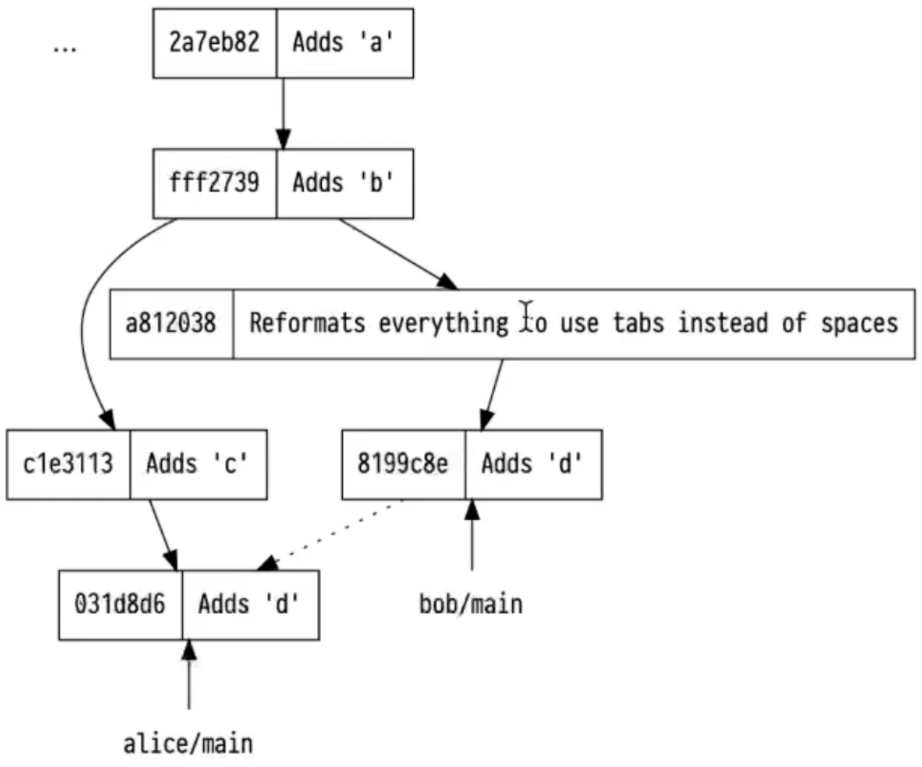
Git: cherry-pick & ignore

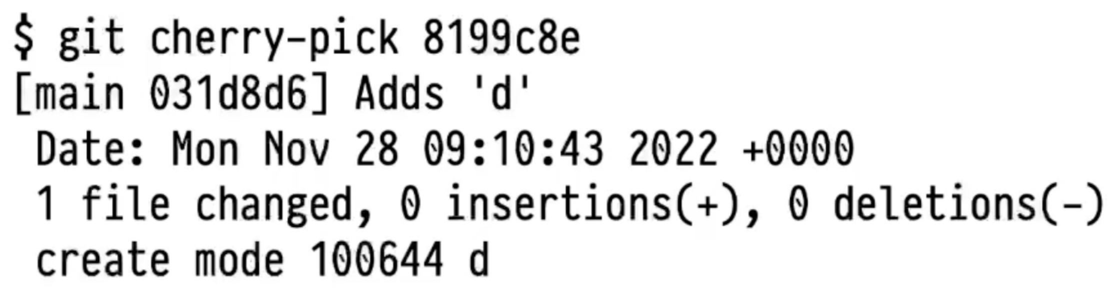
- say bob has done some interesting work on *their* main branch, but also some less interesting work...
  - theyve fixed some bugs but theyve also switched all your files from using spaces to tabs

How do you cherry pick the things you want and ignore then things you dont want?

git cherry-pick



- in this example Alice wants the commit 8199c8e but not a812038
- to get only that change you can use the cherry-pick command



```
$ git cherry-pick 8199c8e
[main 031d8d6] Adds 'd'
 Date: Mon Nov 28 09:10:43 2022 +0000
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 d
```

  - this lets you take individual commits from someone elses branch, and bring it over to your branch
- if that change relies on other files as well, git will try and be clever and pull those changes as well
- if you merge later, then git will be clever and will know where things came from and will not cause a merge conflict

Basic Cherry-Pick

As previously mentioned, the basic form of `git cherry-pick` involves applying a single commit from one branch onto the current branch. Here's the command:

```
shCopy code
git cherry-pick <commit-hash>
```

Cherry-Picking Multiple Commits

You can cherry-pick a range of commits. This is useful when you want to apply several specific commits in sequence.

Syntax:

```
shCopy code
git cherry-pick <start-commit-hash>^..<end-commit-hash>
```

Note: The caret symbol (`^`) next to the start commit hash is used to include the start commit in the cherry-picked range.

Example:**

Suppose you have a series of commits `A -> B -> C -> D` and you want to cherry-pick commits `B` and `C` onto your current branch. You can do it like this:

```
shCopy code
git cherry-pick B^..C
```

Cherry-Picking from Another Branch

If you know the branch where the desired commit resides, but you're not currently on that branch, you can still cherry-pick directly using the commit hash.

1. Ensure you're on the branch you want to apply the commit to:

    ```
    shCopy code
    git checkout my-current-branch
    ```

2. Then cherry-pick the commit:

    ```
    shCopy code
    git cherry-pick <commit-hash-from-other-branch>
    ```

Solving Conflicts during Cherry-Pick

Sometimes, cherry-picking a commit can result in merge conflicts. Git will pause the cherry-pick operation, allowing you to resolve the conflicts manually.

1. Edit the files to resolve conflicts.
2. Add the files after resolving conflicts:

   ```
   shCopy code
   git add .
   ```

3. Continue the cherry-pick process:

   ```
   shCopy code
   git cherry-pick --continue
   ```

Or, if you decide not to proceed with the cherry-pick, you can abort the operation:

```
shCopy code
git cherry-pick --abort
```

Cherry-Picking a Merge Commit

Cherry-picking a merge commit is a bit different because a merge commit has two parent commits. You need to decide which parent's changes you want to bring into the cherry-pick. Use the `-m` option to specify the parent.

Syntax:

```
shCopy code
git cherry-pick -m 1 <merge-commit-hash>
```

Here, `-m 1` indicates the first parent of the merge commit. You might change the number if you want to follow a different parent.

Interactive Cherry-Picking

For a more controlled cherry-picking process, especially when dealing with multiple commits, you can use the interactive rebase feature, though this isn't directly a cherry-pick command. This is useful for editing, rearranging, or modifying commits before applying them.

Example:

```
shCopy code
git rebase -i <commit-hash>^
```

This starts an interactive rebase session that allows you to pick, edit, squash, or fix up commits as needed. It's a more advanced feature but offers greater control over the commits you're manipulating.

.gitignore

`.gitignore` is a file in your Git repository that tells Git which files or directories to ignore in your project. It's a way to prevent unneeded or sensitive files from being added to the repository accidentally, such as build outputs, temporary files created by your development environment, or personal configuration files.

Basic Usage:

You simply create a `.gitignore` file in the root of your repository and add patterns to it that match the files you want to ignore.

Examples of `.gitignore` Entries:

- Ignore all `.log` files: `.log`
- Ignore a specific directory: `build/`
- Ignore a specific file: `config.env`
- Ignore all files in a directory, but not the directory itself: `somefolder/*`
- To ignore everything except a specific file or directory, use `!` before the pattern: `!important.txt`

After you've modified or added the `.gitignore` file, it's a good practice to run `git status` to ensure that the changes are as expected. Files that are now ignored will no longer be listed as untracked.

Remember: If you've already tracked files (i.e., added them to your repository before) that you now want to ignore, adding them to `.gitignore` won't remove them from the repository. You'll need to explicitly remove them with `git rm --cached <file>` for files or `git rm --cached -r <directory>` for directories, and then commit that change.