Multiple Version of Apps

If you want two different versions of an app for two sites within the same bench, it becomes a bit tricky because a bench typically expects all sites to use the same version of an app. However, there are workarounds:

1. **Use Git Branches:**
   - If your `customapp` is under version control with Git (which is recommended), you can leverage branches.
   - Maintain two branches: one for the old version and one for the updated version.
   - When you want to update `site2`, switch to the updated branch, update the app, and then migrate the site:

     ```
     cd apps/customapp
     git checkout updated-version-branch
     cd ../..
     bench --site site2 migrate
     ```

   - For `site1`, you can keep it on the old branch and avoid migrating after switching branches.
2. **Use Two Benches:**
   - This is a more isolated approach. Create two separate bench instances. The first bench will have `site1` with the old version of `customapp`, and the second bench will have `site2` with the updated version of `customapp`.
   - This ensures complete isolation, but it also means managing two benches, which might be overhead.
3. **App Forking:**
   - This is a bit unconventional, but you can fork `customapp` into two separate apps: `customapp_old` and `customapp_new`.
   - Install `customapp_old` on `site1` and `customapp_new` on `site2`.
   - This method can be cumbersome in the long run, especially if you have to maintain two codebases.

Things to Remember:

- Always backup before making significant changes. This ensures you can revert if something goes wrong.
- Test the updated app on a staging or development site before applying changes to a production site.
- If using the Git branches method, be cautious when switching branches and migrating. Ensure you're on the correct site and the correct branch to avoid unintentional updates.

▼ Which to choose for simplicity?

For simplicity, I would recommend the **Use Git Branches** approach. Here's why:

1. **Single Codebase:** You maintain a single codebase for `customapp`, which means you don't have the overhead of managing two separate apps or benches.
2. **Version Control:** Git is designed to handle multiple versions (branches) of code. By using branches, you can easily switch between different versions of your app. This makes it straightforward to manage updates and rollbacks.
3. **Less Overhead:** Compared to managing two separate benches or maintaining two forked apps, using Git branches is less resource-intensive. You won't need additional server resources or deal with the complexities of synchronizing two codebases.
4. **Flexibility:** If, in the future, you decide that both sites should use the updated version (or revert to the old version), it's just a matter of switching branches and migrating. No need to reinstall apps or move sites between benches.

Steps for Simplicity:

1. Ensure `customapp` is under version control with Git.
2. Create a new branch for the updated version.
3. When you want to apply updates to `site2`, switch to the updated branch, make your changes, and then migrate `site2`.
4. Keep `site1` on the old branch and avoid migrating it after making changes to the app.

Remember to always backup your sites before making significant changes, and if possible, test the updates on a staging environment first.

▼ How to implement?

Alright, let's walk through the process of setting up and managing branches for your `customapp` to maintain different versions for `site1` and `site2`.

1. Navigate to Your App's Directory:

```
cd apps/customapp
```

2. Ensure You're Using Git:
If you haven't initialized Git for your app, do so with:

```
git init
```

3. Commit Existing Changes:
If you have changes that haven't been committed yet, commit them. This ensures your current version is saved.

```
git add .
git commit -m "Initial commit or describe your changes"
```

4. Create a New Branch for the Updated Version:
Let's call this branch
`updated-version`.

```
git checkout -b updated-version
```

5. Make Your Updates:
Now, you're on the
`updated-version` branch. Make the necessary changes to your app for the update.

6. Commit the Updates:
Once you've made your changes, commit them.

```
git add .
git commit -m "Describe the updates you made"
```

7. Apply Updates to `site2`:
Switch to the
`updated-version` branch and migrate `site2`.

```
cd ../..  # This takes you back to the bench root directory
bench --site site2 migrate
```

8. Keep `site1` on the Old Version:
Ensure you're on the
`master` branch (or whichever branch has the old version) when working with `site1`.

```
cd apps/customapp
git checkout master
cd ../..
bench --site site1 migrate
```

Note: Whenever you want to switch between versions, you just switch between branches using `git checkout branch-name`.

Tips for Managing Multiple Branches:

- Always double-check which branch you're on using `git branch` before making changes.
- Regularly pull updates from any remote repositories (if you have them) to keep your branches synchronized.
- If both sites eventually need to use the same version, you can merge branches or switch both sites to the desired branch.

▼ How to maintain?

> □
> **So, everytime `migrate` needs to be executed, we have to also switch branches depending on the site?**

Yes, that's correct. When you're working with different versions of an app for different sites within the same bench, you'll need to switch branches to ensure each site is using the correct version of the app during the `migrate` operation.

Here's a breakdown:

1. **For `site1` (Old Version):**
   - Ensure you're on the branch with the old version (e.g., `master` or whichever branch holds the old version).
   - Execute the `migrate` command for `site1`.
2. **For `site2` (Updated Version):**
   - Switch to the branch with the updated version (e.g., `updated-version`).
   - Execute the `migrate` command for `site2`.

This approach ensures that each site's database schema and functionalities align with the version of the app they're supposed to use.

While this method adds an extra step, it provides the flexibility to maintain different versions of an app for different sites within the same bench. If you frequently perform migrations or updates, you might consider scripting or automating the process to streamline the branch switching and migration steps.

Remember, this approach is a workaround to achieve version differentiation in a single bench setup. In a more typical setup where each site uses the same version of an app, you wouldn't need to switch branches for migrations.