Perforce Tutorial

Contents

1 Introduction to Perforce

Perforce Helix Core is a centralized version control system (VCS) that manages large codebases and digital assets. It is designed to handle complex development environments with speed and efficiency, making it a popular choice for large organizations and game development studios.

Key Benefits

- **Performance:** Optimized for large files and large numbers of files.
- **Security:** Granular access controls and audit trails.
- **Scalability:** Handles thousands of users and terabytes of data.

2 Installation and Setup

2.1 Download and Install Perforce

1. **Download the Perforce Command-Line Client (P4):**
   - Visit the Perforce Downloads page.
   - Choose the appropriate client for your operating system.
2. **Install the Client:**
   - Follow the installation instructions specific to your OS.

2.2 Set Up Environment Variables

- **P4PORT:** Specifies the address of the Perforce server.

```
export P4PORT=perforce:1666
```

- **P4USER:** Your Perforce username.

```
export P4USER=your_username
```

- **P4CLIENT:** Your workspace/client name.

```
export P4CLIENT=your_workspace
```

2.3 Authenticate with the Server

```
p4 login
```

3 Perforce Concepts in Depth

3.1 Depot

The **depot** is the central repository on the Perforce server where all versioned files are stored. It can contain multiple projects and branches.

3.2 Workspace (Client)

A **workspace** (also known as a client) is a local copy of files from the depot. It maps files in the depot to your local file system.

Creating a Workspace

```
p4 client
```

This command opens a text editor where you can define the workspace view and other settings.

3.3 Changelists

A **changelist** is a collection of file revisions that are submitted to the depot as a single unit. Each changelist has a unique number and a description.

Types of Changelists

- **Default Changelist:** The changelist where files are opened by default.
- **Numbered Changelist:** A user-created changelist with a specific number.

3.4 Revision

A **revision** is a specific version of a file in the depot. Revisions are numbered sequentially (e.g., `filename#1`, `filename#2`).

3.5 Integration

Integration in Perforce refers to the process of propagating changes between different branches. It encompasses both branching and merging.

4 Basic Workflow with Examples

Let's walk through a typical Perforce workflow with practical examples.

4.1 Setting Up Your Workspace

Example: Creating a new workspace named `alice_workspace`.

```
p4 client alice_workspace
```

In the editor that opens:

- **Client:** `alice_workspace`
- **Root:** `/Users/alice/projects/perforce`
- **View:**

  ```
  //depot/... //alice_workspace/...
  ```

Save and close the editor.

4.2 Syncing Files from the Depot

Example: Sync all files in the workspace to the latest revision.

```
p4 sync
```

Output:

```
//depot/project/file1.txt#3 - added as /Users/alice/projects/perforce/project/file1.txt
//depot/project/file2.txt#5 - updated /Users/alice/projects/perforce/project/file2.txt
```

4.3 Editing Files

Before editing a file, you need to open it for edit.

Example: Editing `file1.txt`.

```
p4 edit project/file1.txt
```

Output:

```
//depot/project/file1.txt#3 - opened for edit
```

Make your changes to `project/file1.txt` using your preferred text editor.

4.4 Adding New Files

Example: Adding a new file `file3.txt`.

1. Create the file locally.

   ```
   echo "New content" > project/file3.txt
   ```

2. Add the file to Perforce.

   ```
   p4 add project/file3.txt
   ```

   Output:

   ```
   //depot/project/file3.txt#1 - opened for add
   ```

4.5 Deleting Files

Example: Deleting `file2.txt`.

```
p4 delete project/file2.txt
```

Output:

```
//depot/project/file2.txt#5 - opened for delete
```

4.6 Reverting Changes

Example: Reverting changes to `file1.txt`.

```
p4 revert project/file1.txt
```

Output:

```
//depot/project/file1.txt#3 - was edit, reverted
```

4.7 Resolving Conflicts

Conflicts occur when multiple users edit the same file concurrently.

Example: Resolving a conflict in `file1.txt`.

1. Sync the latest changes.

   ```
   p4 sync
   ```

   Output:

   ```
   //depot/project/file1.txt#4 - merging //depot/project/file1.txt
   ```

2. Resolve the conflict.

   ```
   p4 resolve
   ```

   Options:

   - `e`: Launches an editor for manual merging.
   - `accept yours/ theirs/ merge/ edit`

4.8 Submitting Changes

Example: Submitting changes in the default changelist.

```
p4 submit
```

An editor opens to enter a changelist description.

- **Change:** `new`
- **Description:** `Fixed bug in file1.txt and added file3.txt`

Save and close the editor.

Output:

```
Change 102 submitted.
```

5 Branching and Merging

Branching allows you to diverge from the main line of development to work on features or fixes independently.

5.1 Creating a Branch

Example: Branching `//depot/project/...` to `//depot/branches/feature1/...`.

1. Use the `integrate` command.

```
p4 integrate //depot/project/... //depot/branches/feature1/...
```

Output:

```
//depot/branches/feature1/file1.txt#1 - branch from //depot/project/file1.txt#3
```

2. Submit the branch.

```
p4 submit -d "Created feature1 branch"
```

5.2 Making Changes in the Branch

Edit files in the branch as usual.

```
p4 edit //depot/branches/feature1/file1.txt
```

5.3 Merging Changes Back to Main

Example: Integrate changes from `feature1` back to `project`.

1. Integrate changes.

```
p4 integrate //depot/branches/feature1/... //depot/project/...
```

2. Resolve conflicts if any.

```
p4 resolve
```

3. Submit the merged changes.

```
p4 submit -d "Merged feature1 into main project"
```

6 Advanced Features

6.1 Shelving

Shelving allows you to store your changes on the server without committing them to the depot.

Example: Shelving changes in a changelist.

```
p4 shelve -c 103
```

6.2 Labeling

Labels are used to tag a group of file revisions for later reference.

Example: Creating a label for a release.

1. Create the label specification.

```
p4 label RELEASE_1.0
```

2. In the editor, define the label view and description.

3. Tag files with the label.

```
p4 labelsync -l RELEASE_1.0 //depot/project/...
```

6.3 Permissions and Protections

Manage user access with the `p4 protect` command (requires admin privileges).

7 Conclusion

By now, you should have a solid understanding of Perforce's basic and advanced features. Remember to practice these commands in a test environment to become more comfortable with Perforce operations.

Further Resources

- **Perforce Official Documentation:** https://www.perforce.com/manuals
- **Perforce Visual Client (P4V):** A GUI tool for interacting with Perforce.