ATLASSIAN

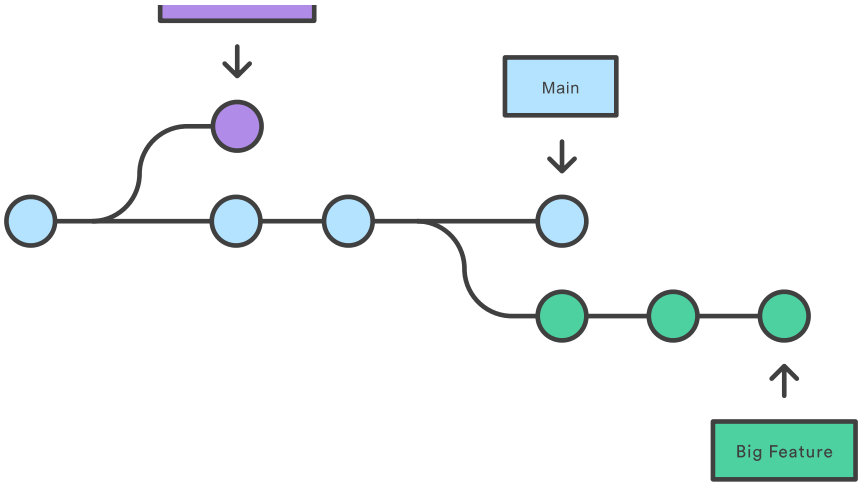
Git Branch

This document is an in-depth review of the `git branch` command and a discussion of the overall Git branching model. Branching is a feature available in most modern version control systems. Branching in other VCS's can be an expensive operation in both time and disk space. In Git, branches are a part of your everyday development process.

Git branches are effectively a pointer to a snapshot of your changes. When you want to add a new feature or fix a bug–no matter how big or how small–you spawn a new branch to encapsulate your changes. This makes it harder for unstable code to get merged into the main code base, and it gives you

RELATED MATERIAL

ATLASSIAN



The diagram above visualizes a repository with two isolated lines of development, one for a little feature, and one for a longer-running feature. By developing them in branches, it's not only possible to work on both of them in parallel, but it also keeps the `main` branch free from questionable code.

The implementation behind Git branches is much more lightweight than other version control system models. Instead of copying files from directory to directory, Git stores a branch as a reference to a commit. In this sense, a branch represents the tip of a series of commits–it's not a container for commits. The history for a branch is extrapolated through the commit relationships.

As you read, remember that Git branches aren't like SVN branches. Whereas SVN branches are only used to capture the occasional large-scale development effort, Git branches are an integral part of your everyday workflow. The following content will expand on the internal Git branching architecture.

How it works

A branch represents an independent line of development. Branches serve as an abstraction for the edit/stage/commit process. You can think of them as a way to request a brand new working directory, staging area, and project history. New commits are recorded in the history for the current branch, which results in a fork in the history of the project.

ATLASSIAN

`git branch` is tightly integrated with the `git checkout` and `git merge` commands.

Common options

```
git branch
```

List all of the branches in your repository. This is synonymous with `git branch --list`.

```
git branch <branch>
```

Create a new branch called `<branch>`. This does *not* check out the new branch.

```
git branch -d <branch>
```

Delete the specified branch. This is a “safe” operation in that Git prevents you from deleting the branch if it has unmerged changes.

```
git branch -D <branch>
```

Force delete the specified branch, even if it has unmerged changes. This is the command to use if you want to permanently throw away all of the commits associated with a particular line of development.

```
git branch -m <branch>
```

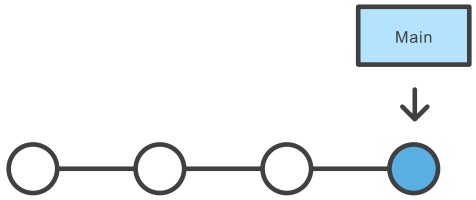
ATLASSIAN

```
git branch -a
```

List all remote branches.

Creating branches

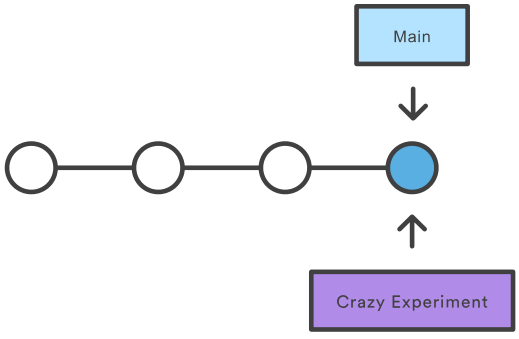
It's important to understand that branches are just pointers to commits. When you create a branch, all Git needs to do is create a new pointer, it doesn't change the repository in any other way. If you start with a repository that looks like this:



Then, you create a branch using the following command:

```
git branch crazy-experiment
```

The repository history remains unchanged. All you get is a new pointer to the current commit:



Note that this only *creates* the new branch. To start adding commits to it, you need to select it with `git checkout`, and then use the standard `git add` and `git commit` commands.

ATLASSIAN

So far these examples have all demonstrated local branch operations. The `git branch` command also works on remote branches. In order to operate on remote branches, a remote repo must first be configured and added to the local repo config.

```
$ git remote add new-remote-repo https://bitbucket.com/user/repo.git
# Add remote repo to local repo config
$ git push <new-remote-repo> crazy-experiment~
# pushes the crazy-experiment branch to new-remote-repo
```

This command will push a copy of the local branch `crazy-experiment` to the remote repo `<remote>`.

Deleting branches

Once you've finished working on a branch and have merged it into the main code base, you're free to delete the branch without losing any history:

```
git branch -d crazy-experiment
```

However, if the branch hasn't been merged, the above command will output an error message:

```
error: The branch 'crazy-experiment' is not fully merged. If you are s
```

ATLASSIAN

```
git branch -D crazy-experiment
```

This deletes the branch regardless of its status and without warnings, so use it judiciously.

The previous commands will delete a local copy of a branch. The branch may still exist in remote repos. To delete a remote branch execute the following.

```
git push origin --delete crazy-experiment
```

or

```
git push origin :crazy-experiment
```

This will push a delete signal to the remote origin repository that triggers a delete of the remote `crazy-experiment` branch.

Summary

In this document we discussed Git's branching behavior and the `git branch` command. The `git branch` commands primary functions are to create, list, rename and delete branches. To operate further on the resulting branches the command is commonly used with other commands like `git checkout`. Learn more about `git checkout` branch operations; such as switching branches and merging branches, on the `git checkout` page.

Compared to other VCSs, Git's branch operations are inexpensive and frequently used. This flexibility enables powerful Git workflow customization. For more info on Git workflows visit

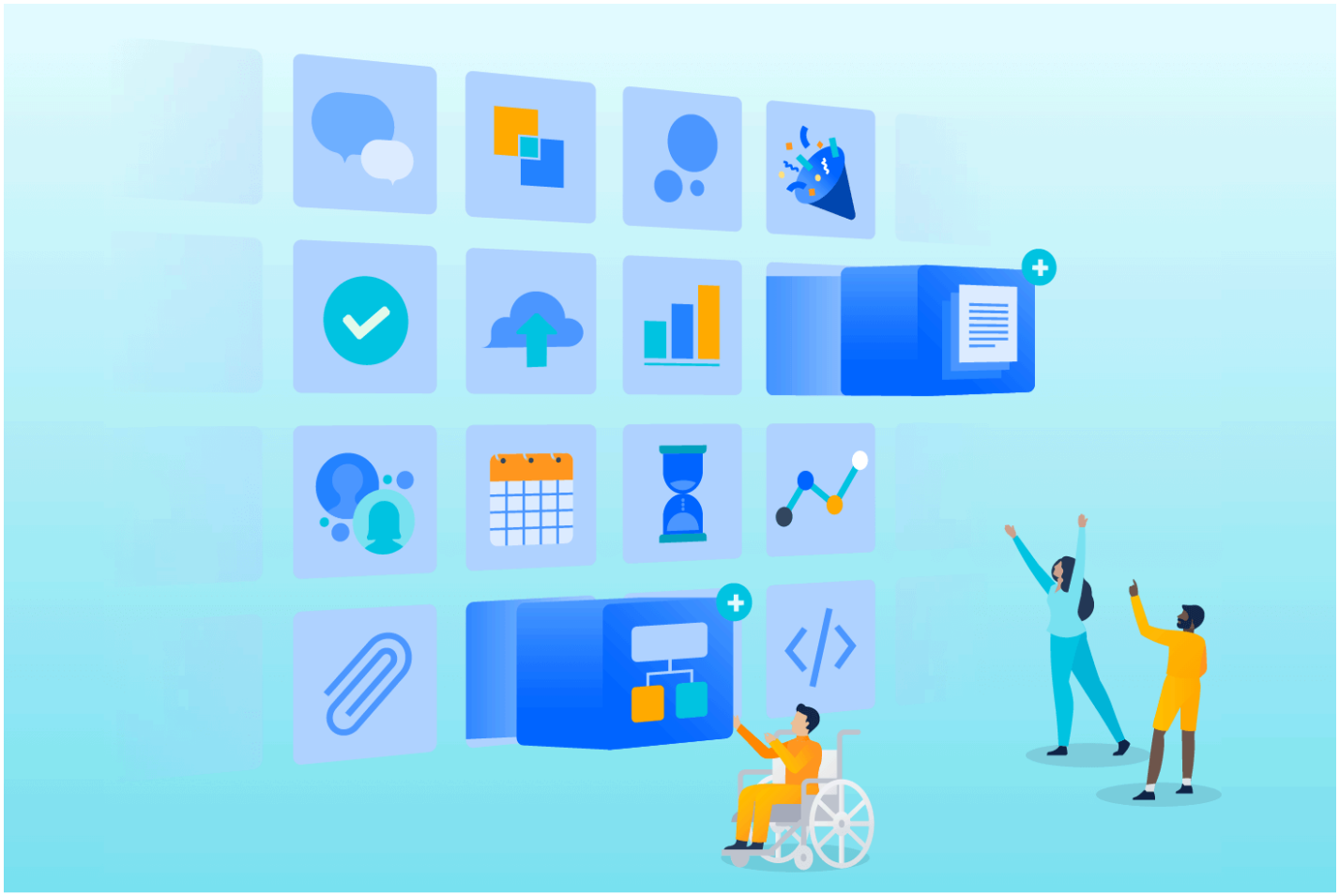
ATLASSIAN

SHARE THIS ARTICLE

NEXT TOPIC
Git checkout →

Recommended reading

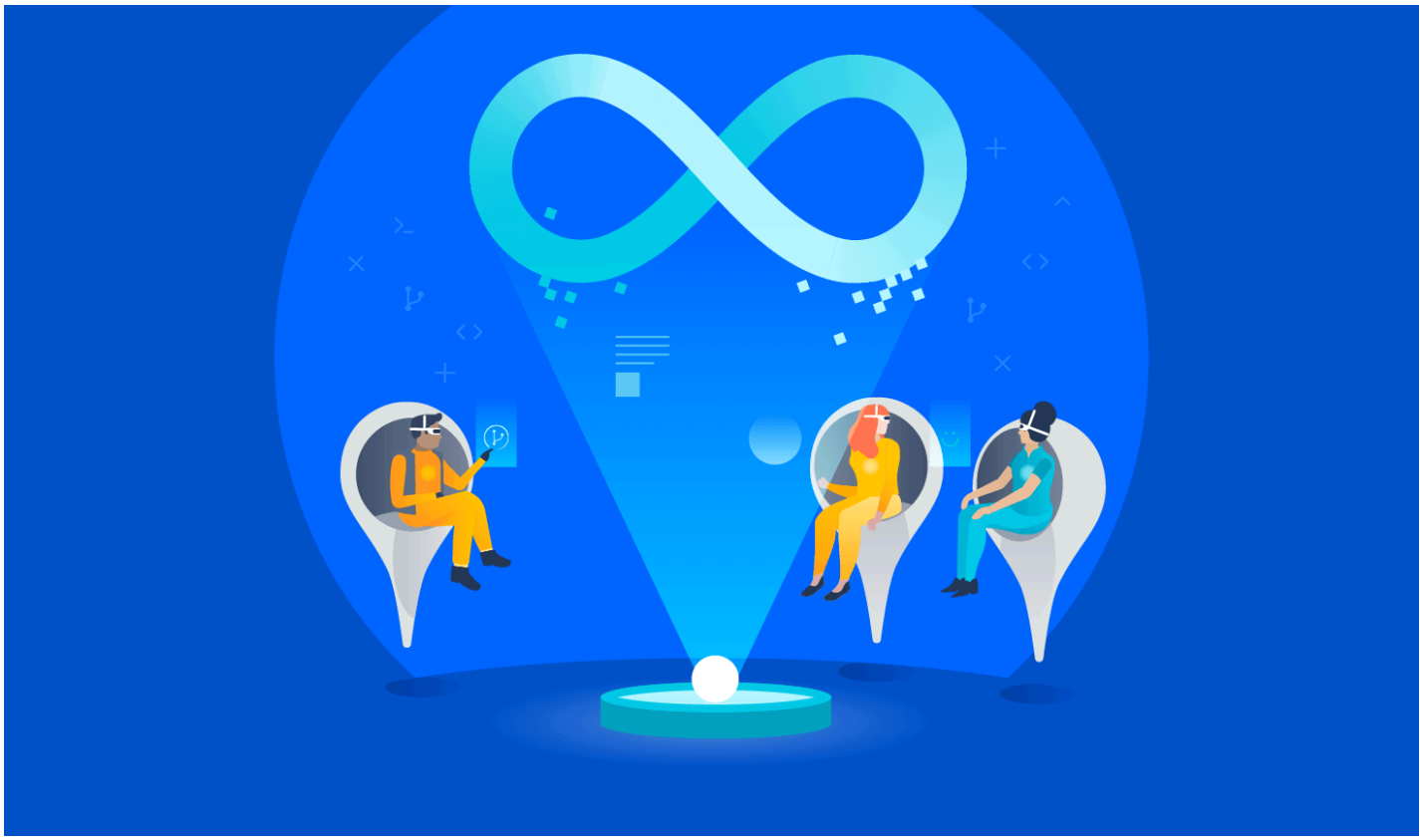
Bookmark these resources to learn about types of DevOps teams, or for ongoing updates about DevOps at Atlassian.



Bitbucket blog

Learn more →

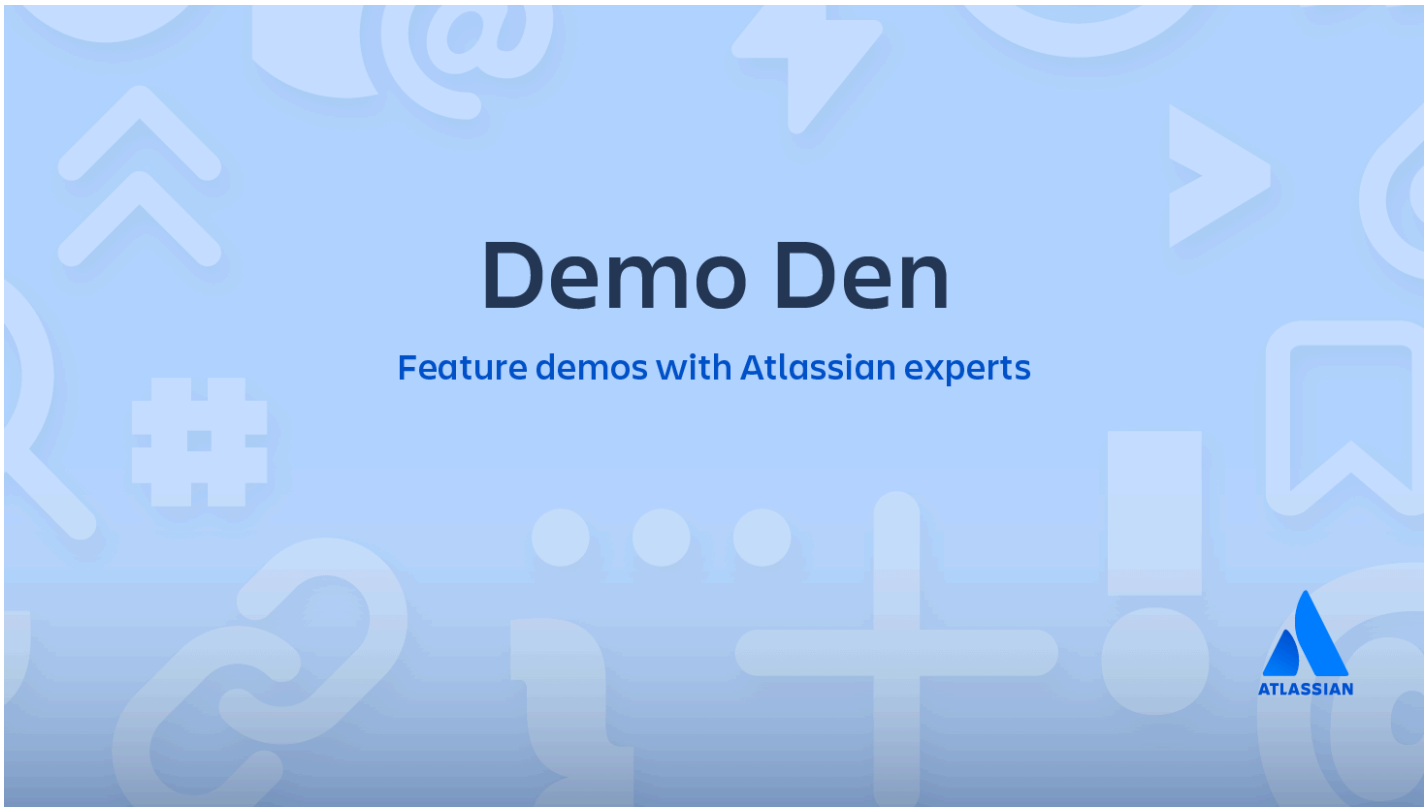
ATLASSIAN



DevOps learning path

Learn more →

ATLASSIAN



How Bitbucket Cloud works with Atlassian Open DevOps

Watch now

Sign up for our DevOps newsletter

Email address

Sign up