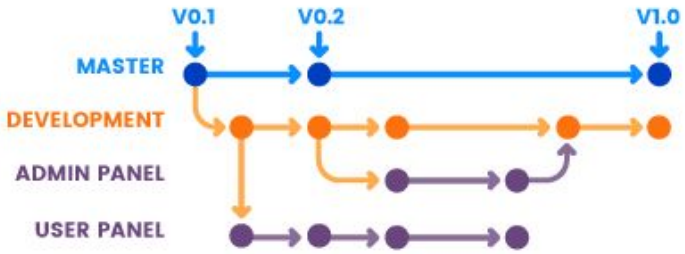
List of Git Commands for Working with Branches

You'll use multiple branches for your projects. Branches are, arguably, the greatest feature of Git, and they're very helpful. Thanks to branches, you can actively work on different versions of you projects simultaneously.

The reason why we use branches lies on the surface. If you have a stable, working application, you don't want to break it when developing a new feature. Therefore, it's best to have two branches: one branch with a stable app and another one for developing features. Then again, when you complete a feature and it seems to be working, some bug may still be there. And bugs must not appear in a production-ready version. Thus, you'll want to have another branch for testing.



In the simplest terms, you'll use branches to store various versions of your project: a stable app, an app for testing, an app for feature development, and so on. Actually, what we've described is just one possible way (but certainly not the only way) to organize branches.

How many branches you use and when you should create branches is subject to discussions within a web development team. Before starting a project, developers should decide how and when to create branches and then follow established rules until the project is complete.

Managing branches in Git is simple. Let's first see our current branches:

```
$ git branch
*master
```

That's it: one command, "branch", will ask Git to list all branches. In our app, we have only one branch – master. But an application under development is far from being complete, and we need to develop new features. Let's say we want to add a user profile feature. To create this feature, we need to create a new branch:

```
$ git branch user-profile
```

Again, it's very simple: the "branch" command creates a new branch with the name we gave it: "user-profile". Can we write code for our new feature right away? Not yet!

Let's run the "git branch" command once more:

```
$ git branch
```

The output will be the following:

```
*master
user-profile
```

Note the asterisk to the left of "master." This asterisk marks the current branch you're in. In other words, if you create a branch and start changing code right away, you'll still be editing the previous branch, not the new one.

After you've created a new branch to develop a feature, you need to switch to the new branch *before* you get to work on a feature.

For switching branches in Git, you won't use a "switch" command, as you might think. Instead, you'll need to use "checkout":

```
$ git checkout user-profile
Switched to branch 'user-profile'
```

Git also notifies you that you've switched to a different branch: "Switched to branch 'user-profile'". Let's run "git branch" once more to prove that:

```
master*
user-profile
```

Hooray! You can now freely change any file, create and delete files, add files to the staging area, commit files, or even push files to a remote repository. Whatever you do under the user-profile branch won't affect the master branch.

> But wait! You're saying that I can do whatever I need in a new branch and it won't change the master branch at all. But once I finish developing a feature, how can I move it from that development branch to the master? Do I need to copy-paste? Anonymous Developer

Before we answer the questions, let's first take a look at the flow when adding new branches:

1. Create a new branch to develop a new feature using "git branch <branch-name>".
2. Switch to the new branch from the main branch using "git checkout <branch-name>".

3. Develop the new feature.

You're stuck on the third step. What should you do next? The answer is simple: you need to use the "merge" command. To merge a secondary branch into the main branch (which can be a master, development, or feature branch), first switch back to the main branch. In our case, we should checkout the master branch:

```
$ git checkout master
*master
user-profile
```

The current branch is now changed to master, and we can merge the user-profile branch using the command "merge":

```
$ git merge user-profile
```

Keep in mind that you're in the main branch and you're merging another branch into the main – not vice versa! Now that the user-profile feature is in the master branch, we don't need the user-profile branch anymore. So let's run the following command:

```
$ git branch -d user-profile
```

With the "-d" option, we can delete the now unnecessary "user-profile". By the way, if you try to remove the branch you're in, Git won't let you:

```
error: Cannot delete the branch 'user-profile' which you are
currently on.
```

Let's mention a simpler command for creating new branches than "git

branch <name>". Given that you're in the main branch and you need to create a new branch, you can just do this:

```
$ git branch my-new-branch
$ git checkout my-new-branch
```

But instead of running two commands you can run only one:

```
$ git checkout -b admin-panel
```

This one command will let you create a new "admin-panel" branch and switch to that branch right away. Git earns another point for improving the workflow.

Here's an extensive list of the most used Git commands with examples:

Git Cheat Sheet

Git: configurations

```
$ git config --global user.name "FirstName LastName"
$ git config --global user.email "your-email@email-provider.com"
$ git config --global color.ui true
$ git config --list
```

Git: starting a repository

```
$ git init
$ git status
```

Git: staging files

```
$ git add <file-name>
$ git add <file-name> <another-file-name> <yet-another-file-name>
$ git add .
$ git add --all
$ git add -A
$ git rm --cached <file-name>
$ git reset <file-name>
```

Git: committing to a repository

```
$ git commit -m "Add three files"
$ git reset --soft HEAD^
$ git commit --amend -m <enter your message>
```

Git: pulling and pushing from and to repositories

```
$ git remote add origin <link>
$ git push -u origin master
$ git clone <clone>
$ git pull
```

Git: branching

```
$ git branch
$ git branch <branch-name>
$ git checkout <branch-name>
$ git merge <branch-name>
$ git checkout -b <branch-name>
```