFORCE メタデータに保存される前に呼びされます。

例 : *特定のユーザにクライアント・ワークスペースの変更を禁止するには、そのユーザを* `lockedws` *という名前のグループに追加し、以下の* `form-save` *トリガを使用します。*

このトリガは、`lockedws` *グループに属するユーザのクライアント・ワークスペース仕様の変更を拒否し、ユーザ名、ユーザのワークステーションの IP アドレス、および変更が行われたワークスペースの名前が記述されたエラー・メッセージを出力して管理者に通知します。*

```
#!/bin/bash
NOAUTH=lockedws
USERNAME=$1
WSNAME=$2
IPADDR=$3
GROUPS=`p4 groups "$1"`
if echo "$GROUPS" | grep -qs $NOAUTH
then
   echo "$USERNAME ($IPADDR) in $NOAUTH may not change $WSNAME"
   mail -s "User $1 workspace mod denial" admin@127.0.0.1
   exit 1
else
   exit 0
fi
```

`form-save` *トリガは、*`client` *フォームに対してのみ動作します。トリガを使用するには、トリガ・テーブルに以下の行を追加します。*

```
sample6 form-save client "ws_lock.sh %user% %client% %clientip%"
```

`p4 groups lockedws` *の出力に名前が表示されたユーザのクライアント・ワークスペースに対する変更がサーバによって解析され、それらの変更が構文的に正しい場合でも、ワークスペースに対する変更は拒否され、管理者に変更されようとしたことが通知されます。*

フォーム - アウト・トリガ

`form-out` トリガタイプを使用して、PERFORCE サーバがユーザに表示するためのフォームを生成するたびに起動するトリガを作成します。

> **警告** `form-out` *トリガでは、同じ* `form-out` *トリガを起動する* PERFORCE *コマンドを使用しないでください。そのようなコマンドを使用すると、無限に反復することになります。例えば、*`job` *フォームに対して動作する* `form-out` *トリガ・スクリプト内から* `p4 job -o` *を実行しないでください。*

例 : *デフォルトの PERFORCE クライアント・ワークスペース・ビューでは、ディポ* `//depot/...` *全体がユーザのクライアント・ワークスペースにマッピングされます。慣れないユーザがディポ全体を同期するのを防ぐために、この* `Perl` *スクリプトで、* `p4 client` *フォームのデフォルト・ワークスペース・ビュー* `//depot/...` *を変更して、* `//depot/releases/main/...` *の現在のリリースのコードラインだけをマッピングします。*

```
#!/usr/bin/perl
# default_ws.pl - Customize the default client workspace view.
$p4 = "p4 -p localhost:1666";
$formname = $ARGV[0];  # from %formname% in trigger table
$formfile = $ARGV[1];  # from %formfile% in trigger table
# Default server-generated workspace view and modified view
# (Note: this script assumes that //depot is the only depot defined)
$defaultin = \t//depot/... //$formname/...\n";
$defaultout = "\t//depot/releases/main/... //$formname/...\n";
# Check "p4 clients": if workspace exists, exit w/o changing view.
open CLIENTS, "$p4 clients |" or die "Couldn't get workspace list";
while ( <CLIENTS> )
{
        if ( /^Client $formname .*/ ) { exit 0; }
}
# Build a modified workspace spec based on contents of %formfile%
$modifiedform = "";
open FORM, $formfile or die "Trigger couldn't read form tempfile";
while ( <FORM> )
{       ## Do the substitution as appropriate.
        if ( m:$defaultin: ) { $_ = "$defaultout"; }
        $modifiedform .= $_;
}
# Write the modified spec back to the %formfile%,
open MODFORM, ">$formfile" or die "Couldn't write form tempfile";
print MODFORM $modifiedform;
exit 0;
```

`form-out` *トリガは、* `client` *ワークスペース・フォームに対してのみ動作します。トリガを使用するには、トリガ・テーブルに以下の行を追加します。*

```
sample7 form-out client "default_ws.pl %formname% %formfile%"
```

クライアント・ワークスペースを作成する新しいユーザには、カスタマイズされたデフォルトのビューが提供されます。

フォーム - イン・トリガ

`form-in` トリガタイプを使用して、ユーザがフォームをサーバに送信しようとしたとき、フォームが PERFORCE サーバによって解析される前に起動するトリガを作成します。

例 : *ジョブの編集が許可されているすべてのユーザが* `jobbers` *という名前の指定グループに配置されています。次の Python スクリプトは、*`-G` *(Python 配置オブジェクト) オプションを指定した* `p4 group -o jobbers` *を実行して、スクリプトをトリガから実行したユーザが jobbers グループに属するかどうかを判断します。*

```
import sys, os, marshal
# Configure for your environment
tuser = "triggerman"    # trigger username
auth_group = "jobbers"  # PERFORCE group authorized to edit jobs
# Get trigger input args
user = sys.argv[1]
# Get authorized user list
# Use global -G flag to get output as marshaled Python dictionary
CMD = "p4 -G -u %s -p 1666 group -o %s" % \
        (tuser, auth_group)
result = {}
result = marshal.load(os.popen(CMD, 'r'))
auth_users = []
for k in result.keys():
        if k[:4] == 'User': # user key format: User0, User1, ...
                u = result[k]
                auth_users.append(u)
# Compare current user to authorized users.
if not user in auth_users:
        print "\n\t>>> You don't have permission to edit jobs."
        print "\n\t>>> You must be a member of '%s'.\n" % auth_group
        sys.exit(1)
else: # authorized user -- OK to create/edit jobs
        sys.exit(0)
```

`form-in` *トリガは、*`job` *フォームに対してのみ動作します。トリガを使用するには、トリガ・テーブルに以下の行を追加します。*

```
sample8 form-in job "python jobgroup.py %user%"
```

ユーザが `jobbers` *グループに属する場合、*`form-in` *トリガは成功し、ジョブは解析のために PERFORCE サーバに渡されます。それ以外の場合は、エラー・メッセージが表示され、ジョブに対する変更は拒否されます。*

フォーム - デリート・トリガ

`form-delete` トリガタイプを使用して、ユーザがフォームを削除しようとしたとき、フォームが PERFORCE サーバによって解析された後、そのフォームが PERFORCE データベースから削除される前に起動するトリガを作成します。

例 : *ジョブをシステムから削除せず、対象のジョブの状態を* `closed` *にするという規則を、管理者がユーザに徹底させたいとします。*

```
#!/bin/sh
echo "jobs may not be deleted. Please mark jobs as closed instead."
exit l
```

`form-delete` *トリガは、*`job` *フォームに対してのみ動作します。トリガを使用するには、トリガ・テーブルに以下の行を追加します。*

```
sample9 form-delete job "nodeljob.sh"
```

ユーザがジョブを削除しようとすると、ジョブ削除要求は拒否され、エラーメッセージが出力されます。

フォーム – コミット・トリガ

他のフォーム・トリガとは異なり、`form-commit` トリガはフォームがデータベースにコミットされた後に実行されます。このトリガはフォームのサブミットが正常に完了することが想定される（または要求される）処理に使用してください。ジョブ・フォームでは、ジョブがデータベースにコミットされるまでジョブ名は設定されません。`form-commit` トリガは、ジョブ作成処理の一部で新しいジョブの名前を取得できる唯一の方法です。

例 : `newjob.sh` *にコピーしたとき、次のスクリプトはジョブ作成処理中にジョブ名を取得する方法を示し、ジョブ修正に関連するチェンジリストのステータスを報告します。*

```
#!/bin/sh
# newjob.sh - illustrate form-commit trigger
COMMAND=$0
USER=$1
FORM=$2
ACTION=$3
echo $COMMAND: User $USER, formname $FORM, action $ACTION >> log.txt
```

トリガを使用するには、トリガ・テーブルに以下の行を追加します。

```
sample10 form-commit job "newjob.sh %user% %formname% %action%"
```

`%action%` *変数を使用すると、ジョブへの変更がジョブを直接操作するユーザによって* `p4 job` *を使用して要求されたのか、または* `p4 fix` *のコンテキスト内またはチェンジリストの* `[Jobs:]` *フィールドでジョブを修正することによって間接的に要求されたのかを識別することができます。*

最も簡単なケースは `p4 job` *コマンドによる新しいジョブの作成（または既存ジョブへの変更）です。この処理ではトリガが実行され、スクリプトはユーザに新規に作成された（または編集された）ジョブの名前を報告します。これらのケースでは、* `%action%` *変数はヌルです。*

またこのトリガは、変更されたジョブが `p4 fix`、`p4 fix -d` *の実行により、あるいは* `p4 change` *フォームまたは* `p4 submit` *フォームで示されるチェンジリストの* `[Jobs:]` *フィールドの編集により操作されているかどうかにかかわらず、ユーザがジョブをチェンジリストに追加または削除するときにも実行されます。その場合には、* `%action%` *変数はジョブが追加または削除されるチェンジリストのステータス (作業中またはサブミット済）を保持します。*

`%action%` *変数は常に設定されるとは限らないため、* `form-commit` *トリガ・スクリプトに渡される最後の引数でなければなりません。*

外部認証のためのトリガの利用

外部認証管理（LDAP や Active Directory など）が機能するよう PERFORCE を構成するには、*認証トリガ*（`auth-check` および `auth-set`）を使用します。これらのトリガはそれぞれ、`p4 login` コマンドと `p4 passwd` コマンドで起動します。

認証トリガがチェンジリストやフォーム・トリガと異なる点は、認証プロセスの一部としてユーザに入力されたパスワードは、コマンドライン上ではなく標準入力として認証スクリプト

に渡されることです。(コマンドラインに渡される引数は、%user%、%clientip% などすべてのトリガタイプに共通する引数のみです。)

トリガをトリガ・テーブルに追加する際はトリガ名のつづりを間違えないよう注意してください。つづりを間違えるとすべてのユーザが PERFORCE にログインできなくなることがあります。

トリガおよびトリガテーブルの起動については本番環境に実装する前に十分に試験を行ってください。

サーバへのアクセスの回復に補助が必要な場合は PERFORCE 技術サポートに連絡してください。

本書にある例は説明のための一例にすぎません。LDAP 環境用のサンプル・コードへのリンクを含む詳細な解説は、以下の TechNote を参照してください。

```
http://www.perforce.com/perforce/technotes/note074.html
```

auth-check トリガを追加した後、このトリガを有効にするために PERFORCE サーバを再起動しなければなりません。ただし、サーバを再起動せずに既存の auth-check トリガ・テーブルのエントリ(またはトリガ・スクリプト)を変更することもできます。

auth-check トリガ を配置してサーバを再起動すると、PERFORCE の security カウンタは無視されます。それは、認証がトリガ・スクリプトにより制御されるようになったため、パスワード強度の要求に対するサーバのデフォルトのメカニズムは冗長であるからです。

認証チェック・トリガ

auth-check タイプトリガはユーザが p4 login コマンドを実行したとき起動します。スクリプトが 0 を返すと、ログインは成功し、そのユーザのチケット・ファイルが作成されます。

例 : *簡単な認証チェック・スクリプトを示します。すべてのユーザはログイン・チケットを受け取るために "secret" というパスワードを入力しなければなりません。ユーザが指定するパスワードは標準入力からスクリプトに送られます。*

```
#!/bin/bash
# checkpass.sh - a trivial authentication-checking script
# in this trivial example, all users have the same "secret" password
USERNAME=$1
PASSWORD=secret
# read user-supplied password from stdin
read USERPASS
# compare user-supplied password with correct password
if [ "$USERPASS" = $PASSWORD ]
then
    # Success
    exit 0
fi
# Failure
echo checkpass.sh: password $USERPASS for $USERNAME is incorrect
exit 1
```

ユーザが p4 login を起動するたびに auth-check トリガが起動します。トリガを使用するには、トリガ・テーブルに以下の行を追加します。

```
sample11 auth-check auth "checkpass.sh %user%"
```

"secret" というパスワードを入力したユーザは、ログイン・チケットを与えられます。

認証セット・トリガ

auth-set タイプのトリガは、ユーザが `p4 passwd` コマンドを実行し、古いパスワードに対する `auth-check` トリガによる認証が成功したときに起動します。

1. ユーザが `p4 passwd` コマンドを起動します。
2. PERFORCE サーバがユーザにパスワードの入力を要求します。
3. PERFORCE サーバが `auth-check` トリガを起動し、外部の認証サービスに対する古いパスワードを検証します。
4. `auth-check` トリガに関連付けられたスクリプトが起動します。`auth-check` トリガが失敗すると、プロセスは直ちに終了します。ユーザは新しいパスワードを要求されず、`auth-set` トリガは起動しません。
5. `auth-check` トリガが成功すると、PERFORCE サーバはユーザに新しいパスワードの入力を要求します。
6. PERFORCE サーバは `auth-set` トリガを起動し、標準入力から古いパスワードと新しいパスワードの両方を改行で区切ってトリガ・スクリプトに渡します。

> **注** *I ほとんどの場合、外部の認証環境のユーザはそのまま PERFORCE を使用せずにパスワードを設定します。`auth-set` トリガタイプは万全を期するために用意されています。*

PERFORCE サーバはユーザの現在のパスワードを検証する必要があるため、auth-set トリガを作成する前に、正しく機能する auth-check トリガが用意されていなければなりません。

例 : *簡単な認証設定スクリプト*

```
#!/bin/bash
# setpass.sh - a trivial authentication-setting script
USERNAME=$1
read OLDPASS
read NEWPASS
echo setpass.sh: $USERNAME attempted to change $OLDPASS to $NEWPASS
```

`auth-set` トリガはユーザが `p4 passwd` を実行し `auth-check` トリガに要求される外部認証に合格した後に起動します。トリガを使用するには、トリガ・テーブルに以下の 2 行を追加します。

```
sample11 auth-check auth "checkpass.sh %user%"
sample12 auth-set auth "setpass.sh %user%"
```

この例では実際にはパスワードの変更は行われません。ユーザが試行した内容を返すだけです。

複数のトリガの利用

チェンジリストおよびフォーム・トリガはトリガ・テーブル上の順番に従って実行されます。1 つのファイル・パターンにつき同じパスに対して実行される同じタイプの複数のトリガを生成した場合、各トリガがトリガ・テーブル上の順番に従って実行され、いずれかのトリガが失敗すれば、それ以下のトリガは実行されません。

例 : *同じファイルに対する複数トリガの実行*

すべての `*.c` *ファイルがスクリプト* `check1.sh`、`check2.sh`、`check3.sh` *をパスしなければなりません。*

```
Triggers:
   check1  submit   //depot/src/*.c  "/usr/bin/check1.sh %change%"
   check2  submit   //depot/src/*.c  "/usr/bin/check2.sh %change%"
   check3  submit   //depot/src/*.c  "/usr/bin/check3.sh %change%"
```

いずれかのトリガ（例えば `check1.sh`*）が失敗すれば、サブミットも失敗し、以下のトリガ（この場合、*`check2.sh` *と* `check3.sh`*）は実行されません。1 つのトリガが成功すれば、次の該当するトリガが実行されます。*

複数のファイル指定を同一のトリガ（およびトリガ・タイプ）にリンクするには、トリガ・テーブルの中にそのトリガを複数回記述します。

例 : *複数のファイル指定に対する同じトリガの起動*

```
Triggers:
   bugcheck  submit   //depot/*.c   "/usr/bin/checkit.pl %change%"
   bugcheck  submit   //depot/*.h   "/usr/bin/checkit.pl %change%"
   bugcheck  submit  //depot/*.cpp  "/usr/bin/checkit.pl %change%"
```

この場合トリガ `bugcheck` *は、* `*.c` *ファイル、* `*.h` *ファイルおよび* `*.cpp` *ファイルに対して適用されます。*

同じパスに対して実行されるタイプの異なる複数のチェンジリスト・サブミット・トリガは、以下の順番で実行されます。

1. `change-submit`（ファイル転送の前に、チェンジリスト・サブミットに対して実行される）
2. `change-content` トリガ（チェンジリスト・サブミットおよびファイル転送の後）
3. `change-commit` トリガ（サーバによる自動チェンジリスト・ナンバリングに対して実行される）

同様に、タイプの異なる仕様トリガは以下の順番で起動されます。

1. `form-out`（フォーム生成）
2. `form-in`（変更されたフォームがサーバに転送される）
3. `form-save`（ 検証されたフォームが PERFORCE データベースへの保存のための準備ができる）
4. `form-delete`（検証されたフォームは既に PERFORCE データベースに保存されている）

複数の PERFORCE サーバをサポートするトリガの作成

複数の PERFORCE サーバから同じトリガ・スクリプトを呼び出すには、変数 %serverhost%、%serverip%、および %serverport% を使用してトリガ・スクリプトの移植性を高めます。

例えば、ハードコーディングされたポート番号とアドレスを使用するスクリプトがある場合は、以下のスクリプトを作成することになります。

```
#!/bin/sh
# Usage: jobcheck.sh changelist
CHANGE=$1
P4CMD="/usr/local/bin/p4 -p 192.168.0.12:1666"
$P4CMD describe -s $1 | grep "Jobs fixed...\n\n\t" > /dev/null
```

トリガ・テーブル内の以下の行で上記のスクリプトを呼び出します。

```
sample1  submit //depot/qa/... "jobcheck.sh %change%"
```

移植性を高めるには、スクリプトを以下のように変更します。

```
#!/bin/sh
# Usage: jobcheck.sh changelist server:port
CHANGE=$1
P4PORT=$2
P4CMD="/usr/local/bin/p4 -p $P4PORT"
$P4CMD describe -s $1 | grep "Jobs fixed...\n\n\t" > /dev/null
```

次に、サーバ固有のデータを上記のトリガ・スクリプトに引数として渡します。

```
sample2  submit //depot/qa/... "jobcheck.sh %change% %serverport%"
```

各トリガ・タイプに適用される変数の完全な一覧については、91 ページの「トリガ・スクリプトの変数」を参照してください。

トリガとセキュリティ

警告！ p4d *プロセスはトリガを生成するため、*UNIX *システム上では* p4d *を* root *で実行しないよう強くお勧めします。*

トリガと Windows

デフォルトでは、PERFORCE のサービスは Windows のローカル・システム・アカウントのもとで実行されます。

Windows では、ネットワーク・ドライブ上のファイルにアクセスするためには実のアカウント名とパスワードが必要です。したがって、トリガ・スクリプトがネットワーク・ドライブ上にある場合には、実のユーザ ID とパスワードを用いてそのスクリプトにアクセスできるように、サービスを構成しなければなりません。

詳しくは、125 ページの「ネットワーク・ドライブへの PERFORCE サービスのインストール」をご覧ください。

デーモン

*デーモン*は、バックグラウンドで定期的に呼び出されるプロセス、または連続的に実行されるプロセスです。PERFORCE を使用するデーモンは通常、必要に応じてサーバのメタデータを調べ、必要に応じてアクションを起こします。

代表的なデーモンのアプリケーションには、次のようなものがあります。

- 10 分ごとに起動して、プロダクション・ディポにチェンジリストがサブミットされていないかどうかをチェックするチェンジ・レビュー・デーモン。チェンジリストがサブミットされ

ていたら、そのチェンジリストに含まれるファイルに「サブスクライブ」しているユーザに電子メールを送り、ユーザが関わっているファイルが変更されたことを知らせます。

- 一日の終わりに、作業状態のジョブに関するレポートを毎日生成するジョブ・デーモン。カテゴリ別のジョブの数、各ジョブの重要度などを示します。この報告はメールですべての関係ユーザに送られます。
- 特定のディポ・サブディレクトリにあるファイルが変更されたかどうかを調べる Web デーモン。新しいファイル・リビジョンが見つかったら、それを運用中の Web ページを含むクライアント・ワークスペースに同期させます。

PERFORCE のメタデータが変更されたときは、様々なタスクの起動が必要になります。デーモンを使用すれば、これらのタスクのほとんどを起動することができます。主としてサブミットの妥当性評価に用いられるトリガと異なり、デーモンはディポへの情報書き込み（すなわち、ファイルのサブミット）にも使えます。

PERFORCE のチェンジ・レビュー・デーモン

PERFORCE チェンジリスト・レビュー・デーモン (`p4review.py`) は、以下の Web サイトの PERFORCE Supporting Programs ページから入手できます。

```
http://www.perforce.com/perforce/loadsupp.html#daemon
```

チェンジ・レビュー・デーモンは `http://www.python.org/` から入手できる Python 上で動作します。レビュー・デーモンを実行する前に、デーモン本体に含まれる構成説明書を読み、指示に従ってください。

ユーザは、`p4 user` を呼び出して、`Email:` フィールドに電子メール・アドレスを入力し、関心のあるファイルに対応するファイル・パターンの番号を `Reviews:` フィールドに入力して、ファイルにサブスクライブします。

```
User:     sarahm
Email:    sarahm@elmco.com
Update:   1997/04/29 11:52:08
Access:   1997/04/29 11:52:08
FullName: Sarah MacLonnogan
Reviews:
        //depot/doc/...
        //depot.../README
```

チェンジ・レビュー・デーモンは、個々の新しいサブミット済チェンジリストに含まれていたファイルを監視し、そのファイルのいずれかに「サブスクライブ」しているすべてのユーザに電子メールを送り、問題のファイルが変更されたことを知らせます。

特別なパス `//depot/jobs` を `Reviews:` フィールドに含めることにより、ジョブのデータが更新されたときにも、ユーザは PERFORCE チェンジ・レビュー・デーモンからメールを受け取ることができます。

チェンジ・レビュー・デーモンは、次のスキームを実行します。

1. `p4 counter` を使用して、PERFORCE メタデータ中の*カウンタ*と呼ばれる変数を読み取り、変更します。このデーモンが使用するカウンタ `review` は、レビューされた最新のチェンジリストの番号を保存します。
2. `p4 review -t review` で PERFORCE のディポをポーリングし、まだレビューされていないサブミット済チェンジリストを探します。
3. `p4 reviews` を使用して、見つかったチェンジリストのそれぞれについてレビュー者のリストを生成します。
4. Python メール・モジュールを使用して、`p4 describe` で出力されるチェンジリストの情報を電子メールでレビューアに送ります。

5. 前記 1 ～ 3 のステップは 3 分ごとに繰り返されます。この時間間隔は、デーモンをインストールするときに変更できます。

ステップ 4 で使用されるコマンド（`p4 describe`）は、直接レポートするコマンドです。その他のコマンド（`p4 review`、`p4 reviews`、`p4 counter`）は、ほとんどの場合レビュー・デーモンによって実行されます。

その他のデーモンの生成

`p4review.py`（104 ページの「PERFORCE のチェンジ・レビュー・デーモン」を参照）をもとに、必要に応じて変更を行うことで、独自のデーモンを生成することができます。1 例として、チェンジリストのサブミット後に、PERFORCE のジョブ情報を外部のバグ追跡システムにアップロードするデーモンなどが考えられます。このデーモンはまず、新しいレビュー・カウンタ `p4 review` を使用して、新しいチェンジリストをリストし、`p4 fixes` を使用して、新たにサブミットされたチェンジリストによって修正完了となったジョブのリストを取得します。最後に、その情報を外部のシステムに送り、一部のジョブが完了したことを知らせます。

独自のデーモンを作成し、それを他のユーザと共用したい場合には、それを PERFORCE Public Depot へサブミットしてください。詳しくは、`http://www.perforce.com` にアクセスし、「PERFORCE Public Depot」リンクをご覧ください。

デーモンで使用されるコマンド

PERFORCE コマンドには、ほとんどレビュー・デーモンによってのみ実行されるコマンドがあります。これらのコマンドは以下のとおりです。

	使用法
`p4 review -c change#`	*change#* から最新のサブミット済チェンジリストまでの全チェンジリストについて、チェンジリスト番号、生成者、および生成者の電子メール・アドレスをリストします。 実行するには、少なくとも `review` アクセス権限が必要です。
`p4 reviews -c change# filespec`	指定されたチェンジリスト内の指定ファイルまたは任意のファイルのレビューにサブスクライブされているユーザ全員をリストします。
`p4 counter name [value]`	新しいカウンタを作成するかまたは既存のカウンタに値を設定するには、少なくとも `review` 権限が必要です。カウンタの値を表示するには、少なくとも `list` 権限が必要です。 引数 *value* が指定されていない場合、`p4 counter` は *name* の値を返します。カウンタが存在しない場合は `0` を返します。 引数 *value* が指定されている場合、`p4 counter` は *name* の値に *value* をセットします。*name* というカウンタが存在しない場合は作成されます。 ***警告:** レビュー・カウンタ* `journal`、`job`、および `change` は、PERFORCE の内部で使用されます。***これらのいずれかをレビュー・ナンバーとして使用すると、PERFORCE のデータベースが破損することがあります!*** カウンタの値は、内部的に符号付整数（signed int）で表現されています。ほとんどのプラットフォームにおいて、最大値は 2^{31} - 1 または 2147483647 です。 64 ビットのプラットフォームでは、2^{63} - 1 または 9223372036854775807 となります。
`p4 counters`	すべてのカウンタならびにその値をリストします。
`p4 changes -m 1 -s submitted`	PERFORCE サーバが認識している最大のチェンジリスト番号ではなく、最新のサブミット済チェンジリストの番号を 1 行に表示します。

デーモンとカウンタ

チェンジ・レビュー・デーモン、またはサブミット済チェンジリストを扱う他のデーモンを作成する場合、通常は最新の*サブミット済*チェンジリストの番号を追跡する必要があります。この番号は、`p4 changes -m 1 -s submitted` コマンドのアウトプットの第 2 フィールドとして取得できます。

これは `p4 counter change` のアウトプットと同じでは*ありません*。PERFORCE サーバが認識している最新のチェンジリスト番号（`p4 counter change` のアウトプット）は、ユーザによって生成されたが、まだディポにサブミットされていない作業中のチェンジリストも含んでいます。

スクリプト作成とバッファリング

ご使用のプラットフォームによって、個々の p4 コマンドのアウトプットが完全にバッファリングされている（所定のバイト数が生成されたところで初めてアウトプットがフラッシュされる）場合、行単位でバッファリングされている（tty のように、改行ごとに 1 行が送られる）場合、バッファリングが行われていない場合があります。

一般に、ファイルまたはパイプへの標準出力は十分にバッファリングされていて、tty への標準出力は行単位でバッファリングされています。トリガまたはデーモンが行単位のバッファリング（またはバッファリングなし）を必要としていれば、当該の p4 コマンドに -v0 デバッグ・オプションを付けることで、バッファリングを無効にすることができます。

パイプを用いて PERFORCE コマンドからの標準出力を転送しようとしている場合には（-v0 オプションの有無にかかわらず）、カーネルによってもたらされるバッファリングの問題に遭遇することがあります。-v0 オプションは、コマンドそのものの出力のバッファリングを無効にするのみです。

第 7 章

PERFORCE のパフォーマンス調整

PERFORCE サーバは、本来システム・リソースをそれほど消費しません。しかし、システムが拡大してくると、あらためてシステム構成を調べ、最適なパフォーマンスが得られる構成になっているかどうかを確認する必要があります。

本章では、PERFORCE サーバのパフォーマンスに影響しそうないくつかの要因について簡単に概略を説明するとともに、ネットワークに関する問題の診断に役立つ情報を紹介し、システムの大型化によるサーバの負荷を軽減する方法を説明します。

パフォーマンスを調整する

PERFORCE サーバのパフォーマンスに影響する因子は次のとおりです。

メモリ

サーバのパフォーマンスは十分なメモリが確保されているかどうかに大きく依存します。これに関して 2 つの問題点があります。この問題点を回避する 1 つの方法は、大量のクエリを実行中のサーバにページングさせないことです。もう 1 つの方法は、`db.rev` テーブル（または少なくともその実際的な大きさ）がメイン・メモリにキャッシュされるようにすることです。

- 大量のクエリに必要なメモリ量は簡単に割り出せます。サーバはページングを避けるためにファイル当たり約1KBのRAMを必要とします。10,000ファイルなら10MBのRAMが必要です。
- `db.rev` をキャッシュするには、既存のインストレーションの `db.rev` ファイルのサイズを確認し、1 つの目安として利用すればよいでしょう。新しいインストレーションの PERFORCE では、`db.rev` はリビジョン当たり約 150 ～ 200 バイト必要で、ファイル当たり約 3 リビジョンとすると、ファイル当たり約 0.5KB の RAM が必要ということになります。

したがって、ファイル当たり 1.5KB、100,000 ファイルに対して 150MB の RAM が利用可能であれば、たとえ全ファイルが関係する動作を実行しても、サーバはページングしないことになります。もちろん、複数の大規模な動作を同時に実行することもあり得るので、ページングを避けるためには、さらにメモリが必要です。ただし、大多数の操作は小さなサブセットのファイルくらいしか使用しません。

通常のインストレーションの場合、ディポにファイル当たり 1.5KB 程度の RAM があれば十分でしょう。

ファイルシステムのパフォーマンス

PERFORCE はディスク I/O の利用に関して優れています。メタデータが検索しやすく、アクセスは、通常限られたサブセットのデータのシーケンシャル・スキャンです。

唯一のディスク集約的な動作はファイルのチェックインで、PERFORCE サーバがアーカイブのファイルの書き込みとリネームをしなければならない場合です。サーバのパフォーマンスはオペレーティング・システムのファイルシステムの実装、特に、ディレクトリ更新が同期式かどうかに強く依存しています。

PERFORCE は特定のファイルシステムを推奨しませんが、Linux サーバは一般に動作が最も速い（Linux の非同期式ディレクトリ更新による）反面、不適切なタイミングで電源が切れると、復旧に手間取ることがあります。BSD ファイルシステム（Solaris でも使用されている）は比較的動作が遅い反面、信頼性は高くなります。NTFS のパフォーマンスは前の 2 つの中間に落ち着くでしょう。IRIX や OSF で使用されているファイルシステムは、速度と丈夫さの両方をうまく兼ね備えています。

データベースとバージョン管理されたファイルが、NFS マウントされているボリュームに保存されているシステムのパフォーマンスは、通常その NFS の実装やその土台となるストレージ・ハードウェアに依存します。PERFORCE は Solaris の NFS 上でテストされ、サポートされています。

Linux および FreeBSD 上では、ファイル・ロッキングが比較的遅いため、NFS を介するデータベース更新に問題が生じる可能性があります。これらのプラットフォーム上では、ジャーナルが NFS マウントされている場合、すべての動作が遅くなるでしょう。一般に（Linux や FreeBSD では特に）PERFORCE のデータベース、ディポ、ジャーナルのファイルは、PERFORCE サーバ・プロセスを実行するマシンのローカル・ディスク上に保存することをお勧めします。

上記の問題は、PERFORCE サーバ・プロセス（p4d）だけに影響します。PERFORCE クライアント・プログラム（PERFORCE コマンドライン・クライアントの p4 など）はいつでも NFS マウントされたドライブ（例えば、ユーザのホーム・ディレクトリ）上のクライアント・ワークスペースで作業することができます。

ディスク容量の割り当て

PERFORCE が使用するディスク容量は次の 3 つの要素によって決まります。

- クライアント・ワークスペースの数とサイズ
- サーバ・データベースのサイズ
- サーバの全バージョン化ファイルのアーカイブのサイズ

3 つとも、扱うデータの性質と PERFORCE の利用度に応じて変わります。

必要なクライアント・ファイル・スペースは、ユーザが任意の時点にクライアント・ワークスペースで必要とするファイルのサイズです。

サーバのデータベースのサイズはかなりの精度で計算できます。推定で、1 ユーザ 1 ファイル当たり 0.5KB が必要でしょう（すなわち、10,000 ファイルのシステムで、ユーザが 50 人いれば、データベースに 250MB のディスク容量が必要になります）。データベースは、時間が経過し、個々のファイルの履歴が増加するにつれて、大きくなると考えられます。

サーバのバージョン化ファイルのアーカイブのサイズは、保存されているオリジナル・ファイルのサイズに依存し、リビジョンが増えるにつれて大きくなります。通常はオリジナル・ファイルのサイズの少なくとも 3 倍のディスク空間を割り当てておきます。

データベースが GB レベルまで大きくなると予想されるなら、ご使用のプラットフォームが大規模なファイルシステムを適切にサポートしていることを確認しておいた方がよいでしょう。21 ページの「成長を見越した十分なディスク容量の割当」をご覧ください。

db.haveファイルはクライアント・ワークスペースで作業状態になっているファイルのリストを保持しているため、データベースの他のファイルより急速に大きくなる傾向があります。db.haveファイルのサイズに関する問題を抱えていて、大きなファイルを適切にサポートしているサーバにすぐに切り換えられない場合は、使っていないクライアント・ワークスペース仕様を削除し、クライアント・ワークスペースのビューの範囲をせまくすれば、問題は多少緩和されるでしょう。

ディスク空き容量の監視

ディポ内（の一部）のファイルに現在占有されているディスク容量を調べるには、p4 sizes コマンドにサイトのディスク記憶方法で使用されるブロック・サイズを指定します。例えば、

```
p4 sizes -a -s -b 512 //depot/...
```

というコマンドは、ブロック・サイズが 512 バイトである場合の //depot/... 内にあるすべてのリビジョン (-a) の合計値 (-s) を示します。

```
//depot/... 34161 files 277439099 bytes 5429111 blocks
```

ネットワーク

PERFORCE は TCP/IP ネットワーク上で動作することができます。ネットワーク上の制限はまだ見つかっていませんが、帯域幅は広ければ広いほどよいでしょう。おそらく、10Mb/s の Ethernet より FDDI の方がよいでしょうが、一部のユーザから、T1 （1.5 Mb/s）を使用していても、PERFORCE をローカルで使用しているときに匹敵する応答時間を得られたという報告もあります。PERFORCE 社の従業員は、ISDN （64 Kb/s）回線で問題なく作業しています。

PERFORCE は各クライアントのサーバとのやりとりに TCP/IP 接続を使用します。サーバのポート・アドレスは P4PORT で定義されますが、クライアントのポート番号は TCP/IP によって決められます。コマンドが完了し、接続が閉じると、ポートは 2 分間、TIME_WAIT という状態に置かれます。ポート番号は 1025 ～ 32767 の範囲ですが、一般に数 100 ～数 1000 の番号だけが同時に使用できます。このため、PERFORCE クライアント・コマンドを、例えばスクリプトなどを用いて何度も立て続けに呼び出すことにより、利用可能なポートをすべてふさぐこともできます。

リリース 99.2 より前には、サーバも接続のクライアント側も TIME_WAIT の状態にとどまり、1 ユーザのマシンで動作しているスクリプトがサーバ側のすべての利用可能なポートとつながっているため、他のユーザがサービスを受けられなくなることがありました。リリース 99.2 以降はクライアント側だけが TIME_WAIT になり、PERFORCE サーバは自由に他のクライアントの処理ができるようになっています。

CPU

PERFORCE はクライアント / サーバ・アーキテクチャを基本にしています。クライアントもサーバも、CPU のリソース消費量は少なめです。例えば、ローエンド（140 MHz）SPARC Ultra サーバ上で 80 ユーザをサポートしている PERFORCE サーバの、1 日あたりの CPU 時間は 7 分程度または利用可能な処理能力の約 0.5% です。ピーク時の利用や処理時間の余裕を考えると、このようなサーバで 800 以上のユーザをサポートできるでしょう。

一般に、CPU の能力は PERFORCE サーバをインストールするプラットフォームを決めるときの主要な検討項目ではありません。

応答時間の遅延を診断する

PERFORCE は通常、ネットワーク・リソースをあまり使用しません。ユーザが極端に大規模な操作を行うと、PERFORCE サーバの応答が遅くなる可能性はありますが、p4 コマンドに対する

応答が一貫して遅いのはたいていネットワークの問題に起因します。応答時間が遅くなる原因となり得るものは、以下のとおりです。

1. ドメイン・ネーム・システム（DNS）の構成の誤り
2. Windows ネットワーキングの構成の誤り
3. ネットワーク化されたファイルシステムで p4 実行ファイルへのアクセス障害

初めに `p4 info` を実行して確認します。これがすぐに応答しなければ、ネットワークに問題があります。ネットワークの問題を解決することはこのマニュアルの対象外ですが、以下に若干のトラブルシューティングのヒントを紹介しておきます。

ホスト名を IP アドレスに変えてみる

クライアント・マシン上で `P4PORT` をホスト名の代わりにサーバの IP アドレスに設定してみます。例えば、

```
P4PORT=host.domain:1666
```

とする代わりに、サイト特定の IP アドレスとポート番号を使って、

```
P4PORT=1.2.3.4:1666
```

と設定します。

通常、ホストの IP アドレスは次のコマンドでわかります。

```
ping hostname
```

`p4 info` が、IP アドレスを使うとすぐに応答するにもかかわらず、ホスト名を使うとなかなか応答しない場合には、問題は DNS に関係している可能性が高いでしょう。

p4 info と P4Win の応答時間を比べてみる

P4win を使用している場合は、P4win の［**接続情報を表示**］（［**ヘルプ**］(Help) → ［**接続情報を表示**］(Show Connection Info)）の応答時間をコマンドライン `p4 info` コマンドの応答時間と比べてみるといいでしょう。

P4Win の方が速く情報を返すが p4 info が遅い場合には、DNS 関係に問題があります。(PERFORCE サーバはコマンドラインまたは P4V から `p4 info` の要求を受け取ると、クライアントとサーバのホスト名を他の構成情報とともに送り返すため、リバース・ネーム・ルックアップを行います。しかし、サーバが P4win の ［**接続情報を表示**］の要求を受け取ったときには、IP アドレスを返すだけです。)

> **注** *このテストはリリース 99.1 以降のサーバにのみ有効です。99.1 より前は、PERFORCE サーバは常にリバース・ネーム・ルックアップを行いました。*

Windows のワイルドカード

ディポ・シンタックスとワイルドカードを組み合わせていてファイルパターンが引用されていない p4 コマンド、例えば、

```
p4 files //depot/*
```

などは Windows 上で応答時間が遅くなることがあります。この遅れは、前後にダブルクォーテーションマークを入れることによって防ぐことができます。

```
p4 files “//depot/*”
```

この問題の原因は、p4 コマンドがワイルドカードを拡張する Windows の機能を利用するところにあります。ダブルクォーテーションマークがついていないと、`//depot` はネットワーク

接続されたコンピュータ・パスとして解釈されてしまい、時間をかけて、depot という名のマシンがむなしく検索されます。

DNS ルックアップとホスト・ファイル

Windows では、%SystemRoot%¥system32¥drivers¥etc¥hosts ファイルを用いて IP アドレスとホスト名の組をハードコード化することができます。このファイルへのエントリを追加することによって、DNS の問題を回避できます。UNIX のこれに対応するファイルは /etc/hosts です。

“p4” 実行ファイルの位置

以上のどの診断ポイントによっても応答時間の遅延を説明できない場合は、p4 の実行可能プログラムそのものが、きわめてパフォーマンスの低いネットワーク・ファイルシステム上に存在している可能性があります。この点を確認するには、次のコマンドを実行します。

```
p4 -V
```

これは、ネットワーク・アクセスを試みずにただバージョン情報をプリントアウトするだけのコマンドです。応答時間が遅いのは、p4 実行ファイルそのものへのネットワーク・アクセスに問題があると思われます。ローカル・ファイルシステム上に p4 をコピーまたはダウンロードすれば、応答時間は改善されるでしょう。

サーバの停滞を防止する

一般に PERFORCE のパフォーマンスは、ディポのサイズではなく、ユーザが 1 回のコマンド実行で操作しようとするファイルの数に左右されます。すなわち、3,000,000 ファイルのディポから 30 ファイルのクライアント・ビューを同期するのも、30 ファイルのディポから 30 ファイルのクライアント・ビューを同期するのもそれほど時間は変わらないはずです。

1 つのコマンドによって影響を受けるファイルの数は、主として次の項目により決定されます。

- p4 コマンドラインの引数（GUI 操作の場合は選択されたフォルダ）

  引数がなければ、ほとんどのコマンドはそのクライアント・ワークスペース・ビューのすべてのファイルに対して実行されるか、少なくともすべてのファイルを参照します。

- クライアント・ビュー、ブランチ・ビュー、ラベル・ビュー、プロテクション

  引数のないコマンドは、そのワークスペース・ビューのすべてのファイルに対して動作するので、ビューを無制限にし、プロテクションも設定していないと、コマンドはディポのすべてのファイルに対して実行されることになります。

サーバは、要求に答えて処理を開始すると、その間データベースをロックします。通常の動作の場合は、サーバが要求のバックログを防げる程度にすばやく「入出力」できるため、この方法で成功します。しかし、異常に大きな要求は数秒ないしは数分かかることもあります。いらいらしたユーザが CTRL キーと C キーを同時に押してリトライすると、問題はよけいに悪化します。サーバはさらにメモリを消費し、応答はますます遅くなります。

きわめて大きなディポを持つサイトでは、ビューの範囲を無制限にしたり、引数を指定しないでコマンドを実行すると、PERFORCE サーバの動作が必要以上にきびしくなります。ユーザと管理者は、次の方法によりサーバにかかる負荷を緩和することができます。

- “絞り込んだ” ビューを使用する
- プロテクションを割り当てる
- データベースへの問い合わせを制限する
- 効率のいいスクリプトを作成する

- 効率的に圧縮を使用する

絞り込んだビューを使用する

次のような“制限のゆるい”ビューは、設定は簡単ですが、きわめて大きなディポではトラブルの原因となることがあります。

```
//depot/...        //workspace/...
```

このビューでは、ディポ全体がクライアント・ワークスペースへマッピングされています。通常、このビューはまだまだ“絞り込む”ことができます。例えば、次のビューはディポの特定の領域に限定されています。

```
//depot/main/srv/devA/...          //workspace/main/srv/devA/...
//depot/main/drv/lport/...         //workspace/main/dvr/lport/...
//depot/rel2.0/srv/devA/bin/...    //workspace/rel2.0/srv/devA/bin/...
//depot/qa/s6test/dvr/...          //workspace/qa/s6test/dvr/...
```

また、クライアント・ビューは、ブランチ・ビューやラベル・ビューに対しても、ユーザに必要な作業を行うのに十分な範囲だけを与えるように設定する必要があります。

クライアント、ブランチ、ラベルの各ビューは、PERFORCE 管理者または個々のユーザにより、それぞれ `p4 client`、`p4 branch`、`p4 label` のコマンドを用いて設定されます。

スクリプトを最適化する方法のうちの 2 つ（117 ページの「ブランチ・ビューの使用」および 118 ページの「一時クライアント・ワークスペースの活用」をご覧ください）は、同様のテクニックを利用します。コマンドに使用できるビューのサイズを制限することにより、実行する必要のあるコマンドの数を減らすとともに、コマンドの実行に必要なリソースも減らします。

プロテクションを割り当てる

プロテクション（69 ページの「PERFORCE の管理 : プロテクション」をご覧ください）は別タイプの PERFORCE ビューです。`p4 protect` コマンドで設定され、ユーザが実行するコマンドによって影響を受けるディポ・ファイルを制限します。

ただし、クライアント、ブランチ、ラベルの各ビューとは異なり、プロテクションによって使用されるビューは、PERFORCE スーパー・ユーザにしか設定できません。（プロテクションはディポ・ファイルに対する読み取り、書き込みの権限も制限しますが、この権限のレベルそのものはサーバのパフォーマンスに影響しません。）PERFORCE スーパー・ユーザは、PERFORCE の中にプロテクションを割り当てることにより、ユーザが“制限のゆるい”クライアント仕様を使用していても、ユーザのビューのサイズを効果的に制限することができます。

プロテクションは、次のようにユーザにもグループにも割り当てることができます。

```
write    user     sam          *    //depot/admin/...
write    group    rocketdev    *    //depot/rocket/main/...
write    group    rocketrel2   *    //depot/rocket/rel2.0/...
```

PERFORCE グループは、スーパー・ユーザが `p4 group` コマンドを使って作成します。PERFORCE グループはプロテクションの割り当てを容易にするだけでなく、次に説明する `maxresults` および `maxscanrows` というエラー防止機構を使えます。

データベースへの問い合わせを制限する

各 PERFORCE グループには、それぞれ *maxresults*、*maxscanrows* および *maxlocktime* という値があります。デフォルトでは、`unlimited`（無制限）ですが、スーパー・ユーザは、`p4 group` コマンドを用いて特定グループにおけるこれらの値を制限することができます。

このグループのユーザは、そのグループの MaxResults の値より多いデータベース行に影響するコマンドを実行することはできません。(ほとんどのコマンドの場合、使用するデータベース行の数は使用するファイルの数とほぼ等しくなっています。)

MaxResults と同様に、MaxScanRows もサーバへの過度の要求に対して、ユーザからの特定のコマンドを制限します。(ほとんどのコマンドの場合、走査される可能性がある行の数は、使用するファイルの数にディポ内のファイルが持つ平均リビジョン数を積算した結果とほぼ等しくなっています。)

最後に、MaxLockTime はある種のコマンドがデータベースを長期間にわたってロックし続けないようにするため使用されます。MaxLockTime はミリ秒単位の数値により、データベースのロックを最大限許容する時間を設定します。

これらの制限を設定するには、p4 group コマンドの表示フォーム内で当該フィールドに数値を入力します。1 人のユーザが複数のグループに登録されている場合、それらのグループの MaxResults（または MaxScanRows または MaxLockTime）*制限値*の*最大値*（デフォルトの unlimited 設定を*除く*）がそのユーザの MaxResults（または MaxScanRows または MaxLockTime）値とみなされます。

例: maxresults、maxscanrows および maxlocktime *設定の効果 :*

管理者が、グループ rocketdev *のメンバーの操作を 20,000 ファイル以下の操作に、さらに走査を 100,000 リビジョン以下の走査に、そしてデータベース・テーブルを 30 秒を超えてロックしないように制限するものとします。*

```
Group:        rocketdev
MaxResults:   20000
MaxScanRows:  100000
MaxLockTime:  100000
Timeout:      43200
Subgroups:
Users:
        bill
        ruth
        sandy
```

制限のない（“ゆるい”）クライアント・ビューに設定されているルースが、

```
p4 sync
```

とコマンド入力したとき、ディポに 20,000 を超えるファイルがあると、この sync *コマンドは拒否されます。*MaxResults *の制限を回避するには、自分のクライアント・ビューを限定するか、ビューの全ファイルが必要なら、次のように小さいセットのファイルに分けて同期させます。*

```
p4 sync //depot/projA/...
p4 sync //depot/projB/...
```

いずれの方法でも、サーバを 1 つの極端に大きなコマンドの処理に専念させることなく、ルースは目的のファイルを自分のワークスペースに同期させることができます。

ルースが次のように全ファイルの全リビジョンを走査するようなコマンドを実行しようとし、

```
p4 filelog //depot/projA/...
```

*このときファイル数は 20,000 より少なくても、リビジョン数が 100,000 より多い（もしかしたら projA には 8000 ファイルを持つディレクトリが存在し、それぞれのファイルは 20 リビジョンずつを保持しているかもしれません）場合には、*MaxResults *の制限は適用されませんが、*MaxScanRows *の制限が適用されます。*

どちらの制限が有効であるかにかかわらず、ルースが実行したどのコマンドも、`MaxLockTime` *である 30000 ミリ秒を超えてデータベースをロックすることは許されません。*

特定のグループについて、処理された結果の行数（または走査されたデータベースの行数）の制限を解除するには、そのグループの`MaxResults:`または`MaxScanRows:`の値を`unlimited`に設定します。

これらの制限はユーザを不便にすることがあるため、特定の操作がサーバの動作を遅くしていない限り、使用しないでください。また、`MaxResults:` の値は、PERFORCE の Windows クライアント P4Win によって実行される特定の操作に 5,000 ～ 8,000 必要になるため、10,000 より小さくしないでください。同様に `MaxScanRows:` の値も、50,000 より小さい値に設定する必要はありません。

PERFORCE コマンドと関連するファイル数との対応など、さらに詳しい情報は、コマンドラインから、

```
p4 help maxresults
p4 help maxscanrows
p4 help maxlocktime
```

を実行して、見ることができます。

複数グループのユーザを対象とする MaxResults、MaxScanRows および MaxLockTime

前述のように、ユーザが複数のグループに登録されていれば、所属するすべてのグループの MaxResults 制限値の最大値がそのユーザに適用されます。デフォルト値 unlimited は数値上の制限ではありません。MaxResults（、MaxScanRows または MaxLockTime）が“`unlimited`”に設定されているグループに属しているユーザであっても、他のグループに所属していればそのグループの MaxResults（、MaxScanRows または MaxLockTime）値の最大値によって制限されます。ユーザのコマンドが実際に無制限になるのは、そのユーザがどのグループにも属さないとき、あるいは、そのユーザが属するすべてのグループの MaxResults が“unlimited”に設定されているときだけです。

この動作の結果、スーパー・ユーザのグループとして `unlimited` を割り当てたグループを作成することはできません。このグループに属するユーザのいずれかが MaxResults の数値上の制限をもつ別のグループにも属している場合、他のグループに設定されている数値上の制限の最大値が適用される制限値となります。この現象に対して有効な回避策は、きわめて大きな MaxResults 値をスーパー・ユーザのグループに割り当てることです。

例 :

```
Group:        superusers
Maxresults:   10000000
Maxscanrows:  100000000
MaxLockTime:  100000000
```

（設定可能な `MaxResults`、`MaxScanRows` または `MaxLockTime` の最大値は、プラットフォームによって異なります。ほとんどのプラットフォームで 32 ビット整数です。）

効率のいいスクリプトを作成する

PERFORCE のコマンドライン・クライアントである `p4` は、対話形式で実行できることをすべてスクリプト内で実行可能にします。PERFORCE サーバは、ユーザが発行するよりも速くコマンドを処理できるため、完全な対話形式の環境での応答時間については優れています。しかし、スクリプト -- 例えば、トリガ、レビュー・デーモン、コマンド・ラッパ -- によって発行された `p4` コマンドは、効率に注意しておかないとパフォーマンス上の問題を引き起こす可能性が

あります。p4 コマンドが、本来非効率的だからではなく、ユーザが対話しながら処理を行うときの p4 の実行方法が、繰り返しの操作に必ずしも適していないからです。

このセクションでは、よく発生する効率上の問題とその解決法をいくつか紹介します。

ファイルに対する操作の繰り返し

発行された PERFORCE の各コマンドにより接続スレッドが生じると、p4d サブプロセスを起動します。スクリプトによって実行する PERFORCE コマンドの数を減らすことは、スクリプトの効率をよくする第一歩です。

スクリプトでは、各ファイルに対して同じ PERFORCE コマンドを繰り返し実行するような記述を避けてください。ファイルのリストに対して 1 つの PERFORCE コマンドを実行し、その実行結果に対して繰り返し処理を行うことにより上記のような操作が可能であれば、この方法を採用してください。

例えば、より効率的な手法として、次のようにしてください。

```
for i in `p4 diff2 path1/... path2/...`
do
    [process diff output]
done
```

以下のような効率の悪い方法は避けてください。

```
for i in `p4 files path1/...`
do
    p4 diff2 path1/$i path2/$i
    [process diff output]
done
```

リスト入力ファイルの利用

PERFORCE コマンドが、コマンドラインの引数にファイルのリストを指定できる場合、あるファイルから同じ引数のリストを読み取ることもできます。最初にファイルのリストを作成し、そのリスト・ファイルを p4 -x に引き渡すことにより、スクリプトにそのリスト入力ファイルを利用することができます。

例えば、スクリプトが以下のように記述されている場合、

```
for components in header1 header2 header3
do
p4 edit ${component}.h
done
```

次のように記述するとより効率がよくなります。

```
for components in header1 header2 header3
do
    echo ${component}.h >> LISTFILE
done
p4 -x LISTFILE edit
```

-x オプションは、p4 に対して指定のファイルから 1 行ずつ引数を読み取ることを指示します。ファイル名が“–”（ダッシュ）の場合は、標準入力が読み取られます。

ブランチ・ビューの使用

ブランチ・ビューは、p4 integrate または p4 diff2 で、PERFORCE コマンドの実行回数を減らすために使用できます。例えば、次のコマンドを実行するスクリプトがあれば、

```
p4 diff2 pathA/src/...   pathB/src/...
p4 diff2 pathA/tests/... pathB/tests/...
p4 diff2 pathA/doc/...   pathB/doc/...
```

次のようなブランチ・ビューを作成することにより、効率をよくすることができます。

```
Branch:        pathA-pathB
View:
        pathA/src/...      pathB/src/...
        pathA/tests/...    pathB/tests/...
        pathA/doc/...      pathB/doc/...
```

そして上記の 3 つのコマンドを次のように 1 つにまとめることができます。

```
p4 diff2 -b pathA-pathB
```

ラベル参照を制限する

大きなラベルへの反復参照は特に無駄を増やします。リビジョンにラベルを使用するファイル参照コマンドは、個々のファイルに対して、ラベル全体の走査が行われることになります。PERFORCE サーバに無駄な動作をさせないために、サーバからラベルの適用されているファイルを取り出し、取り出されたファイルに対して必要なファイルを走査するスクリプトにした方がよいでしょう。

例えば、次のようなスクリプトの方が、

```
p4 files path/...@label | egrep “path/f1.h|path/f2.h|path/f3.h”
```

次のようなスクリプトや、

```
p4 files path/f1.h@label path/f2.h@label path/f3.h@label
```

次のようなスクリプトより PERFORCE サーバに与える負荷が軽くなります。

```
p4 files path/f1.h@label
p4 files path/f2.h@label
p4 files path/f3.h@label
```

次に説明する“一時クライアント・ワークスペース”の活用も、ラベルを使ってファイルを参照する回数を減らすのに役立ちます。

一時クライアント・ワークスペースの活用

ほとんどの PERFORCE コマンドは、現在のワークスペース・ビューのすべてのファイルを 1 つの

コマンドライン引数で処理できます。作業したいファイルだけが含まれている一時クライアント・ワークスペース・ビューを利用すれば、実行するコマンドの数や各コマンドに与える必要のあるファイル引数の数を減らすことができます。

例えば、次のコマンドを実行するスクリプトを考えます。

```
p4 sync pathA/src/...@label
p4 sync pathB/tests/...@label
p4 sync pathC/doc/...@label
```

次のようなクライアント・ワークスペース仕様を使えば、1 度にまとめてコマンドを実行し、ラベル走査を 3 回から 1 回に減らすことができます。

```
Client:        XY-temp
View:
        pathA/src/...      //XY-temp/pathA/src/...
        pathB/tests/...    //XY-temp/pathB/tests/...
        pathC/doc/...      //XY-temp/pathC/doc/...
```

このワークスペース仕様を使って実行するときは、次のように入力します。

```
p4 -c XY-temp sync @label
```

効率的に圧縮を使用する

デフォルトでは、ファイルタイプが `binary` のファイル・リビジョンは、PERFORCE サーバに保存される際に圧縮されます。

ファイル・フォーマットの一部に圧縮を含むファイル（例えば、.GIF および .JPG イメージ、.MPG および .AVI メディア・コンテンツ、`.gz` や `.ZIP` で圧縮されたファイル）があります。PERFORCE サーバでこのようなファイルを圧縮しようとすると、ディスク容量の節約がほとんどできない割に、サーバの CPU リソースを消費する結果となります。

これらのファイルタイプに対してサーバ領域の圧縮の無効にするには、コマンドラインまたは `p4 typemap` テーブルを用いて、ファイルタイプとして `binary+F`（圧縮せずにサーバへ格納するバイナリ）を指定します。

サンプルのタイプマップ・テーブルを含め、`p4 typemap` についての詳細は 46 ページの「p4 typemap でファイルタイプを定義する」をご覧ください。

データベース・ツリーのバランス回復のためのチェックポイント

PERFORCE の内部データベースでは、データは Bayer ツリー（通称 : B ツリー）という構造で保存されます。B ツリーは高速アクセスのためのきわめて一般的なデータ構成方法ですが、時間の経過とともに、ツリーに対するエレメントの追加、削除が行われ、しだいにデータ構造がアンバランスになってくることがあります。

最終的には、そのツリーがアンバランスになるあまり、パフォーマンスに悪い影響が出ることもあります。PERFORCE のチェックポイントと復旧のプロセス（25 ページの「バックアップとリカバリの考え方」をご覧ください）を実行すれば、ツリーはバランスのとれた状態に再生されます。結果的に、バックアップ、`db.*` ファイルの削除、チェックポイントからの `db.*` ファイルの再生の後、サーバのパフォーマンスにある程度の改善が見られるでしょう。

このツリーのバランス回復は、通常はデータベース・ファイルがチェックポイントのサイズの 10 倍以上になったときに初めて有効になります。PERFORCE を正常に使用しているときは、ツリーがアンバランスになるまでに必要な時間の長さから考えて、大多数のサイトではパフォーマンス改善のためにチェックポイントからのデータベースの復旧（ツリーのバランス回復）を行う必要はないでしょう。

第 8 章　PERFORCE と Windows

本章では、Windows 上で PERFORCE サーバをセットアップし、保守する管理者にとって重要な情報を記述します。

> **注** *異なる記述のない限り、本章の内容は Windows NT、Windows 2000 および Windows XP にも同様に適用されます。*

PERFORCE インストーラの使用

PERFORCE のインストーラ・プログラム、`perforce.exe` を実行すると、ユーザとして（PERFORCE コマンドライン・クライアント）、通常の管理者として（PERFORCE は Windows サービスとしてインストールされる）またはカスタムの管理者として（PERFORCE はカスタマイゼーション・オプションを加えたサービスとしてインストールされる）PERFORCE をインストールしたり、PERFORCE をシステムからアンインストールしたりすることができます。

P4Win や P4V など、他の PERFORCE クライアント・プログラムには別のインストーラが用意されています。

Administrator の特権を持っている場合は、通常、PERFORCE をサービスとしてインストールするのが最善の選択です。Administrator の特権を持っていない場合は、サーバとしてインストールしてください。

Windows 2000 以上で使用する場合、サービスとして PERFORCE をインストールするには Administrator の権限が必要です。また、サーバとして PERFORCE をインストールするには Power User の権限が必要です。

アップグレード・ノート

PERFORCE インストーラは、97.3 より前のバージョンも含め、あらゆるタイプの PERFORCE サーバ（またはサービス）を自動的にアップグレードします。このアップグレードはきわめて安全な方法で実行されます。途中で何らかの処理がうまく行えなかった場合、インストーラはアップグレードを停止します。その後も、従来のサーバ（またはサービス）を引き続き使用することができます。

インストーラ使用時のオプション

インストーラを起動すると、初期画面に対象の PERFORCE ソフトウェアのリビジョンが一覧表示されます。次に、以下の操作のいずれかを選択します。

- ユーザとしてのインストール（user install）
- 管理者としての標準インストール (typical administrator install)
- 管理者としてのカスタム・インストール（custom administrator install）

- PERFORCE のアンインストール（deinstall）

ユーザとしてのインストール

“user install”オプションでは PERFORCE コマンドライン・クライアント（p4.exe）のみがインストールされます。

その他の Windows 環境の PERFORCE クライアントである PERFORCE ビジュアル・クライアント（P4V）や PERFORCE Windows クライアント（P4Win）、およびサードパーティのプラグインをダウンロードおよびインストールする方法については、PERFORCE ウェブサイトの Downloads のページをご覧ください。

```
http://www.perforce.com/perforce/loadprog.html
```

インストールに際しては、クライアント実行ファイルの保存場所、クライアントが PERFORCE サーバに接続するためのポート（P4PORT）、デフォルト・エディタ、デフォルト・ユーザ名の指定が要求されます。

クライアントが使用するポートを指定するときには、*hostname*:*port* という形式にし、ホスト名を含めるのを忘れないでください。P4PORT の設定方法については、15 ページの「PERFORCE クライアント・プログラムにサーバの接続ポートを指定する」をご覧ください。

インストーラがマシン上の古いバージョンの PERFORCE クライアントまたはサーバ・ソフトウェアを検出した場合は、PATH に基づく衝突を防ぐために古い実行ファイルをリネームする選択肢が与えられます。

管理者としての標準インストール

“typical administrator install”では、PERFORCE のクライアントとサーバ両方のソフトウェアをインストールします。この操作には、Administrator の権限が必要です。

インストールに際しては、クライアントおよびサーバの実行ファイルのディレクトリ、PERFORCE サーバまたはサービスがクライアントからの接続要求待ちをするローカル・マシン上のポート（P4PORT）、デフォルト・エディタ、デフォルト・ユーザ名の指定が要求されます。

インストーラは、P4LOG エラー・ログ・ファイルと journal ファイルのデフォルトの保存場所を選択します。マシンに古いバージョンの PERFORCE がインストールされている場合は、従来使用されていた保存場所が選択されます。

管理者特権がある場合、インストーラは PERFORCE をインストールし、自動開始サービスとして構成します。このサービスは、PERFORCE のインストール完了後にセットアップされ、自動で開始します。以後、マシンがリブートされるたびに自動で再開します。管理者特権がない場合は、PERFORCE をサーバとして起動するためのショートカットが、スタート・メニューに作成されます。

インストーラがマシン上の古いバージョンの PERFORCE クライアントまたはサーバ・ソフトウェアを検出した場合は、PATH に基づく衝突を防ぐために古い実行ファイルをリネームする選択肢が与えられます。

管理者としてのカスタム・インストール

“custom administrator install”では、特定のカスタマイズを行って PERFORCE のクライアントとサーバ両方のソフトウェアをインストールします。この操作には、Administrator の権限が必要です。

インストールに際しては、管理者としての標準インストールの場合と同様にクライアントおよびサーバの実行ファイルの保存場所、PERFORCE サーバまたはサービスがクライアントからの接続要求待ちをするローカル・マシン上のポート、デフォルト・エディタ、デフォルト・ユーザ名の指定が要求されます。

管理者としての標準インストールの場合と異なり、オプションでクライアントおよびサーバの実行ファイル用のディレクトリを別々に指定することができます。さらに、サーバ・ルート、サーバ・ポートに加えて、PERFORCE を自動開始（または手動開始）のサービスまたはサーバ・プロセスのどちらにするかなどの指定も行えます。既存の `P4LOG` ファイルおよび `journal` ファイルの保存場所が参照用に表示され、後で `p4 set` を用いて変更することができます。

別の PERFORCE サーバ動作中に PERFORCE サービスをインストールしようとすると、次のエラー・メッセージが表示されます。

```
Setup has determined that a PERFORCE Server could be running. Please
shut down all PERFORCE Servers before continuing the installation.
```

（別の PERFORCE サーバが動作中の可能性があります。インストールを続行する前に、すべての PERFORCE サーバを終了させてください。）

動作中の PERFORCE サーバ・プロセスを終了させるのを怠ると、新たにインストールされたサービスと既存のサーバとの間で衝突が発生します。

インストールに際しては、他のインストール操作の場合と同様に、インストーラがマシン上の古いバージョンの PERFORCE クライアントまたはサーバ・ソフトウェアを検出した場合は、`PATH` に基づく衝突を防ぐために古い実行ファイルをリネームする選択肢が与えられます。

PERFORCE のアンインストール

Windows マシンから PERFORCE をアンインストールするには、**Uninstall** を選択します。この操作には、Administrator の権限が必要です。

アンインストールにより、サーバ・データを*除き*、PERFORCE サーバ、サービス、クライアントの実行ファイル、レジストリ・キー、サービス・エントリなどがすべて削除されます。ただし、サーバ・ルートのデータベースおよびディポ・ファイルは保持されます。

スクリプト化された展開と自動インストール

PERFORCE インストーラはスクリプト化されたインストールをサポートしています。これにより、PERFORCE を多数のデスクトップに迅速に展開することができます。

スクリプト化されたインストールは、PERFORCE インストーラのスクリプト可能なバージョンに付属している設定ファイルで制御します。このファイルを編集すると、PERFORCE の環境変数（`P4PORT` など）の事前設定、サイトで使用する PERFORCE クライアント・プログラムの自動選択などを行うことができます。

PERFORCE の展開を自動化する方法については、次のページの Tech Note 68 をご覧ください。

```
http://www.perforce.com/perforce/technotes/note068.html
```

PERFORCE テクニカル・サポートのスタッフは、PERFORCE 展開の自動化についてのいかなるご質問、お問い合わせにもお答えします。

Windows サービスと Windows サーバ

Windows *サーバ*としてタスクを実行するには、ユーザ・アカウントのログインが必要です。ユーザ・アカウントにログインしないと、ユーザの **[スタートアップ]** フォルダのショートカットを実行することができません。一方、Windows *サービス*はマシン起動時に自動的に開始し、ユーザがそのマシンにログインしているかどうかに関係なく動作します。

たいていどの文書でも、“PERFORCE サーバ”も“`p4d`”も、“データベースを管理し、PERFORCE のクライアントからの要求に応じるバック・エンドのプロセス”を指す言葉として使われています。Windows では、このような表現では不明確かも知れません。このバックエンド・プロセスは、サービス（スレッドとして動作する `p4s.exe`）としても、サーバ（レギュラー・プロセスとして動作する `p4d.exe`）としても実行されることがあるからです。Windows の管理者

の観点からは、これらは重要な違いです。本章では、そのような用語を厳密に区別して使用します。

PERFORCE サービス（p4s.exe）と PERFORCE サーバ（p4d.exe）の実行ファイルは相互にコピーになっています。ファイル名が違うだけで、この 2 つは全く同じものです。実行時、これらは起動に使用した名前の最初の 3 文字（p4s または p4d）で動作を決定します。例えば、p4smyservice.exe という名前のコピーを起動すればサービスとして、p4dmyserver.exe という名前のコピーを起動すればサーバとして起動します。

PERFORCE サービスの開始と停止

PERFORCE がサービスとしてインストールされていれば、Administrator の特権を持つユーザは［コントロールパネル］の［サービス］アプレットを用いてそれを起動、停止することができます。

リリース99.2以降を動作させる場合は、次のコマンドを使ってもサービスを停止することができます。

```
p4 admin stop
```

PERFORCE サーバの起動と停止

PERFORCE がサーバとしてインストールされている場合は、［スタート］メニューに“PERFORCE Server”のショートカットがあるはずです。サーバを起動するには、このショートカットをダブルクリックします。サーバを停止するには、タスクバーの“PERFORCE Server”ボタンを右クリックし、［閉じる］を選択します。

PERFORCE サーバはコマンド・プロンプトから手動で起動することもできます。サーバ実行ファイル p4d.exe は、通常は P4ROOT のディレクトリにあります。サーバを起動するには、まず現在の P4ROOT、P4PORT、P4LOG、P4JOURNAL の設定が正しいことを確認してから、%P4ROOT%\p4d を実行します。

P4ROOT、P4PORT、P4LOG、または P4JOURNAL の設定とは異なる設定によってサーバを起動したい場合は、p4d コマンドライン・オプションを使用します。例えば、

```
c:¥test¥p4d -r c:¥test -p 1999 -L c:¥test¥log -J c:¥test¥journal
```

とすると、ルート・ディレクトリ c:¥test、接続待ちポート 1999、エラー・ログ c:¥test¥log、ジャーナル・ファイル c:¥test¥journal という設定で PERFORCE サーバ・プロセスが起動します。

p4d コマンドライン・オプションでは大文字と小文字が区別されることに注意してください。

リリース 99.2 以降を動作させている場合は、コマンド、

```
p4 admin stop
```

を使っても PERFORCE サーバを停止することができます。

> **注** *99.1 以前のリリースを動作させている場合は、コマンド・プロンプト・ウィンドウに Ctrl-C と入力するか、コマンド・プロンプト・ウィンドウの［閉じる］アイコンをクリックします。*
>
> *このような PERFORCE サーバの停止方法は、PERFORCE のすべてのバージョンで有効ですが、必ずしも「いい方法」とはいえません。リリース 99.2 以降では* p4 admin stop *コマンドが使えるようになっているので、この方法はもはやお勧めできません。*

ネットワーク・ドライブへの PERFORCE サービスのインストール

デフォルトでは、PERFORCE サービスはローカル・システム・アカウントの下で動作します。ローカル・システム・アカウントには、ネットワーク・アクセス権がないので、メタデータとディポ・ファイルがネットワーク・ドライブに保存されている場合、PERFORCE サービスを機能させるには、正規のユーザ ID とパスワードが必要です。

ネットワーク・ドライブにサーバ・ルートをインストールする場合、PERFORCE インストーラ (`perforce.exe`) はインストールの時点で有効なユーザ ID とパスワードの組み合わせを要求します。ユーザは管理者の特権を持っていなければなりません。(サービスは、動作しているときには、`System` ではなく、このユーザの名前のもとに動作します。)

PERFORCE サービスはネットワーク・ドライブをサーバ・ルートとして使用して信頼できる動作をしますが、ネットワーク・トラフィックが増加し、ファイル・アクセスが遅くなるため、やはりパフォーマンスは著しく低下します。このため、ディポ・ファイルと PERFORCE データベースは、PERFORCE サービスが動作しているマシンのローカル・ドライブに配置することをお勧めします。

Windows 環境下での複数の PERFORCE サービスの使用

デフォルトでは、Windows 用の PERFORCE インストーラは単一の PERFORCE サーバを単一のサービスとしてインストールします。同一のマシン上で複数の PERFORCE システムを (例えば、1 つを開発用として、1 つをテスト用として) 動作させたい場合には、コマンドラインから手動で PERFORCE サーバを起動するか、PERFORCE に付属のユーティリティ `svcinst.exe` を用いて追加の PERFORCE サービスを設定することができます。

> *サポートするユーザの数を増やす目的で、ユーザ・ライセンスを追加購入せずに複数のサービスをセットアップすることは、PERFORCE ライセンス契約の合意条項に違反します。*

PERFORCE の構成を決定する際に環境変数が優先されることを理解していると、同一マシン上で複数の PERFORCE サービスを構成するときに役に立ちます。操作を始める前に 126 ページの「Windows 環境下での構成パラメータの優先順位」を読んで理解しておいてください。

2 番目の PERFORCE サービスをセットアップするには :

1. PERFORCE サービス用にディレクトリを生成します。
2. サーバ実行ファイル、サービス実行ファイル、およびライセンス・ファイルを上記のディレクトリにコピーします。
3. 下記の例のように、`svcinst.exe` ユーティリティを用いて新しい PERFORCE サービスを生成します。(`svcinst.exe` ユーティリティは PERFORCE インストーラに付属しており、PERFORCE サーバ・ルートに格納されています。)
4. 環境変数を設定し、新しいサービスを開始します。

最初の PERFORCE サービスは PERFORCE インストーラを用いてインストールすることをお勧めします。この最初のサービスを“PERFORCE”といい、そのサーバ・ルート・ディレクトリに、当該マシン上に生成する他の PERFORCE サービスによって必要とされるファイルが格納されています。

例 : *2 番目の PERFORCE サービスの追加*

2 番目の PERFORCE サービスを、`C:\p4root2` にルートを置いて、“PERFORCE2”というサービス名で生成するものとします。`svcinst` 実行ファイルは `C:\PERFORCE` にインストールした最初の PERFORCE システムのサーバ・ルートにあります。

p4d.exe *実行ファイルがリリース 99.1/10994 以降であることを確認します。*

```
p4d -V
```

99.1/10994 より前のリリースを使用している場合には、以下の手順を実行する前に、まず http://www.perforce.com *からそれ以降のリリースをダウンロードし、サーバをアップグレードしなければなりません。*

新しいサービス用の P4ROOT *ディレクトリを生成します。*

```
mkdir c:¥p4root2
```

サーバ実行ファイル – p4d.exe *（サーバ）と* p4s.exe *（サービス）の両方 およびライセンス・ファイルを上記の新ディレクトリにコピーします。*

```
copy c:¥perforce¥p4d.exe c:¥p4root2
copy c:¥perforce¥p4d.exe c:¥p4root2¥p4s.exe
copy c:¥perforce¥license c:¥p4root2¥license
```

PERFORCE の svcinst.exe *（サービス・インストーラ）を使用して "*PERFORCE2*" サービスを生成します。*

```
svcinst create -n PERFORCE2 -e c:¥p4root2¥p4s.exe -a
```

*"*PERFORCE2*" サービスを生成したら、"*PERFORCE2*" サービス用のサービス・パラメータを設定します。*

```
p4 set -S PERFORCE2 P4ROOT=c:¥p4root2
p4 set -S PERFORCE2 P4PORT=1667
p4 set -S PERFORCE2 P4LOG=log2
p4 set -S PERFORCE2 P4JOURNAL=journal2
```

*最後に、PERFORCE サービス・インストーラを使用して "*PERFORCE2*" サービスを開始します。*

```
svcinst start -n PERFORCE2.
```

これで 2 番目のサービスも稼動状態になりました。次にリブートすると、1 番目と 2 番目の両方のサービスが自動的に開始します。

Windows 環境下での構成パラメータの優先順位

Windows では、PERFORCE 構成パラメータをさまざまな形で設定することができます。PERFORCE クライアント・プログラム（例えば p4 や p4v）、または PERFORCE サーバ・プログラム（p4d）が起動されると、以下の優先順位にしたがって構成パラメータを読み込みます。

1. プログラムのコマンドライン・オプション（最優先）
2. P4CONFIG ファイル（P4CONFIG が設定されている場合）
3. ユーザ環境変数
4. システム環境変数
5. PERFORCE ユーザ・レジストリ（p4 set によって設定）
6. PERFORCE システム・レジストリ（p4 set -s によって設定）

PERFORCE サービス（p4s）は、開始すると以下の優先順位にしたがって構成パラメータを読み込みます。

1. Windows サービス・パラメータ（p4 set -*S servicename* で設定）（最優先）
2. システム環境変数
3. PERFORCE システム・レジストリ（p4 set -s によって設定）

ユーザ環境変数は、以下のいずれかの手段によって設定します。

- MS-DOS の `set` コマンド
- `AUTOEXEC.BAT` ファイル
- メニュー操作（**[コントロールパネル] → [システム] → [詳細] → [環境設定] →ユーザ環境変数**）

システム環境変数は、以下のメニュー操作により設定します。

- メニュー操作（**[コントロールパネル] → [システム] → [詳細] → [環境変数] →システム環境変数**）

Windows に関連した動作不安定の解消

PERFORCE が Windows 上で問題なく動作している大規模なサイトは、数多くあります。その一方で、PERFORCE サービスまたはサーバの動作が不安定なサイトもあります。サーバが不可解に停止したり、サービスが開始しなかったり、極端な場合は、システムがクラッシュしたりすることもあります。このような場合、通常マシンに加えられた変更か、オペレーティング・システムの欠陥が原因になっています。

PERFORCE の故障がすべて OS に起因するわけではありません。しかし、システムがクラッシュする、エラーログの記録や Windows によるサーバ・クラッシュの表示もないまま PERFORCE サーバが終了してしまう、PERFORCE サービスが正常に開始しないというような現象は、OS の欠陥を示すものと考えられます。

PERFORCE は、Windows 2000 Intel x86、Windows XP Intel x86、Windows Server 2003 を始め、Windows NT4.0 sp6a 以降でサポートされています。

サービスパックのインストール*後*に他のソフトウェアをインストールすると、サービスパックによってインストールされた重要なファイルが上書きされることがあります。そのような場合は、最後にインストールしたサービスパックを再インストールすることにより問題が解消することがしばしばあります。サービスパックのインストール後に別のアプリケーションをインストールしたらサーバの動作が不安定になったと思われる場合は、サービスパックの再インストールを検討してください。

それでも問題を解決できない場合は、PERFORCE を他のシステムにインストールし、同じ問題が発生するかどうかを見たり、問題が発生したシステムに OS と PERFORCE を再インストールしてみたりすることも問題解決に役立つ可能性があります。

P4EDITOR、P4DIFF 使用上の問題点

Windows ユーザにとっては、`P4EDITOR` または `P4DIFF` 環境変数を使用している場合、PERFORCE コマンドライン・クライアント（`p4.exe`）を使用しにくいことがあるかもしれません。

これは、PERFORCE クライアントがユーザ指定のエディタや差分ユーティリティなどのプログラムの起動に DOS シェル（`cmd.exe`）を使用する場合があるからです。DOS シェルは、Windows コマンド（例えば、`notepad.exe` のような GUI ベースのエディタ）が実行されているときでも、そのコマンドの実行完了を待たずに終了することができます。そのような状態で DOS シェルが終了すると、PERFORCE クライアントは Windows コマンドの実行が完了したと判断して処理を進めようとします。その結果、Windows のエディタや差分ユーティリティが使用していた一時ファイルが削除されてしまうことがあります。そうすると、一時ファイルが見つからないというエラー・メッセージが出たり、他のおかしな現象が発生することになります。

この問題は、次の 2 つの方法で回避することができます。

- 環境変数 `SHELL` の設定を解除します。Windows で動作している PERFORCE クライアントは、`SHELL` が設定されているときのみ `cmd.exe` を使用します。それ以外の場合は、`spawn()` を起動して、Windows プログラムの完了を待ちます。

- P4EDITOR または P4DIFF の変数の名前に、以下のコマンドが入っているバッチファイルの名前を指定します。

  ```
  start /wait program %1 %2
  ```

  ここで、*program* は起動したいエディタまたは差分ユーティリティの名前です。/wait オプションの指定により、システムはエディタまたは差分ユーティリティの終了を待つようになり、PERFORCE クライアントは正常に動作することができます。

  Windows 用エディタの中には、待機の指示があっても正常に動作しないものがあります (Wordpad が代表例)。現在、そのようなプログラムへの対策はありません。

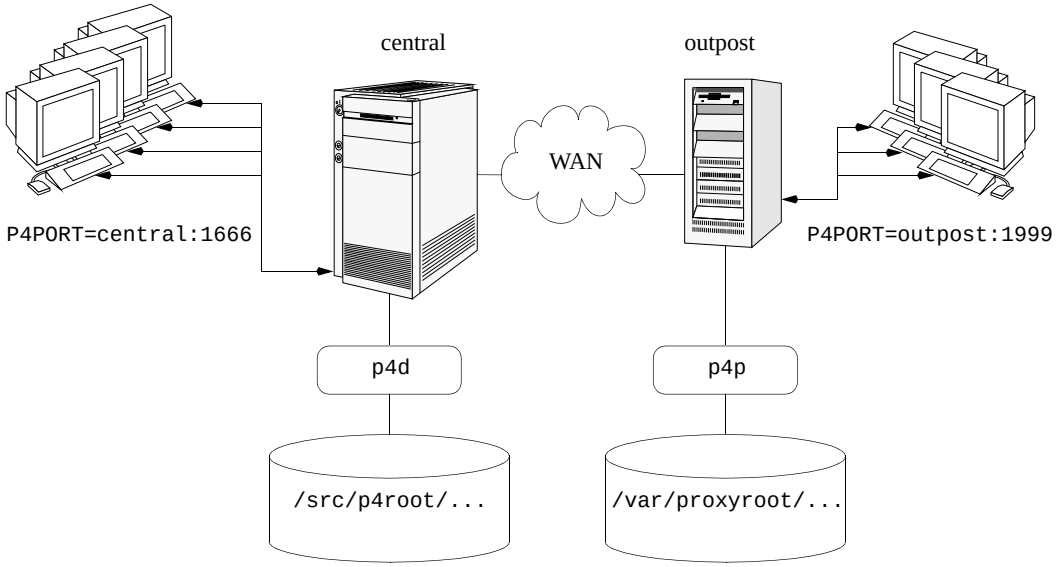
第 9 章

PERFORCE プロキシ

PERFORCE は、広範なネットワーク形態における分散開発を取り扱えるようになっています。分散拠点へのバンド幅が限られている場合に、P4P（PERFORCE プロキシ）は頻繁に転送されるファイル・リビジョンをキャッシュすることによって PERFORCE クライアント／サーバ間を調整し、パフォーマンスを改善します。キャッシュされたファイル・リビジョンに対する要求を横取りし、P4P は PERFORCE サーバおよびネットワークへの要求を減らします。

複数の PERFORCE クライアントが、WAN を通じて中央の PERFORCE サーバへアクセスする際のパフォーマンスを改善するためには、クライアントに近い側のネットワークに P4P を構成し、その P4P にアクセスするクライアントを構成し、さらに中央の PERFORCE サーバへアクセスする P4P を構成します。（LAN においては、プロキシをセットアップすることによって、中央のサーバにおける CPU やディスクの負荷を軽減させることができます。）

次の図は、典型的な P4P の構成を表しています。



この図において、リモート開発拠点のユーザによって要求されたファイル・リビジョンは、まず中央の PERFORCE サーバ（`central` で動作している `p4d`）から取り出され、比較的遅い WAN を通じて転送されます。しかし、同一のリビジョンに対する次の要求は、PERFORCE プロキシ（`outpost` で動作する `p4p`）で処理されます。これは、リモート開発拠点の LAN を使用した転送であり、WAN におけるネットワーク・トラフィックと中央サーバの CPU 負荷の両方を軽減していることになります。

システム要求

PERFORCE プロキシを使用するには、以下が必要です。

- リリース 2002.2 以降の PERFORCE サーバ
- ファイル・リビジョンのキャッシュを保持するための、プロキシ・ホストにおける十分なディスク容量

P4P をインストールする

UNIX

UNIX 上に P4P をインストールするには、次のように実行します。

1. プロキシを実行したいマシンに、p4p の実行形式をダウンロードします。
2. キャッシュするファイル・リビジョンを保持するためのディレクトリ（P4PCACHE）を、このマシン上に用意します。
3. PERFORCE クライアント・プログラムからの要求を受け付けるためのポート（P4PORT）を決定します。
4. このプロキシがキャッシュするためのターゲット PERFORCE サーバ（P4TARGET）を決定します。

Windows

インストーラの custom/administrator インストール・ダイアログから、P4P をインストールします。

P4P を実行する

P4P を起動するには、環境変数もしくはコマンドライン・オプションを設定し p4p を実行します。コマンドラインにおいて設定したオプションは、環境変数による設定をオーバライドします。

例えば、次のコマンドラインを実行することで起動するプロキシは、central という名前のホストにおいてポート 1666 で接続待ちをしている PERFORCE サーバと通信します。

```
p4p -p 1999 -t central:1666 -r /var/proxyroot
```

この P4Proxy を使用するためには、PERFORCE クライアントは、プロキシが起動しているホストにおいてポート 1999 で接続待ちをしている P4P に接続します。P4P がキャッシュするファイル・リビジョンは、/var/proxyroot という名前のディレクトリに保持されます。

Windows サービスとして起動する

P4P を Windows のサービスとして起動するには、P4P を Windows インストーラからインストールするか、p4p.exe 起動時に -s オプションを指定するか、P4P 実行ファイルを p4ps.exe にリネームします。環境変数が設定されていない場合は、P4POPTIONS 変数で p4p.exe または p4s.exe 起動時に指定するコマンドライン・オプションを設定します。

P4P のオプション

PERFORCE プロキシのコマンドライン・オプションとして、以下がサポートされます。

	意味
`-c`	PERFORCE サーバから P4P へ転送されるファイルを圧縮しません。(ネットワークのバンド幅に少し余裕があるときには、このオプションによって中央サーバの CPU 負荷を軽減することができます。)
`-d`	デーモンとして起動します。― まず `fork` してから実行します。(UNIX のみ)
`-f`	`fork` しません。― シングル・スレッド・サーバとして起動します。(UNIX のみ)
`-i`	`inetd` として起動します。(`stdin/stdout` 上の `socket` ― UNIX のみ)
`-q`	起動時のメッセージを抑止します。
`-s`	NT サービスとして起動します。(Windows のみ) `p4p.exe -s` と起動するのは、`p4ps.exe` を起動するのと同じです。
`-e` *size*	

次のグローバル・オプションがサポートされます。

	意味
`-h` または `-?`	ヘルプ・メッセージを表示します。
`-L` *logfile*	ログファイルの場所を指定します。 デフォルトは `P4LOG` であり、もし `P4LOG` が設定されていない場合は、`p4p` が起動されたディレクトリとなります。
`-p` *port*	PERFORCE クライアント・プログラムからの要求を受け付けるためのポートを指定します。 デフォルトは `P4PORT` であり、もし `P4PORT` が指定されていない場合は、1666 となります。
`-r` *root*	リビジョンをキャッシュするためのディレクトリを指定します。 デフォルトは `P4PCACHE` であり、もし `P4PCACHE` が指定されていない場合は、`p4p` が起動されたディレクトリとなります。
`-t` *port*	ターゲット PERFORCE サーバのポートを指定します。(このサーバに対して、P4P はプロキシとして動作します。) デフォルトは `P4TARGET` であり、もし `P4TARGET` が指定されていない場合は、`perforce:1666` となります。
`-v` *level*	サーバのトレース・レベルを指定します。デバッグのメッセージがプロキシ・サーバのログファイルに格納されます。`p4p` からのデバッグ・メッセージは `p4d` に渡されず、`p4d` からのデバッグ・メッセージは `p4p` のインスタンスに渡されません。 デフォルトは `P4DEBUG` であり、もし `P4DEBUG` が指定されていない場合は、設定なしとなります。
`-V`	PERFORCE プロキシのバージョンを表示します。

P4P を管理する

バックアップは不要

P4P のキャッシュ・ディレクトリは、一切バックアップの必要はありません。もし必要ならば、P4P は PERFORCE サーバのメタデータに基づいたキャッシュを構築します。

P4P を終了する

P4P は事実上、独立したプロセスとして動作します。したがって、UNIX 上で p4p を終了するには SIGTERM または SIGKILL を指定して kill コマンドを実行します。Windows 上では、[タスク マネージャ] から [プロセスの終了] を選択します。

ディスク容量の消費を管理する

P4P は、キャッシュ・ディレクトリ配下にファイル・リビジョンをキャッシュします。これらのリビジョンは、ユーザが削除しない限り蓄積され続けます。P4P は自身のキャッシュ・ファイルを削除しませんので、別の方法によってディスクの消費を管理する必要があります。

> もし、ユーザによってキャッシュ・ファイルが削除されないと、ディスク容量はついには枯渇します。ディスクの空きを作るには、プロキシのルート配下にあるファイルを削除します。これらのアーカイブ・ファイルを削除するのは、プロキシが動作中であっても構いません。

PERFORCE クライアントがプロキシを使用しているかどうかを判断する

もし PERFORCE クライアントプログラムがプロキシを使用しているならば、p4 info の出力にプロキシのバージョン情報が表示されます。

例えば、PERFORCE サーバが central:1666 で起動しており、PERFORCE クライアントが outpost:1999 で起動している PERFORCE プロキシに接続しているとしたら、p4 info の出力は次のようになります。

```
$ export P4PORT=outpost:1999
$ p4 info
User name: p4adm
Client name: admin-temp
Client host: remotesite22
Client root: /home/p4adm/tmp
Current directory: /home/p4adm/tmp
Client address: 192.168.0.123:55768
Server address: central:1666
Server root: /src/p4root
Server date: 2002/10/14 15:03:05 -0700 PDT
Server version: P4D/FREEBSD4/main/36609 (2002/09/30)
Proxy version: P4P/SOLARIS26/main/36884 (2002/10/14)
Server license: P4 Admin <p4adm> 20 users (expires 2003/02/01)
```

P4P とプロテクション

PERFORCE プロキシを使うユーザの IP アドレスをプロテクション・テーブルに対して指定するには、ワークステーションの IP アドレスに proxy- の文字列を付加します。

例えば、192.168.10.0/24 のサブネット上のワークステーションでリモート開発を行っている組織を考えます。この組織にはローカル開発を行っている中央のオフィスもあり、中央のオフィスは 10.0.0.0/8 のサブネットに属しています。PERFORCE サーバは 10.0.0.0/8 のサブネットにあり、PERFORCE プロキシは 192.168.10.0/24 のサブネットにあります。リモートサイトのユーザはグループ remotedev に属しており、ときおり中央のサーバに対してアクセスする場合もあるとします。

プロテクション・テーブルに次のような行を追加することにより、グループ remotedev のメンバーは確実に、リモートサイトで作業する間はプロキシを使用し、かつローカルサイトで作業するときにはプロキシを使用しないことになります。

```
list    group    remotedev     192.168.10.*          -//...
write   group    remotedev     proxy-192.168.10.*     //...
list    group    remotedev     proxy-10*             -//...
write   group    remotedev     10.*                   //...
```

最初の行では、remotedev グループに属するすべてのユーザから、192.168.10.* のサブネットにあるワークステーションからプロキシを使わずに PERFORCE にアクセスしようとする際の list アクセスを拒否します。2 行目では、remotedev グループに属するすべてのユーザが 192.168.10.* サブネットから PERFORCE プロキシを使ってアクセスする場合に、write アクセスを保証します。リモート・サイトのワークステーションにいるユーザは、プロキシを使わなければなりません。

同様に 3 行目と 4 行目では、remotedev グループのユーザが中央オフィスのサブネット（10.0.0.0/8）上のワークステーションからプロキシを使おうとしたときに list アクセスを拒否し、中央オフィスのサブネット上のワークステーションから直接 PERFORCE サーバを使おうとしたときに write アクセスを保証します。ローカル・オフィスにいるとき、remotedev グループのユーザは PERFORCE サーバを直接アクセスしなければなりません。

特定のファイルがプロキシから提供されたのかどうかを判断する

-Zproxyverbose オプションをつけて p4 を実行すると、そのリビジョンがプロキシ（p4p）から転送されるのか、中央のサーバ（p4d）から転送されるのかを示すメッセージが表示されます。

例えば、:

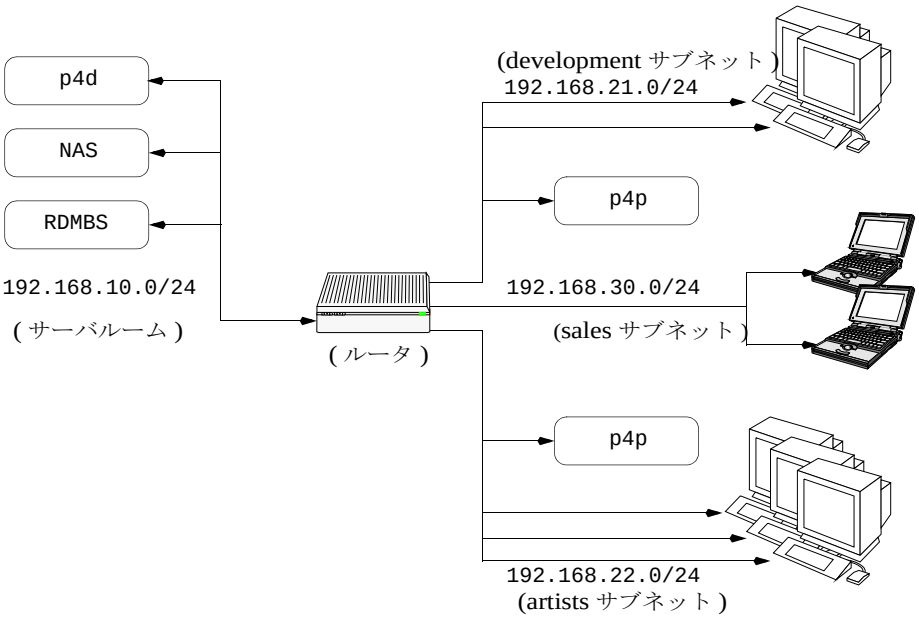
```
$ p4 -Zproxyverbose sync noncached.txt
//depot/main/noncached.txt - refreshing /home/p4adm/tmp/noncached.txt
$ p4 -Zproxyverbose sync cached.txt
//depot/main/cached.txt - refreshing /home/p4adm/tmp/cached.txt
File /home/p4adm/tmp/cached.txt delivered from proxy server
```

最大限にパフォーマンスを改善する

ネットワーク形態と P4P

中央の PERFORCE サーバと同じサブネットにあるネットワークのバンド幅がほぼ飽和状態にあるとき、同じサブネット内にプロキシを展開することは、パフォーマンスの改善にはなりません。その代わり、ルータをまたいだサブネットにプロキシを展開すれば、PERFORCE サーバがあるサブネットから切り離されますので、クライアントからプロキシへのトラフィックは独立します。

例えば、:



ネットワークのバンド幅がほぼ飽和状態のとき、そのサブネット上に追加のプロキシを展開することは、パフォーマンスの改善にはなりません。その代わりサブネットを複数に分割し、そのそれぞれにプロキシを展開します。

図に示した構成では、サーバルームに会社の PERFORCE サーバ、ネットワーク記憶装置（NAS）、データベース・サーバ（RDBMS）があります。営業部門はデータベースに対して絶えず日々の更新を行っており、開発部門とグラフィック設計部門は PERFORCE サーバに対して頻繁に大きなファイルのアクセスを行っているため、サーバルームのネットワーク・セグメントは飽和状態にあります。

2 つの PERFORCE プロキシを展開し、その 1 つは development サブネットに、もう一つは artists サブネットに置くことにします。これによって、サーバルームのネットワーク使用率が減り、3 つの部門すべてに対してよい結果が得られました。

最適化された初期パフォーマンスを得るためにキャッシュ・ディレクトリを事前取得する

P4P は、クライアントの 1 つから要求があった場合のみ、そのファイル・リビジョンを保持します。すなわち、ファイル・リビジョンを“事前取得”しているわけではありません。P4P によってパフォーマンスが改善できるのは、ファイル・リビジョンがキャッシュされた後に限ります。

P4P が起動した後、クライアント・ワークスペースを作成してディレクトリを同期することによって、効果的にキャッシュを事前取得することができます。続いてプロキシに接続したすべてのクライアントは、直ちに P4P によるパフォーマンス改善を得ることができます。

例えば、北米の PERFORCE サーバをターゲットとしている PERFORCE プロキシをもつアジアの開発サイトでは、北米サイトの開発作業が完了した頃を見計らって、ディポ全体に対して `p4 sync` を実行するような自動スクリプトを実行します。このタイミングはアジア・サイトで作業を開始する前になりますので、こうすることによってキャッシュ・ディレクトリを事前取得できることになります。

ディスクの消費を分散する

P4P は、ただ 1 つのディポ・ツリーが存在するかのようにリビジョンを保持します。もし、これによって 1 つのファイルシステムに格納されるファイルが多すぎるようになるのであれば、シンボリックリンクを使って複数のファイルシステムに分散させることもできます。

例えば、P4P のキャッシュ・ルートが `/disk1/proxy` にあり、PERFORCE サーバが 2 つのディポ `//depot` と `//released` を持つならば、次のようにして `//depot` を disk1 に、`//released` を disk2 に保持するようディスクを分散させることができます。:

```
mkdir /disk2/proxy/released
cd /disk1/proxy
ln -s /disk2/proxy/released released
```

この設定は、P4P が `//released` ディポのファイルを `/disk1/proxy/released` にキャッシュしようとしたとき、そのファイルが `/disk2/proxy/released` に格納されることを意味しています。

ファイルの圧縮を無効にしてサーバ CPU の使用を抑える

デフォルトで P4P は、PERFORCE との通信において圧縮を行いますので、サーバに対して余計なオーバヘッドを発生させることになります。

圧縮を無効にするには、`p4p` を実行するときに `-c` オプションを指定します。(サーバではなく) プロキシは、クライアントにファイルを転送する前にキャッシュのファイルを圧縮します。したがってこのオプションは、超過したネットワークとディスク容量を持ち、大量のバイナリファイルをディポに保持している場合に対して特に有効となります。

付録 A

PERFORCE サーバ（p4d）・リファレンス

概要

PERFORCEサーバの起動、およびチェックポイント/ジャーナル作成(システム管理)作業の実行。

シンタックス

```
p4d [ options ]
p4d.exe [ options ]
p4s.exe [ options ]
p4d -j [ -z ] [ args ... ]
```

説明

上記最初の 3 形式のコマンドは、PERFORCE のバックグラウンド・プロセス（“PERFORCE サーバ”）を起動します。4 番目の形式のコマンドは、システム管理作業に使用します。

UNIX および Mac OS X 上での実行ファイルは `p4d` です。

Windows 上での実行ファイルは `p4d.exe` （サーバとして動作）または `p4s.exe` （サービスとして動作）です。

完了コード

`p4d` は起動成功後、通常次の起動メッセージ、

```
PERFORCE server starting...
```

を出力し、バックグラウンドで動作します。

起動に失敗すると、`p4d` は 0 以外のエラー・コードを返します。

また、`-j` オプション（チェックポイント / ジャーナル作成）付きで起動した場合も、エラーが発生すると、`p4d` は 0 以外のエラー・コードを返して終了します。

オプション

	意味
`-d`	デーモンとして（バックグラウンドで）動作します。
`-f`	シングルスレッド（fork を行わない）プロセスとして動作します。
`-i`	UNIX 上の `inetd` から動作します。
`-q`	起動メッセージなど表示せずに動作します。
`-s`	NT サービスとして `p4d.exe` を実行します（p4s.exe の実行に相当）。
`-xi`	PERFORCE サーバ（およびそのメタデータ）を UNICODE モードで動作するように不可逆的に再構成します。UNICODE モードを必要とすることがわかっていない限り、このオプションは使用しないでください。詳しくは、*PERFORCE 国際語モードに関する注意事項（i18nnotes.txt）*をご覧ください。
`-xu`	データベースをアップグレードして終了します。
`-c command`	データベース・テーブルをロックしてコマンドを実行し、テーブルをアンロックして終了します。
`-jc [ prefix ]`	ジャーナル－作成（Journal-create）；チェックポイントを作成し、ジャーナルの保存 / トランケート（空にする）を行います。
`-jd [ file ]`	ジャーナル－チェックポイント（Journal-checkpoint）；チェックポイントの作成を行い、ジャーナルの保存 / トランケートは行いません。
`-jj [ prefix ]`	ジャーナルのみ（Journal-only）；ジャーナルの保存 / トランケートを行い、チェックポイントは作成しません。
`-jr file`	ジャーナル － 復元（Journal-restore）；チェックポイントやジャーナルからメタデータを復元します。
`-z`	チェックポイントおよびジャーナルを圧縮します（`gzip` フォーマット使用）。
`-h, -?`	ヘルプ・メッセージを表示します。
`-V`	サーバ・バージョンを表示します。
`-A auditlog`	監査ログ・ファイルを指定します。`P4AUDIT` の設定をオーバーライドします。デフォルト設定は null です。
`-J journal`	ジャーナル・ファイルを指定します。`P4JOURNAL` の設定をオーバーライドします。デフォルト設定は `journal` です。
`-L log`	ログ・ファイルを指定します。`P4LOG` の設定をオーバーライドします。デフォルト設定は stderr です。
`-p port`	接続待ちポートを指定します。`P4PORT` の設定をオーバーライドします。デフォルト設定は `1666` です。
`-r root`	サーバ・ルート・ディレクトリを指定します。`P4ROOT` の設定をオーバーライドします。デフォルト設定は現在の作業ディレクトリです。
`-v debuglevel`	サーバトレースオプションを設定します。`P4DEBUG` の設定値をオーバーライドします。デフォルト設定は null です。

使用上の注意

- すべてのプラットフォーム環境において、ジャーナル作成はデフォルトで有効になっています。サーバの起動時に `P4JOURNAL` が未設定の場合、ジャーナルを作成するデフォルトの場所は、`$P4ROOT/journal` になります。ジャーナル作成を手動で無効にするには、明示的に `P4JOURNAL` に `off` を設定しなければなりません。
- チェックポイントの作成とジャーナルのトランケートをひんぱんに行ってください。できれば毎日のバックアップ・プロセスの中に組み入れてください。
- チェックポイントとジャーナルによって保存されるのは、PERFORCE のメタデータ（保存されているファイル*に関する情報*）だけです。保存ファイルそのもの（すなわち、ソース・コー

ド）は P4ROOT にあり、やはり日常のバックアップの手順の一環としてバックアップしてください。

- ユーザがトリガを使用している場合は、-f （ノンフォーキング・モード）オプションを使用しないでください。トリガ・スクリプトを実行するには、PERFORCE サーバはコピーの生成（“fork”）を行わなければなりません。
- ハードウェアが故障した場合、チェックポイントとジャーナル・ファイルからのメタデータの復元に必要なオプションは、データが破損しているかどうかによって異なります。
- P4ROOT のファイルが消失したりした後にバックアップからデータ復旧を行う場合、しばしばジャーナル・ファイルが必要になります。したがって、ジャーナル・ファイルは、P4ROOT とは別のファイル・システムに配置することを強くお勧めします。そのようにしておけば、P4ROOTが存在するファイル・システムが破損しても、ジャーナルへのアクセスは可能でしょう。
- データベース・アップグレード・オプション（-xu）はかなりのディスク容量を要するかもしれません。2000.2 以前のサーバから 2001.1 以降のサーバへアップグレードする場合は、*リリースノート*および 18 ページの「リリース 2001.1 以降に関する重要な注意事項」をご覧ください。

関連コマンド

	`p4d -d -p 1999 -J /opt/p4d/journalfile`

デフォルト以外のジャーナル・ファイルを用いてサーバのチェックポイントを作成します。-J オプションの引数（または環境変数 P4JOURNAL）は、サーバ起動時に指定したジャーナル・ファイルと合致していなければなりません。	チェックポイント作成コマンド：
	`p4d -J /p4d/jfile -jc`
	または
	`P4JOURNAL=/p4d/jfile ; export P4JOURNAL`
	`p4d -jc`

ディレクトリ P4ROOT のファイルを使用し、サーバから圧縮チェックポイントを作成します。	`p4d -r $P4ROOT -z -jc`
ディレクトリ P4ROOT のファイルを使用し、サーバからユーザ指定のプレフィクス“ckp”を付けた圧縮チェックポイントを作成します。	`p4d -r $P4ROOT -z -jc ckp`
ルート・ディレクトリ P4ROOT のサーバのチェックポイント checkpoint.3 からメタデータを復元します。	`p4d -r $P4ROOT -jr checkpoint.3`
ルート・ディレクトリ P4ROOT のサーバの圧縮チェックポイント checkpoint.3.gz からメタデータを復元します。	`p4d -r $P4ROOT -z -jr checkpoint.3.gz`

索　引