GIT to PERFORCE

Here's a clear mapping between common **Git** and **Perforce (Helix Core)** activities so we can transfer our experience smoothly:

🔶 Repository / Depot Level

Git Term	Perforce Term	Description
Repository (`git init`, `git clone`)	Depot, Workspace (`p4 client`, `p4 sync`)	Perforce has a central **depot** (like Git's central repo). You set up a **workspace** (client spec) to map depot files to your local machine.

🔶 Checking out / Updating Files

Git	Perforce	Notes
`git pull`	`p4 sync`	Syncs latest changes from depot to your local workspace.
`git status`	`p4 opened` / `p4 diff`	Check what files are checked out or modified.
Edit file freely	`p4 edit <file>`	In Perforce, you must explicitly **check out** a file for editing.

🔶 Committing Changes

Git	Perforce	Notes
`git add` + `git commit`	`p4 add` / `p4 delete` + `p4 submit`	You add new files (`p4 add`), delete files (`p4 delete`), then **submit** changes.
`git commit -m "message"`	`p4 submit -d "message"`	Both attach a changelog message.
Amend commit	Modify changelist before submit	Once submitted, Perforce changelists are immutable (need to make a new changelist).

🔶 Branches / Streams

Git	Perforce	Notes
`git branch`	Streams / Branches in depot	Perforce uses **streams** (or classic branches) for parallel development.

Git	Perforce	Notes
`git checkout <branch>`	`p4 switch <stream>` (if using streams)	Switches workspace to a different stream or branch.
`git merge`	`p4 merge` + `p4 resolve` + `p4 submit`	Perforce separates merge, resolve (conflict resolution), and submit steps.
`git rebase`	Manual reorder / integrate work	Perforce has no direct equivalent; integration relies on merging + cherry-picking.

◆ Stashing / Shelving

Git	Perforce	Notes
`git stash`	`p4 shelve`	Temporarily store work without committing or submitting.
Apply stash	`p4 unshelve`	Bring shelved changes back to active changelist.

◆ Logs & History

Git	Perforce	Notes
`git log`	`p4 changes` / `p4 filelog`	Show changelists or file history.
`git show <commit>`	`p4 describe <changelist>`	Show details of a specific changelist.

◆ Tags / Labels

Git	Perforce	Notes
`git tag`	`p4 label`	Create a snapshot or marker on specific revisions.
`git checkout <tag>`	Sync workspace to label	Sync to the label revision using `p4 sync @label`.

◆ Undoing Changes

Git	Perforce	Notes
`git reset` / `git checkout`	`p4 revert`	Discard local changes (revert to last synced state).
`git revert`	Submit a new changelist that undoes	Explicitly submit a changelist that reverses prior changes.

🔶 Collaboration / Review

Git	Perforce	Notes
Pull requests, code reviews	Swarm (Perforce code review tool)	Perforce uses **Swarm** for reviewing changelists before submission.

⚠️ Key Differences

☑ Perforce enforces **file locking & explicit checkout** for edits (vs Git's optimistic model).
☑ Perforce changelists are **atomic**, but once submitted they can't be changed (no amend or rebase).
☑ Perforce scales better for **large binary assets** (common in game dev, hardware design).
☑ Git is **distributed**; Perforce is **centralized** — all history lives on the server.

📄 Perforce vs Git Command Cheatsheet

Action	Git Command	Perforce Command
Clone repo / setup	`git clone <url>`	`p4 client` + `p4 sync`
Check current branch / stream	`git branch` / `git status`	`p4 info` / `p4 streams` / `p4 client -o`
Update to latest	`git pull`	`p4 sync`
Check file status	`git status`	`p4 opened` / `p4 diff`
Edit a file	*(edit directly)*	`p4 edit <file>`
Add new file	`git add <file>`	`p4 add <file>`
Delete file	`git rm <file>`	`p4 delete <file>`
Commit locally	`git commit -m "message"`	*(no local commit, goes to changelist)*
Submit to server	`git push`	`p4 submit -d "message"`
Create new branch	`git branch <name>`	Create new stream or branch mapping
Switch branch	`git checkout <branch>`	`p4 switch <stream>` (if streams used)
Merge branches	`git merge <branch>`	`p4 integrate` + `p4 resolve` + `p4 submit`
Stash changes	`git stash`	`p4 shelve`
Apply stashed/shelved changes	`git stash apply`	`p4 unshelve`
View history	`git log`	`p4 changes` / `p4 filelog`

Action	Git Command	Perforce Command
See specific changelist details	`git show <commit>`	`p4 describe <changelist>`
Revert local changes	`git reset --hard` / `git checkout -- <file>`	`p4 revert <file>`
Label/tag a revision	`git tag <tagname>`	`p4 label` + `p4 labelsync`
Checkout by tag/label	`git checkout <tag>`	`p4 sync @label`

⚡ Quick Notes

☑ **No local commits in Perforce** — everything works through changelists (pending or submitted).
☑ **Explicit file operations** — you must `p4 edit` or `p4 add` before changing or adding files.
☑ **Conflicts resolved manually** — use `p4 resolve` after merges or integrates.
☑ **Shelving ≠ stashing** — but similar; shelved work is visible to others if needed.

🌟 Basic Perforce Development Workflow

Step 1 → Set up your workspace (like `git clone`)

☑ **Perforce** is centralized, so you don't "clone" the whole depot like Git.
Instead, you set up a **client workspace** that maps depot files to a local directory.

- `p4 client` → create a workspace (define name, root folder, depot mappings)
- `p4 sync` → pull the latest files from the depot into your local workspace

👉 **Analogy**: Like `git clone` but only for the parts of the depot your workspace maps.

Step 2 → Edit files (like `git checkout -b` + edit)

☑ Perforce requires **explicit checkout**:

- `p4 edit <file>` → marks a file for edit (unlocks it, tracks it)
- Or, `p4 add <file>` → for new files
- Or, `p4 delete <file>` → for deletes

👉 **Git difference**: In Git, you can just start editing or adding files freely.

Step 3 → Work on your feature or fix

☑ Make your code changes locally in your workspace.
Perforce tracks modified files in a **pending changelist**.

- Use `p4 opened` → to check what files you have open for edit.
- Optionally, create a new changelist (like a task-specific staging area).

👉 **Analogy**: Like building up your local commits in Git, except there's no local commit — you're preparing files for submission.

Step 4 → Update if others have submitted (like `git pull` / `git rebase`)

☑ Before submitting, **sync** the latest changes:

- `p4 sync` → updates your workspace to the latest depot version.
- If there are conflicts:
  - `p4 resolve` → interactively resolve merge conflicts.

👉 **Analogy**:

- Git: `git pull --rebase` → integrate latest upstream changes.
- Perforce: `p4 sync` + `p4 resolve` before submit.

Step 5 → Submit your changes (like `git commit` + `git push`)

☑ When ready, **submit** your changelist:

- `p4 submit -c <changelist#>` → sends your edits to the server.
- Provide a descriptive changelist description (like a commit message).

👉 **Git difference**: Perforce changelists are atomic submits to the central server (no local commit history).

🌐 Example full cycle:

1 Setup

```
p4 client       # define workspace
p4 sync         # pull latest depot files
```

2 Start work

```
p4 edit file.cpp   # check out file for editing
# make changes in editor
```

3 Check status

```
p4 opened        # see what's open
p4 diff          # view local diffs
```

4 Update from others

```
p4 sync          # get latest from depot
p4 resolve       # handle conflicts if needed
```

5 Submit

```
p4 submit -d "Fixed bug in feature X"
```

⚡ If you need to merge changes from another branch/stream:

- `p4 integrate <from> <to>` → like `git merge` (pulls changes between branches/streams)
- `p4 resolve` → manually handle any conflicts
- `p4 submit` → finalize and apply the merged changes

🔑 Main differences from Git to remember:

☑ You work **directly on the central server** — no local commits.
☑ You need to **check out files explicitly** before editing.
☑ Changelists are **atomic submissions** (can't be modified after submission).
☑ Perforce streams handle branching/merging, but are structured and permissioned more tightly than Git.