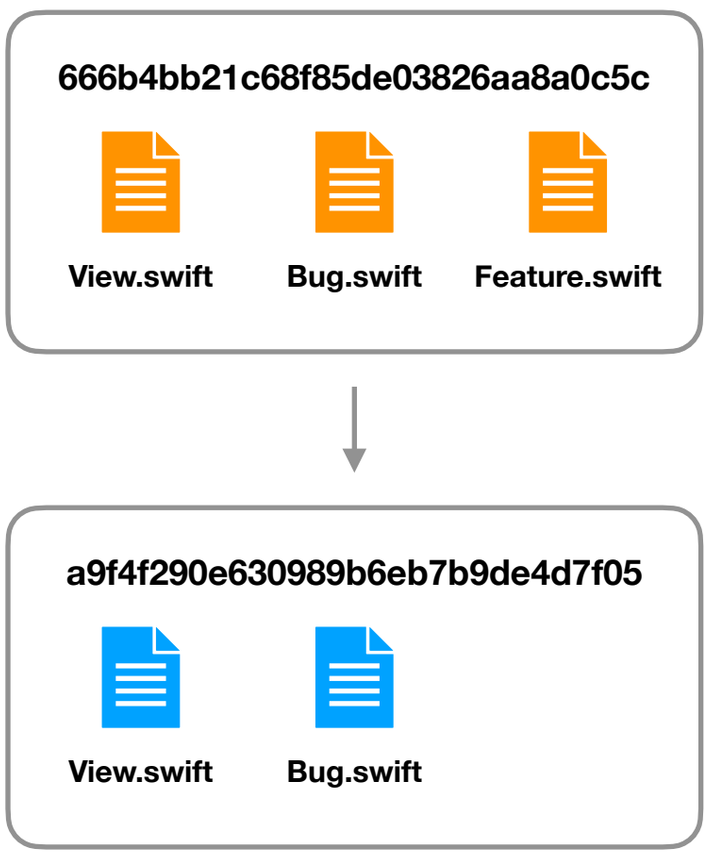
Atomic Commits

The Git best practice that improves all workflows

“Any software project is a collaborative project. It has at least two developers, the original developer and the original developer a few weeks or months later when the train of thought has long left the station.”

–Peter Hutterer

Committing unrelated changes



Problems

- It’s hard to review the changes together
- It’s hard to find new bugs or the reason of a regression
- Reverting undesired changes will also revert desired ones
- It’s impossible to cherry pick a change
- There will be more merge conflicts

Best practices for atomic commits

- Commit only single, complete units of work
- Break large features into smaller units
- Commit early and often
- Don't commit half-done work
- Test your code before you commit
- There is no such thing as too many commits
- Craft a well-written commit message

Crafting a well-written commit message

```
Summarize changes in around 50 characters or less

More detailed explanatory text, if necessary. Wrap it to about 72
characters or so. In some contexts, the first line is treated as the
subject of the commit and the rest of the text as the body. The
blank line separating the summary from the body is critical (unless
you omit the body entirely); various tools like `log`, `shortlog`
and `rebase` can get confused if you run the two together.

Explain the problem that this commit is solving. Focus on why you
are making this change as opposed to how (the code explains that).
Are there side effects or other unintuitive consequences of this
change? Here's the place to explain them.

Further paragraphs come after blank lines.

 - Bullet points are okay, too

 - Typically a hyphen or asterisk is used for the bullet, preceded
   by a single space, with blank lines in between, but conventions
   vary here
```

```
If you use an issue tracker, put references to them at the bottom,
like this:

Resolves: #123
See also: #456, #789
```

Subject line

- Capitalize
- Do not end with a period
- Use the imperative tense
- Limit to 50 characters

Body

- Separate with a blank line
- Wrap at 72 characters
- Keep the message in the present tense
- Divide the text into paragraphs and bullets
- Explain *what* and *why*, not *how*

Issue tracker references

Conventional commit types

- Adding a new feature to the product
- Changing an existing feature
- Removing a feature from the product
- Fixing a bug
- Refactoring code
- Deprecating an existing API
- Performance improvement
- Security
- Stylistic changes
- Adding tests or fixing existing ones
- Adding or updating documentation
- Configuring the build system
- Operational infrastructure
- Chores
- Reverting a previous commit