Perforce Best Practices for Codeline Management

Organizing Codelines, Part 1

Your objectives for this exercise:
- Review the terminology used in defining codelines.
- Evaluate how codelines differ in their relative stability.
- Practice thinking in terms of the mainline model.

Suggested answers are on pages following these questions. Refer to Chapter 7 in Practical Perforce for assistance in answering these questions.

1. When we define a codeline, what is the primary feature all the files in the codeline have in common?
   
   The files in a codeline evolve together in the same phase of development.

2. Name a couple features of the mainline model of software development
   
   Release and development codelines are branched from the mainline.
   Normally changes flow back to the mainline.
   The mainline is persistent as long as the project continues to evolve.

3. List some advantages of creating a release codeline.
   
   Bug fixes can be isolated from the mainline.
   The mainline can continue to evolve independently.
   Fixes can be tested before merging the changes to the mainline.

4. List some advantages of creating a development codeline.
   
   New features that may destabilize the mainline can be developed.
   New features that may not be included in an upcoming release can be developed.
   Code can be shared among developers that would not normally be included in the mainline.
   Testing can be isolated from the mainline.

5. Is there a mainline in our promotion model for website development discussed in class?
   
   No, we do not merge files back to the parent codeline.
   Changes are continually “promoted” or copied to another codeline.

6. We can determine the lineage of files using Perforce commands such as p4 filelog and p4 integrated to view the integration history of a file or codeline. Is there a way Perforce can tell us if a codeline is more stable than another?

   No, codeline stability is an idea we must communicate with our fellow developers. We will see in the next section how we can name codelines to indicate their stability.
7. List these codelines in their “tofu” ranking from firm to soft:

   Codeline “A” that has all of the initial files required for a project checked in.
   Codeline “B” is branched from “A” and is going to be used to develop a new database driver.
   Codeline “C” is branched from “A” and is undergoing final QA testing.

   The ”Tofu” ranking of the codelines is:
   “C” is going to be a release line and is the most stable.
   “A” is the mainline.
   “B” is the least stable, created for new feature development.

8. Suggest which direction changes must flow, maintaining the mainline model, before a new release codeline can be branched. The new release is going to include the new feature “SmartKey” being developed.

   The mainline has been branched to a development codeline for “SmartKey.”
   Several bug fixes have been submitted to an earlier release codeline.

   The bug fixes from the earlier release should flow to the mainline.
   These fixes should flow to the “SmartKey” codeline.
   The completed and tested “SmartKey” codeline should then flow to main.
   Now all the changes are in the mainline and the release codeline can be branched.
Organizing Codelines, Part 2

In the scenario for this set of exercises, you have been assigned the task of reviewing your company’s current organization of codelines and making suggestions for implementing best practices moving forward. You may use the example, ACE engineering, outlined below or use data from your company’s installation of Perforce.

Your objectives for this exercise:

- Follow best practices when naming depots, codelines and modules.
- Specify client workspace templates for your developers.
- Define the role of a codeline curator.

ACE Engineering is a startup company. Your development manager has requested that you follow the mainline model and set up a naming convention for your Perforce server for a main development line and release branches. ACE Engineering’s first product is called Clubs. The first release will be named Clubs 1.0. Besides the development and release branches, a website will be developed for the new product. Your task is to configure development and live server codelines for your website. The Clubs product will initially run on Linux and Windows platforms. Both Linux and Windows development will utilize some shared code.

1. Define a depot name for your product.
   
   //Clubs/... Ace Engineering Clubs Product

2. Define folder names for grouping your codelines.
   
   //Clubs/Prod/... Development and Release codelines
   //Clubs/Web/... Website development codelines

3. Define a folder name for each CODELINE.
   
   //Clubs/Prod/MAIN/... Main codeline
   //Clubs/Prod/DEV/... Development codeline
   //Clubs/Prod/REL1.0/... Release codeline
   //Clubs/Web/MAIN/... Main website development codeline
   //Clubs/Web/LIVE/... Live website codeline

4. Define lowercase names for modules to be included with each codeline.
   
   //Clubs/Prod/MAIN/linux/... Linux specific modules
   //Clubs/Prod/MAIN/windows/... Windows specific modules
   //Clubs/Prod/MAIN/shared/... Shared modules
   //Clubs/Prod/DEV/linux/... Development modules, etc.
   //Clubs/Prod/DEV/windows/
   //Clubs/Prod/DEV/shared/
   //Clubs/Prod/REL1.0/linux/... Release modules, etc.
   //Clubs/Prod/REL1.0/windows/
   //Clubs/Prod/REL1.0/shared/...
5. **Define a View:** field for a client workspace, “ClubsDEV,” that can be used as a template for working on the main development codeline.

   **Client:** ClubsDEV  
   **View:**  
   //Clubs/Prod/DEV/... //ClubsDEV/DEV/...

6. **Define a View:** field for a client workspace, “ClubsREL1.0” that can be used as a template for release, “REL1.0,” codeline developers.

   **Client:** ClubsREL1.0  
   **View:**  
   //Clubs/Prod/REL1.0/... //ClubsREL1.0/REL1.0/...

7. **Define a View:** field for a client workspace, “ClubsWeb,” that can be used as a template for website developers.

   **Client:** ClubsWeb  
   **View:**  
   //Clubs/Web/MAIN/... //ClubsWeb/MAIN/...

8. **List potential responsibilities of a codeline curator for the codelines you have defined.**

   A codeline curator ensures the codeline operations are performed smoothly. Some examples of what a curator might do are:

   - Set naming conventions for codelines.
   - Maintain a group of users who have write access to files.
   - Maintain a group of users who have read access to files.
   - Implement policy statements regarding best practices for codeline usage.
   - Create README files at each codeline root elaborating on the policy statements mentioned above.
   - Monitor integration of files to and from other codelines.
   - Review and implement triggers.
Parallel Development, Part 1

**Your objectives for this exercise:**
- Select the correct accepting action in the resolve dialog to achieve the desired outcome.
- Define requirements for developers working on fully branched codelines.
- Define requirements for developers working on task branches.

In exercises 1-4, specify which `p4 resolve` accepting action would give the correct result when integrating files between two codelines. In our examples we'll assume the `p4 integrate` command has been executed and we are now in the `p4 resolve` dialog.

1. A developer has submitted a change to a web site development branch. The change needs to be copied to a review codeline.
   
   ```
   p4 resolve -at
   "accept theirs" will copy the source file to the target.
   ```

2. A developer wants to merge files that have unintegrated changes in both the source and target codelines.
   
   ```
   p4 resolve -am
   "accept merged" will merge the changes in the source and target files if there are no conflicts.
   ```

3. A developer decides that a change made in a development codeline should not be considered in future integrations to the mainline.
   
   ```
   p4 resolve -ay
   "accept yours" will "ignore" the source and keep the target.
   ```

4. While in the resolve dialog a developer decides the merged file must be edited before being submitted.
   
   ```
   p4 resolve -ae
   "accept edited" will keep the edited file.
   ```

5. What could be an unintended consequence of an "accept theirs" resolve?
   
   *Any unintegrated edits on the target codeline would be lost.*

6. Create a list of requirements for developers working in fully branched codelines.
   
   *Communication is the key.*
   
   *Developers should keep their workspaces current.*
   
   *Submits should always include files that are logical units of work.*
   
   *Changelist descriptions should explicitly list what changes are represented by the changelist.*
   
   *Each changelist should be integrated individually sooner rather than later to appropriate codelines.*
   
   *Verify merged file results locally before submitting.*
   
   *Utilize collaboration tools to communicate with other members of your team.*
7. Create a policy statement for when developers may create a sparse branch.

For example:
Sparse branches may be created if development:
- is by one developer only.
- is short lived.
- involves just a few files.
Parallel Development, Part 2

Your objectives for this exercise:
- Understand the mechanics of using sparse branches.
- Handle virtual modules.
- Know who has permission to edit submitted changelists for collaboration.

1. What is the effect of using the overlay mapping in a workspace view for a sparse branch?

   When we do a `p4 sync` the files in the overlay folder will replace the files in the parent branch so all the file from the parent branch plus the modified files in the sparse branch are available for testing and/or building.

2. What is the advantage of using a branch specification when integrating files to a sparse branch?

   We use a branch specification for performance considerations. When we can limit the number of Perforce commands run to perform an operation our server will respond faster for all users.

3. If we use our workspace with a overlay mapping when integrating back to the parent, why do we have to explicitly sync the target path before integrating?

   If we don’t include the path the overlay mapping will take precedence and the target files will not be in our workspace. This will result in an error when we integrate, since we always resolve against the target files in our workspace.

4. What is an advantage of using "virtual" modules?

   We consider using virtual modules for performance reasons. By integrating less files our Perforce database is smaller, thus allowing our commands to run faster.

5. Your website manager has requested that work on the web site development branch be integrated to a “review” branch before being integrated to the “live” branch. Create a branch specification and map files from the website development branch to the review branch.

   For example you might define your branch thus:

   Branch: ClubsWebMAINtoREVIEW
   View:  
   //Clubs/Web/MAIN/... //Clubs/Web/REVIEW/...
6. The development lead for the windows modules wants to add dynamically linked library files (*.dll files) for Windows development to the Clubs project MAIN branch. The files are going to be virtual files for the developers working on the DEV branch. Define a folder name for the Windows *.dll files.

//Clubs/Prod/MAIN/windll/...

7. Add to the View: field in your client workspace template, “ClubsDEV” mappings for your virtual modules.

Client: ClubsDEV
View:
   //Clubs/Prod/DEV/...  //ClubsDEV/DEV/...  
   //Clubs/Prod/MAIN/windll/... //ClubsDEV/DEV/windll/...

8. Which Perforce users are allowed to edit a submitted changelist?

The owner of a changelist or a user granted admin privilege in the protections table can edit a submitted changelist.
Codeline Operations, Part 1

For this set of exercises, use the codelines as described in the class slides.

Your objectives for this exercise:
- Follow best practices when propagating a rename across several branches.
- Follow best practices when performing a copy from one branch to another.
- Use Perforce to enforce a code "freeze."
- Determining codeline divergence and ensuring convergence.

1. Based on the following Revision Graph, suggest a proper sequence of p4 commands to propagate the renamed file //Sim/Prod/REL2.2/src/fileos2-nt.c to the //Sim/Prod/FLAT/src folder. The "tofu" ranking of these codelines from firm to soft is: REL2.2 > MAIN > FLAT.

Since the REL2.2 codeline is the firmest, we can merge "down."

p4 integ //Sim/Prod/REL2.2/src/fileos2-nt.c //Sim/Prod/MAIN/src/fileos2-nt.c
p4 delete //Sim/Prod/MAIN/src/fileos2.c
p4 submit
p4 integ //Sim/Prod/REL2.2/src/fileos2-nt.c //Sim/Prod/MAIN/src/fileos2-nt.c
p4 resolve -am
p4 submit
p4 integ //Sim/Prod/FLAT/src/fileos2-nt.c //Sim/Prod/FLAT/src/fileos2-nt.c
p4 delete //Sim/Prod/FLAT/src/fileos2.c
p4 submit
p4 integ //Sim/Prod/MAIN/src/fileos2-nt.c //Sim/Prod/FLAT/src/fileos2-nt.c
p4 resolve -am
p4 submit

The Revision Graph result below shows the propagated rename:
2. Suggest a proper sequence of p4 commands to copy finished work from
//Sim/Prod/FLAT/src/... to //Sim/Prod/MAIN/src/... FLAT is softer than MAIN on the “tofu”
scale.

   Find out if any merge “downs” are pending and integrate as necessary:
   p4 integ -n //Sim/Prod/MAIN/src/... //Sim/Prod/FLAT/src/...
p4 integ //Sim/Prod/MAIN/src/... //Sim/Prod/FLAT/src/...
p4 resolve -am
p4 submit

   Copy the source files to the target:
   p4 integ //Sim/Prod/FLAT/src/... //Sim/Prod/MAIN/src/...
p4 resolve -at
p4 submit

   Integrate any files that are identical that have not been integrated:
   p4 diff2 -q //Sim/Prod/FLAT/src/... //Sim/Prod/MAIN/src/...
   Use the list of files from the previous command to force the integration for each file, e.g.:
   p4 integ -f //Sim/Prod/FLAT/src/command.c //Sim/Prod/MAIN/src/command.c ...etc.
p4 resolve -at
p4 submit

   For small databases an alternative recipe can be found on our Perforce website:
Convergence vs. Divergence: Purposeful Merging with Perforce by Laura Wingerd,
Perforce Software, Perforce European User Conference 2006

3. How would you enforce a code freeze using Perforce before you copy “up” to the
//Sim/Prod/MAIN codeline?

   Grant open privilege to MAIN and grant write privilege to the user who will perform the integration.

   p4 protect

   Protections:
   open group SimMAINdevelopers * //Sim/Prod/MAIN/...
   write user bruno * //Sim/Prod/MAIN/...

4. How can you determine if codelines MAIN and REL2.2 are divergent, even if all of the files
have been integrated?

   If our codeline policy is such that we have a branch spec for each branched codeline then we can
easily find out if codelines differ:
   p4 diff2 -q -b SimMAINtoREL2.2

5. What additional step must be performed to ensure codeline convergence beyond the “merge
down”, “copy up” policy?

   You must copy files from source to target that have full integration credit yet are not identical.
Codeline Operations, Part 2

Your objectives for this exercise:
- Investigate before using the “-d” flag when integrating between codelines.
- Use Perforce to ensure proper handling of files that are not intended to be integrated to specific codelines.
- Back out an old changelist
- Think about scripting opportunities for your organization.

1. What is the effect of using the “-d” flag with the p4 integrate command.

   *We are forcing the target to be convergent with the source, so files will be either added or deleted depending upon if the files on the source path are either edited or deleted, respectively.*

   *If your intention is to maintain codeline divergence, then we would not use the “-d” flag.*

2. Let’s say we have a file, macX.c, that has been added to the //Sim/Prod/REL2.2/src release codeline module to fix a customer-reported bug that you do not want to propagate to the //Sim/Prod/MAIN/src codeline. Suggest a method using Perforce to ensure that the new file name does not get copied to the main codeline in the future.

   *Edit the branch specification for the release branch and add an exclusionary mapping for the file:*
   
   `p4 branch SimMAINtoREL2.2`
   
   *View:*
   
   `//Sim/Prod/MAIN/src/... //Sim/Prod/REL2.2/src/...`
   `-//Sim/Prod/MAIN/src/macX.c //Sim/Prod/REL2.2/src/macX.c`

3. How would you back out edits to two files sim.c and sim.h that have been submitted some time ago? Let’s say our bad edits are in changelist 1214.

   `p4 sync sim.c@1213 sim.h@1213`
   `p4 edit sim.c sim.h`
   `p4 sync sim.c@1214 sim.h@1214`
   
   *Here we are telling Perforce what the “theirs” file is for the integration.*
   `p4 resolve-ay`
   `p4 sync sim.c sim.h`
   `p4 resolve`
   
   *Now we merge the latest edits into our files.*
   `p4 submit`
4. What operations we have discussed do you think would be good candidates to be scripted?

5. What operations are currently scripted by your company?

Congratulations! You have completed the Perforce Training Course. We hope this course material has prepared you to use Perforce with confidence and ease.