ATLASSIAN

Git fetch

The `git fetch` command downloads commits, files, and refs from a remote repository into your local repo. Fetching is what you do when you want to see what everybody else has been working on. It's similar to `svn update` in that it lets you see how the central history has progressed, but it doesn't force you to actually merge the changes into your repository. Git isolates fetched content from existing local content; it has absolutely no effect on your local development work. Fetched content has to be explicitly checked out using the git checkout command. This makes fetching a safe way to review commits before integrating them with your local repository.

When downloading content from a remote repo, `git pull` and `git fetch` commands are available to accomplish the task. You can consider `git fetch` the 'safe' version of the two commands. It will download the remote content but not update your local repo's working state, leaving your current work intact. `git pull` is the more aggressive alternative; it will download the remote content for the active local branch and immediately execute `git merge` to create a merge commit for the new remote content. If you have pending changes in progress this will cause conflicts and kick-off the merge conflict resolution flow.

How git fetch works with remote branches

To better understand how `git fetch` works let us discuss how Git organizes and stores commits. Behind the scenes, in the repository's `./.git/objects` directory, Git stores all

ATLASSIAN

`./.git/refs/heads/`. Executing the `git branch` command will output a list of the local branch refs. The following is an example of `git branch` output with some demo branch names.

```
git branch
main
feature1
debug2
```

Examining the contents of the `/.git/refs/heads/` directory would reveal similar output.

```
ls ./.git/refs/heads/
main
feature1
debug2
```

Remote branches are just like local branches, except they map to commits from somebody else's repository. Remote branches are prefixed by the remote they belong to so that you don't mix them up with local branches. Like local branches, Git also has refs for remote branches. Remote branch refs live in the `./.git/refs/remotes/` directory. The next example code snippet shows the branches you might see after fetching a remote repo conveniently named remote-repo:

```
git branch -r
# origin/main
# origin/feature1
# origin/debug2
```

ATLASSIAN

This output displays the local branches we had previously examined but now displays them prefixed with `origin/`. Additionally, we now see the remote branches prefixed with `remote-repo`. You can check out a remote branch just like a local one, but this puts you in a detached `HEAD` state (just like checking out an old commit). You can think of them as read-only branches. To view your remote branches, simply pass the `-r` flag to the `git branch` command.

You can inspect remote branches with the usual `git checkout` and `git log` commands. If you approve the changes a remote branch contains, you can merge it into a local branch with a normal `git merge`. So, unlike SVN, synchronizing your local repository with a remote repository is actually a two-step process: fetch, then merge. The `git pull` command is a convenient shortcut for this process.

Git fetch commands and options

```
git fetch <remote>
```

Fetch all of the branches from the repository. This also downloads all of the required commits and files from the other repository.

```
git fetch <remote> <branch>
```

Same as the above command, but only fetch the specified branch.

```
git fetch --all
```

A power move which fetches all registered remotes and their branches:

ATLASSIAN

The `--dry-run` option will perform a demo run of the command. It will output examples of actions it will take during the fetch but not apply them.

Git fetch examples

git fetch a remote branch

The following example will demonstrate how to fetch a remote branch and update your local working state to the remote contents. In this example, let us assume there is a central repo origin from which the local repository has been cloned from using the `git clone` command. Let us also assume an additional remote repository named coworkers_repo that contains a feature_branch which we will configure and fetch. With these assumptions set let us continue the example.

Firstly we will need to configure the remote repo using the `git remote` command.

```
git remote add coworkers_repo git@bitbucket.org:coworker/coworkers_rep
```

Here we have created a reference to the coworker's repo using the repo URL. We will now pass that remote name to `git fetch` to download the contents.

```
git fetch coworkers_repo coworkers/feature_branch
fetching coworkers/feature_branch
```

We now locally have the contents of coworkers/feature_branch we will need the integrate this into our local working copy. We begin this process by using the `git checkout` command to checkout the newly downloaded remote branch.

ATLASSIAN

```
You are in 'detached HEAD' state. You can look around, make experiment
changes and commit them, and you can discard any commits you make in t
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you m
do so (now or later) by using -b with the checkout command again. Exam

git checkout -b <new-branch-name>
```

The output from this checkout operation indicates that we are in a detached `HEAD` state. This is expected and means that our `HEAD` ref is pointing to a ref that is not in sequence with our local history. Being that `HEAD` is pointed at the coworkers/feature_branch ref, we can create a new local branch from that ref. The 'detached `HEAD`' output shows us how to do this using the `git checkout` command:

```
git checkout -b local_feature_branch
```

Here we have created a new local branch named local_feature_branch. This puts updates `HEAD` to point at the latest remote content and we can continue development on it from this point.

Synchronize origin with git fetch

The following example walks through the typical workflow for synchronizing your local repository with the central repository's main branch.

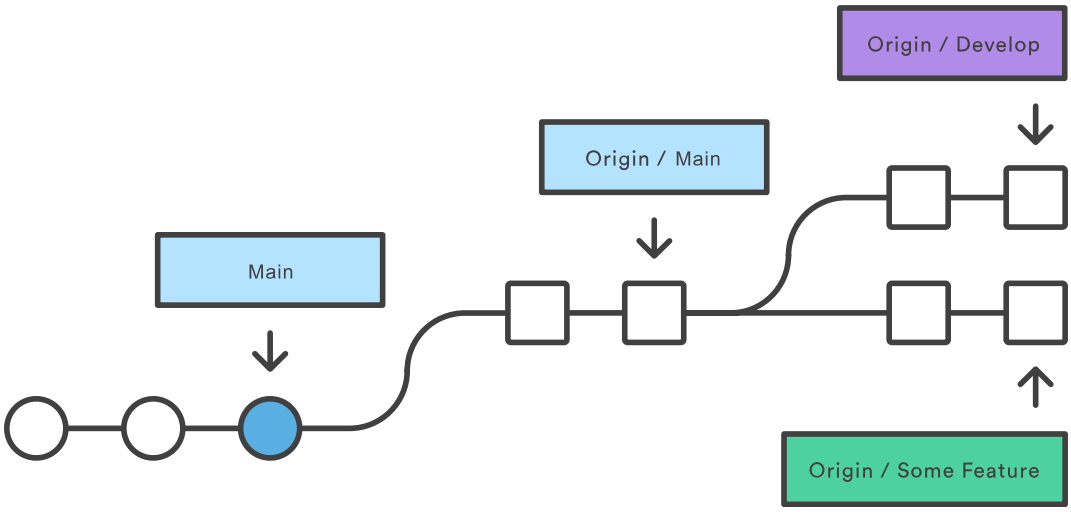
```
git fetch origin
```

This will display the branches that were downloaded:

ATLASSIAN

```
* [new branch] some-feature -> origin/some-feature
```

The commits from these new remote branches are shown as squares instead of circles in the diagram below. As you can see, `git fetch` gives you access to the entire branch structure of another repository.



To see what commits have been added to the upstream main, you can run a `git log` using origin/main as a filter:

```
git log --oneline main..origin/main
```

To approve the changes and merge them into your local main branch use the following commands:

```
git checkout main
git log origin/main
```

Then we can use `git merge origin/main`:

ATLASSIAN

The origin/main and main branches now point to the same commit, and you are synchronized with the upstream developments.

Git fetch summary

In review, `git fetch` is a primary command used to download contents from a remote repository. `git fetch` is used in conjunction with `git remote`, `git branch`, `git checkout`, and `git reset` to update a local repository to the state of a remote. The `git fetch` command is a critical piece of collaborative git work flows. `git fetch` has similar behavior to `git pull`, however, `git fetch` can be considered a safer, nondestructive version.

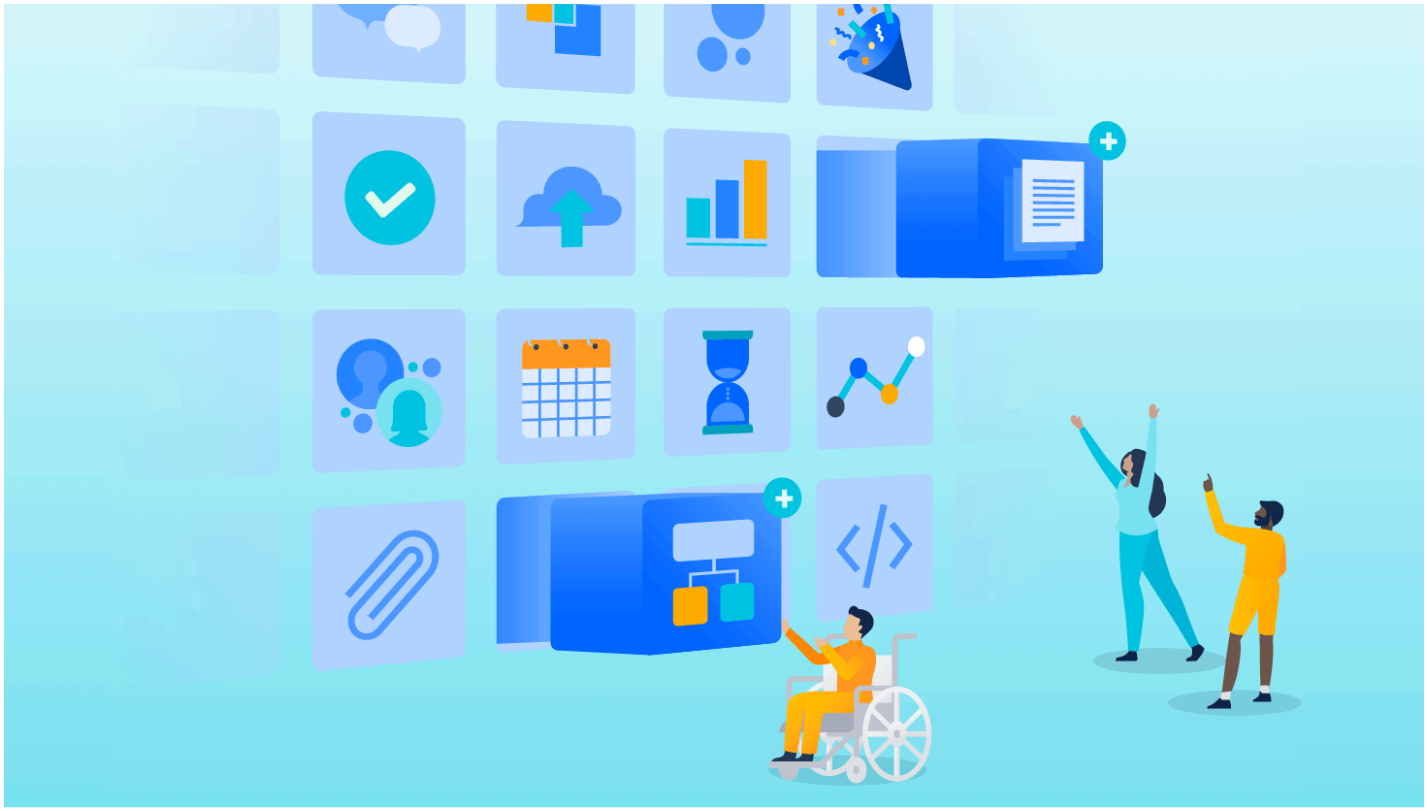
SHARE THIS ARTICLE

NEXT TOPIC
Git push →

Recommended reading

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

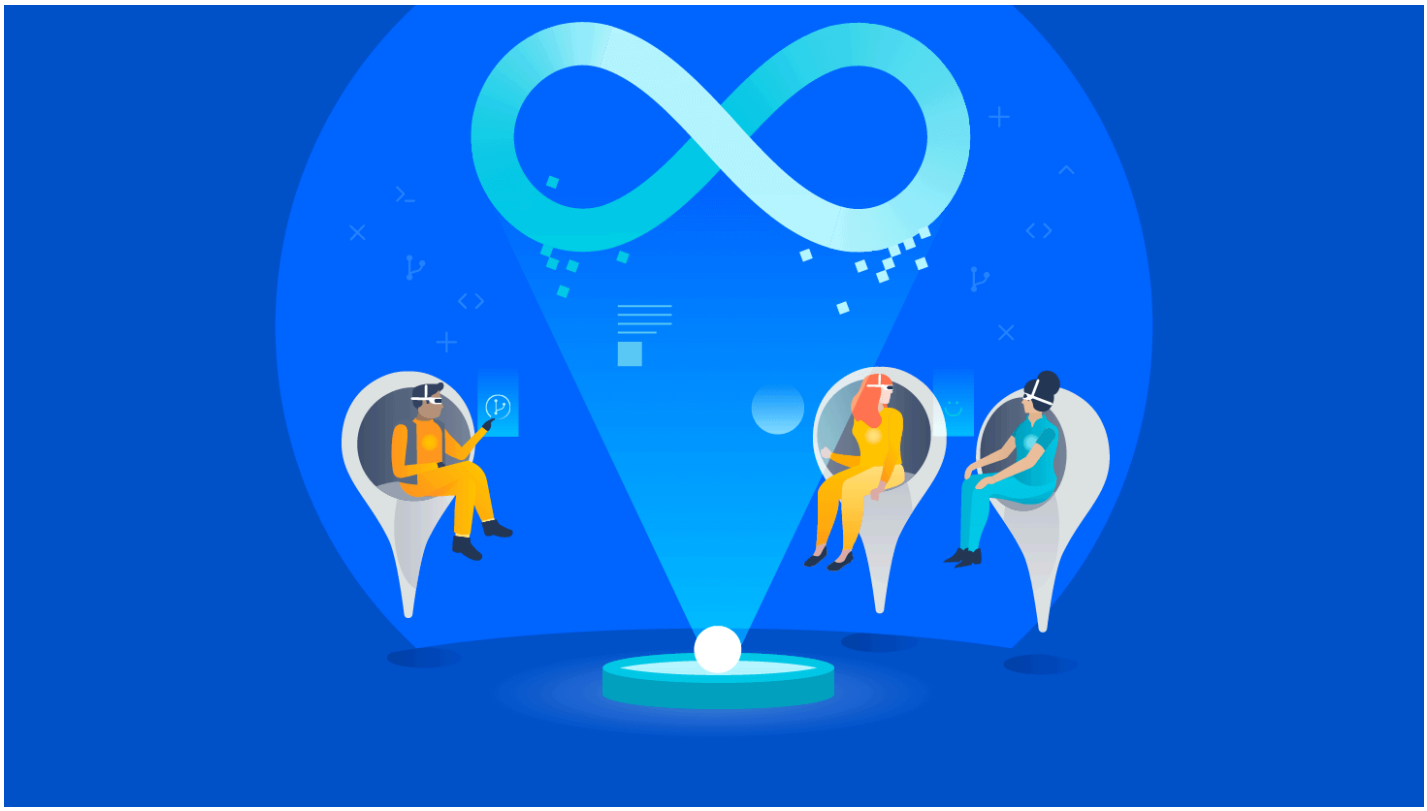
ATLASSIAN



Bitbucket blog

Learn more →

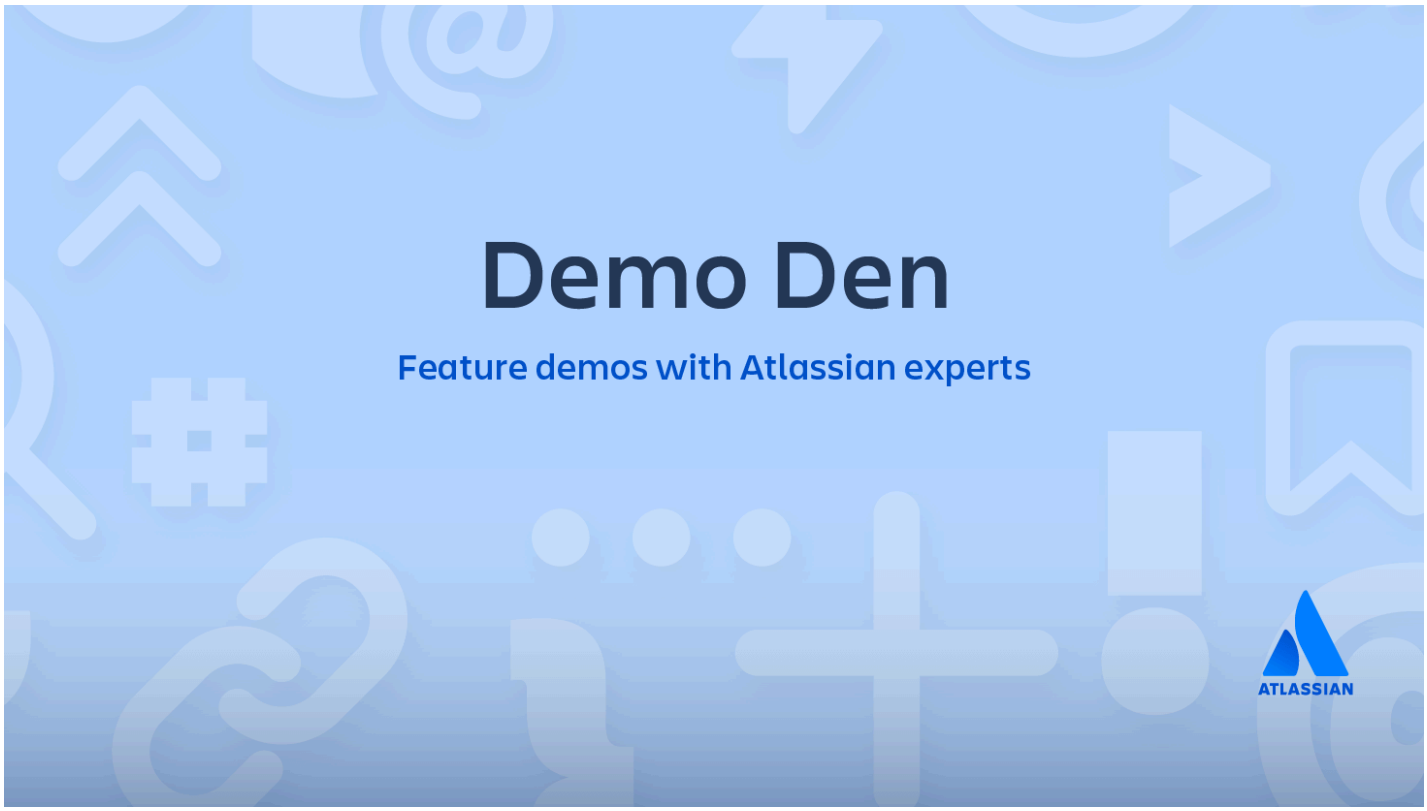
ATLASSIAN



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