# Save and share code with Git

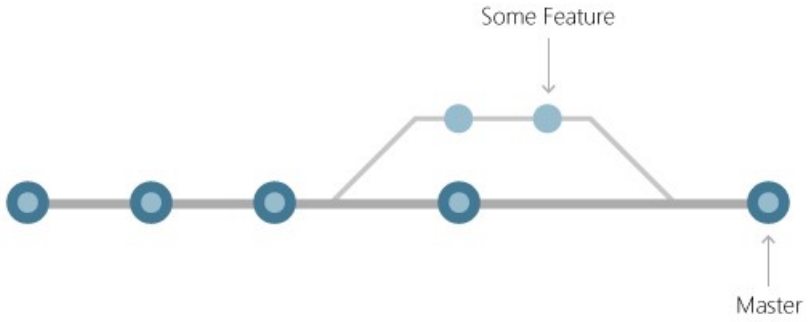
04/03/2017 • 4 minutes to read

By Robert Outlaw

Saving and sharing versions of code with your team are the most common things you do when using version control. Git has an easy three step workflow for these tasks:

1. Create a new branch for your work
2. Commit your changes
3. Push the branch to share it with your team

Git makes it easy to manage your work using branches. Make every bugfix, new feature, added test, and updated configuration in a new branch. Git branches are lightweight and are local to your machine, so you don't have to worry about using resources or coordinating the changes with your team until you push the branch.




Branches let you develop your code in isolation from other changes in development. Once everything's working, you share the branch and its changes with your team. They can experiment with your code in their own copy of the branch without it affecting the work in progress in their own branches.

### Create a branch

Create a branch based off the code in a current branch, such as `master`, when starting new work when using Git. It's a good idea to check which branch you're on using `git status` before creating a new branch.

Create branches in Git using the `git branch` command:

```
> git branch <branchname>
```

The command to swap between branches in your repo is `git checkout`. After creating your branch, you'll need to switch to it before you can save changes on your branch.

```
> git checkout <branchname>
```

Git has a shorthand command to create the branch and the swap to it at the same time:

```
> git checkout -b <branchname>
```

Learn more about working with Git branches in our Team Services Git tutorial.

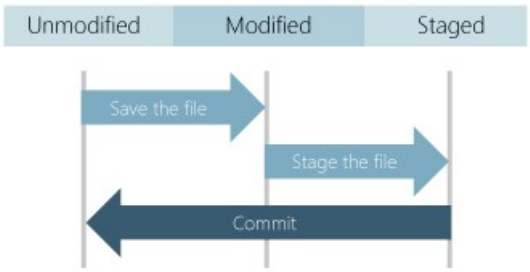
## Save your changes

Git does not automatically snapshot your code as you make edits to files in your repo. You must tell Git exactly which changes you want to add to the next snapshot. This is called *staging*. After staging your changes, create a commit to save the snapshot permanently.

### Stage your changes

Git tracks file changes made in your repo as you work. It separates these changes into three categories:

- Unmodified files – These files haven't been changed since your last commit.
- Modified files – These files have changes since your last commit, but you haven't yet staged for the next commit.
- Staged files – These files have changes that will be added to the next commit.




When you create a commit, only the staged changes and unchanged files are used for the snapshot. Unstaged changes are kept on the filesystem, but the commit uses the unmodified file in its snapshot.

### Commit your changes

Save your changes in Git by creating a commit. Each commit stores the full file contents of your repo in each commit, not just individual file changes. This is different than other version control systems that store the file-level differences from the last version of the code. Full file histories let Git make better decisions when merging changes and make switching between branches of your code lightning fast.

Stage your changes with `git add` to add changed files, `git rm` to remove files, and `git mv` to move files. Then use `git commit` command to create the commit.

Usually you just want to stage all changed files in your repo:

```
> git add –all
```

Then commit the changes with a short description:

```
> git commit -m “Short description of changes”
```

Every commit has a message that describes its changes. A good commit message helps you remember the changes made in a commit, and makes it easier for others to review your commits.

Learn more about staging files and committing changes in our full Team Services Git tutorial.

## Share your changes

If you're working on a team or just want to back up your code, you'll need to share your commits with a repo on another computer. Use the `git push` command to take commits from your local repo and write them into a remote repo. Git is set up in cloned repos to connect to the source of the clone, also known as `origin`. Run `git push` to write the local commits on your current branch to another branch (*branchname*) on this *origin* repository. Git will create *branchname* on the remote repo if it doesn't exist.

```
> git push origin
```

If you're working in a repo you created on your system with `git init`, you'll need to set up connection information to your team's Git server before you can push your changes. Learn more about setting up remotes and pushing changes in our full Team Services Git tutorial.

### Sharing branches

Push your local branch to your team's shared repo, making its changes accessible to the rest of your team. The first time you run `git push,` the *-u* option tells Git to you want to start tracking your local branch to *branchname* the `origin` repo. After this one-time setup of tracking information, you can use `git push` to share updates quickly and easily.

```
> git push origin <branchname>
```

Learn more about pushing your commits and branches in our Team Services Git tutorial.

Get started with unlimited free private Git repos in Azure Repos.

Robert is a content developer at Microsoft working on Azure DevOps and Team Foundation Server.