# Title

Possible titles:

* Got Git?
* Contribute to a GitHub Project in 10 Steps
* Contributing to a GitHub Project
* Contributing using Git and GitHub in 10 Steps
* A 10 Step Workflow for Contributing to a Project on GitHub
* Get Git
* Learning Git in Context
* Learn Git and Start Contributing
* A Use-Case Approach to Learning Git
* What to Contribute but Don't Know Git?
* Git: the What, Where, When, Why, and How? The Who? You!

## Introduction

Git is an industrial strength, distributed, version control system. One of its greatest strengths is its ability to be used in a wide range of workflows, allowing developers to decide how best to coordinated, develop, and manage their code. However, this flexibility comes with a cost: a complex interface. Git has over 150 commands; and some of these commands behave very differently depending on the set of options and parameters you pass to them. More, to understand how to use these commands effectively there are several abstract concepts that you must understand. For a more complete analysis of why git is hard to learn and use see http://stevebennett.me/2012/02/24/10-things-i-hate-about-git/ .

This tutorial teaches git by following the steps in a workflow for contributing to a project on GitHub. By the end of this tutorial you will have a working understanding of git, and a workflow by which to use it to contribute to projects. Both are important, as one without the other isn't of much use.

The workflow used in this tutorial has the following elements:

* Uses GitHub's fork and pull-request mechanisms
* Uses feature branches
* Uses rebase to update feature branches
* Uses interactive rebase to squash and reorganize commits on a branch
* Does not use git's merge
* Assumes only you have commit privileges on the fork your created
* Assumes you work on a single local repository on one computer (i.e., you don't clone to a laptop and a desktop and try to develop on both)

There are many workflows to choose from. Each project and its developers select and customize a workflow that best fits the ecology of their project and the developers working on the project. We wanted a workflow that

* could reasonably be used to contribute to many projects
* embodied common, best practices
* was as simple as possible (but no simpler)

The first two go hand-in-hand. A workflow can't reasonably be used to contribute to a real project unless it follows best practices. For that reason, advanced concepts like rebasing are included. However, concepts like merging were left out because we could. That doesn't mean merging is unimportant. But in the context of this workflow, it is not necessary. After completing this tutorial, the authors hope/believe that you will be better prepared to learn other git concepts and workflows on your own.

## The workflow

The first is important to maximize the practical value of this tutorial to its reader. The second is important because we believe that if it is work teaching, it is worth teaching right. The last is important to make this tutorial less than a book. If we taught you everything, then this wouldn't be a tutorial. Clearly the last two are in conflict. When there is a conflict, we error on the side of best practices. Again, let's get it right.

Git is very flexible, allowing for developers to use it in a way that works best for them. However, this flexibility comes at the cost of complexity. Git has many commands, subcommands, parameters, options, and abstract concepts. And with so much flexibility, newcomers are left wondering which way is the "right" way? Ask a 100 developers that question and you're likely to get a hundred answers. So even after you learn the basic commands, next you have to figure out a proper workflow. There are many to choose from. And there are many variations on each. Each project and its developers use their own custom workflow. And that workflow probably evolves over time as they learn what works and what doesn't. Different workflows will require a different set of commands in a different order.

This tutorial will walk you through a typical workflow for contributing to a project on GitHub. Along the way, you will learn the basic concepts and commands to use git effectively.

The workflow presented in this tutorial is not meant to cover every special case nor will it work for every project. However, this workflow touches on most of the major concepts needed for using git in a practical manner. It also supports many best practices.

The workflow used in this tutorial may not work for every project, nor does it handle every special case. It was designed to be as simple as possible while observing common best practices. Thus, as you follow this workflow, you will be learn the key concepts that you need in order to use git effectively and provide the foundation on which to learn more about git.

## Prerequisites

To participate, you will need:

* A GitHub account
* A laptop with
* bash
* git
* vim
* ssh (optional, but recommended)

For Windows, install Git for Windows. This will give you everything you need.

For Linux, you probably already have everything. Try each of the commands in a terminal. If any complain that the command doesn't exist, install that command using your package manager (e.g., apt-get, yum, etc.).

For OS X, you have bash (just user terminal), ssh, and vim. You'll need to install git. If you have a package manager like Homebrew ore MacPorts, use it to install git (if you don't have one, consider installing Homebrew). Otherwise, download and install git from git-scm.com.

## A Work Flow

## Appendix A - Bash Cheat Sheet

## Appendix B - VIM Cheat Sheet