May 21, 2020

8 Version Control Best Practices

VERSION CONTROL | CODING BEST PRACTICES

By Brent Schiestl

Every team should follow version control best practices — whether your team is big or small.

Here we cover key version control best practices and how to apply them.

8 Version Control Best Practices

Here are 8 of the most critical version control best practices.

Commit Changes Atomically

One best practice is to commit changes atomically in version control.

All files in a commit are either committed together or not at all. No other user should see partial or incomplete changes.

A check-in is similar to a database transaction described by its ACID properties:

- Atomic.
- Consistent.
- Isolated.
- Durable.

Commit all files that belong to a task in a single operation to keep the project consistent at all times.

It's critical to apply best practices to commits. Good-quality commits will improve your project, making you more productive and successful.

Commit Files With a Single Purpose — Not as a Backup

Another best practice is committing files with a single purpose.

Each commit should have a single purpose. For example, fixing a bug or adding a new feature. If a single change makes multiple independent changes to your project, it can become difficult to read and to review. Backing out one of these changes then becomes more complex and unnecessarily time-consuming.

> Remember: A commit is not a backup of your current state of your local files, even if it occurs at the end of the day.

By breaking down a larger task into smaller chunks, you can more readily understand and review the intent of changes. For example, you could break a task into infrastructure and refactoring tasks before making user-visible changes. Keeping the scope narrow also makes it easier to back out a bad commit.

Write Good Commit Messages

Another commit best practice is to write good commit messages.

Each commit should have a description that explains the why — but not necessarily the how — regarding the change. (How is usually deducible by comparing the file contents before and after the change.)

A good commit message makes it easier for a reviewer — and you — to understand the purpose of the commit later. A good commit message also references the issue ID(s) — or even the requirement ID(s) — that the commit addressed (if applicable).

Don't Break Builds

Another version control best practice is to avoid breaking builds by doing complete commits.

Provide test cases and at least stubs for new APIs. This ensures every commit is usable by any other member in the team without breaking their build.

A complete commit is easier to propagate between branches. An incomplete commit of an API, for example, might build locally in your work area and pass all tests. But it could break in another team member's work area.

Do Reviews Before Committing to a Shared Repository

It's also a best practice for version control to do reviews before committing to a shared repository.

A good commit is often reviewed before merging it to a shared repository. This is done either through a review system or a pull-request.

Reviews are a great way to get another perspective on a change and to improve code quality. Code reviews are also useful to increase code awareness within the team. This also enhances the team's productivity through code reuse and higher quality of output.

More on code review best practices >> (https://www.perforce.com/blog/qac/9-best-practices-for-code-review)

Make Sure Every Commit Is Traceable

Another best practice for version control is to ensure traceability.

The project should be able to build and pass its test cases before and after the commit. If you notice a bug and want to track down the change that introduced the bug, you usually reset your working environment to a previous time to verify the bug is still there. (This is done either by hand or through some bisect facility.) If previous changes don't even build, tracing down a bug becomes a lot more difficult.

For security and auditing, you must store the author of the change. You also need to store additional information, such as reviewer comments. A commit is also often associated with a specific issue or new feature request.

Following the version control best practices highlighted here will ensure that each commit can be backed out again if necessary. The best pre-commit reviews and build

tests won't always prevent unintended side effects that appear in later testing.

In such cases, it might be necessary to back out a commit. This returns the state of the project to an earlier time. This operation usually preserves history as well, so that the change can later be re-applied or analyzed and fixed as necessary.

Follow Branching Best Practices

It's also important in version control to follow branching best practices. So, what is the best practice for branching?

There are many.

Using branches (https://www.perforce.com/resources/vcs/version-control-branching) is important for managing releases, new features and bugs. But there can be some challenges in branching. For instance, changes in one branch often have to flow to other branches. This makes it critical to follow branching best practices to avoid merge conflicts, lost updates, and unintentional overwriting of existing changes.

Branching best practices include:

- Try to keep things simple.
- Have well-defined code branching policies.
- Give codelines an owner.
- Uses branches for releases or milestones.
- Protect your mainline.
- Merge down and copy up.

More on branching best practices >> (https://www.perforce.com/blog/vcs/branching-definition-what-branch)

Protect Your Assets

Another version control best practice is to incorporate the right security measures to protect your assets.

Your version control system (https://www.perforce.com/blog/vcs/what-is-version-control) is a key repository for your organization. It stores and manages some of the most valuable assets in the company:

- Your intellectual property (IP), which includes source code for applications used internally and/or by your customers.

- Your product designs.
- Export or compliance documentation.
- Your videos, graphics, or images.
- Business documents.
- And much more.

Consider the value of these assets. And consider time and effort needed to recreate them after any potential disaster or the possible risk if they were leaked to a competitor. Then you'll get some idea of why security should be a major consideration when choosing a version control tool, so you can always apply the best practices.

Best practices include:

- Backup and failover.
- Access control.
- Visibility into activity.

Related Content:

- How to Streamline IP Reuse (https://www.perforce.com/blog/vcs/how-streamline-ip-reuse)
- What Is MFA? (https://www.perforce.com/blog/vcs/what-is-multi-factor-authentication)
- How to Lock Down Git (https://www.perforce.com/resources/vcs/how-lock-down-git)
- SVN Branching & Merging (https://www.perforce.com/blog/vcs/svn-branching-and-merging)
- 7 DevOps Best Practices (https://www.perforce.com/blog/vcs/7-devops-practices-outstanding-results)

Version Control Checklist

Here's a quick version control checklist to use to ensure you're applying the right version control best practices.

Commits

Applying version control best practices to commits is critical. Here's what you need to consider.

- Have all commits be atomic, complete, consistent, traceable and with a single intent

- Make changes visible through frequent commits
- Consider how you would use the comments in the future
- Review code before committing to the mainline
- Make commits reversible

Branching

Applying branching best practices is critical to success. But it can be complicated. To reduce the pain (and effort) for your teams, your branching strategy should aim to:

- Optimize productivity.
- Enable parallel development.
- Allow for a set of planned, structured releases.
- Provide a clear promotion path for software changes through production.
- Evolve to accommodate changes that are delivered, perhaps daily.
- Support multiple versions of released software and patches.

Security

Security is another critical version control best practice. Your security plan must consider multiple levels.

- Data: encryption at rest and in transit; specific file and file-type access controls.
- Users: authentication and authorization; integration with enterprise tools.
- Branches and streams: partitioning access control according to the intent of a change— development or release.
- Audit trails: immutable history of all changes.
- Threat detection: using data collected to warn of accidental or malicious risks.

Apply Version Control Best Practices With Helix Core

Helix Core (https://www.perforce.com/products/helix-core) — version control from Perforce — makes it easy to apply version control best practices.

You can use Helix Core to:

- Commit changes atomically.
- Commit files with a single purpose.
- Write good commit messages.
- Avoid broken builds.

- Do reviews before committing to a shared repository.
- Ensure complete traceability.
- Enforce branching best practices with Perforce Streams (https://www.perforce.com/solutions/version-control/branching-brains).
- Incorporate security measures all the way down to the individual file-level.

See for yourself why Helix Core is the best version control tool. You can get started for free for up to 5 users and 20 workspaces.

APPLY BEST PRACTICES WITH HELIX CORE (HTTPS://WWW.PERFORCE.COM/PRODUCTS/HELIX-CORE/FREE-VERSION-CONTROL)

PERFORCE (/)

SEND FEEDBACK