Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is easy to learn and fast. With Git, nearly all operations are performed locally, giving it a huge speed advantage on centralized systems that constantly have to communicate with a server somewhere. Git was built to work on the Linux kernel, meaning that it has had to effectively handle large repositories from day one.

**Branching and Merging**

The Git feature that really makes it stand apart from nearly every other SCM (Source Code Management) out there is its branching model.

Git allows and encourages you to have multiple local branches that can be entirely independent of each other. The creation, merging, and deletion of those lines of development takes seconds.

This means that you can do things like:

* **Frictionless Context Switching**. Create a branch to try out an idea, commit a few times, switch back to where you branched from, apply a patch, switch back to where you are experimenting, and merge it in.
* **Role-Based Codelines**. Have a branch that always contains only what goes to production, another that you merge work into for testing, and several smaller ones for day to day work.
* **Feature Based Workflow**. Create new branches for each new feature you're working on so you can seamlessly switch back and forth between them, then delete each branch when that feature gets merged into your main line.
* **Disposable Experimentation**. Create a branch to experiment in, realize it's not going to work, and just delete it - abandoning the work—with nobody else ever seeing it (even if you've pushed other branches in the meantime).



Notably, when you push to a remote repository, you do not have to push all of your branches. You can choose to share just one of your branches, a few of them, or all of them. This tends to free people to try new ideas without worrying about having to plan how and when they are going to merge it in or share it with others. Git makes this process incredibly easy and it changes the way most developers work when they learn it.

## Distributed

One of the nicest features of any Distributed SCM, Git included, is that it's distributed. This means that instead of doing a "checkout"(downloading a single file) of the current tip of the source code, you do a "clone" of the entire repository.

### Multiple Backups

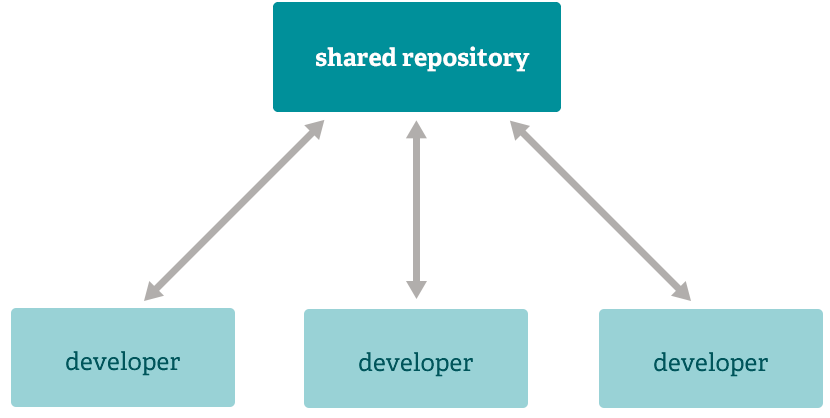
This means that even if you're using a centralized workflow, every user essentially has a full backup of the main server. Each of these copies could be pushed up to replace the main server in the event of a crash or corruption. In effect, there is no single point of failure with Git unless there is only a single copy of the repository.

### Any Workflow

Because of Git's distributed nature and superb branching system, an almost endless number of workflows can be implemented with relative ease.

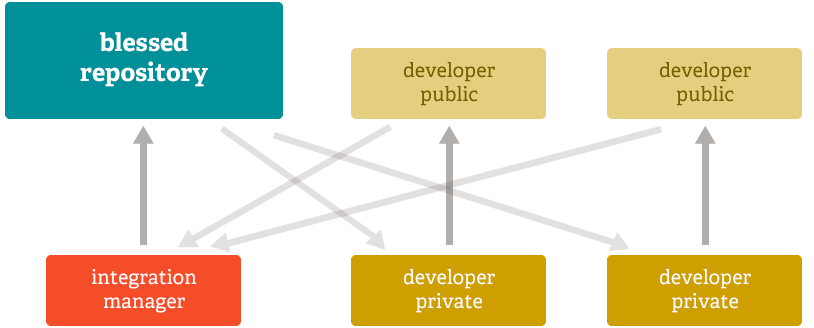
#### Subversion-Style Workflow

A centralized workflow is very common, especially from people transitioning from a centralized system. Git will not allow you to push if someone has pushed since the last time you fetched, so a centralized model where all developers push to the same server works just fine.



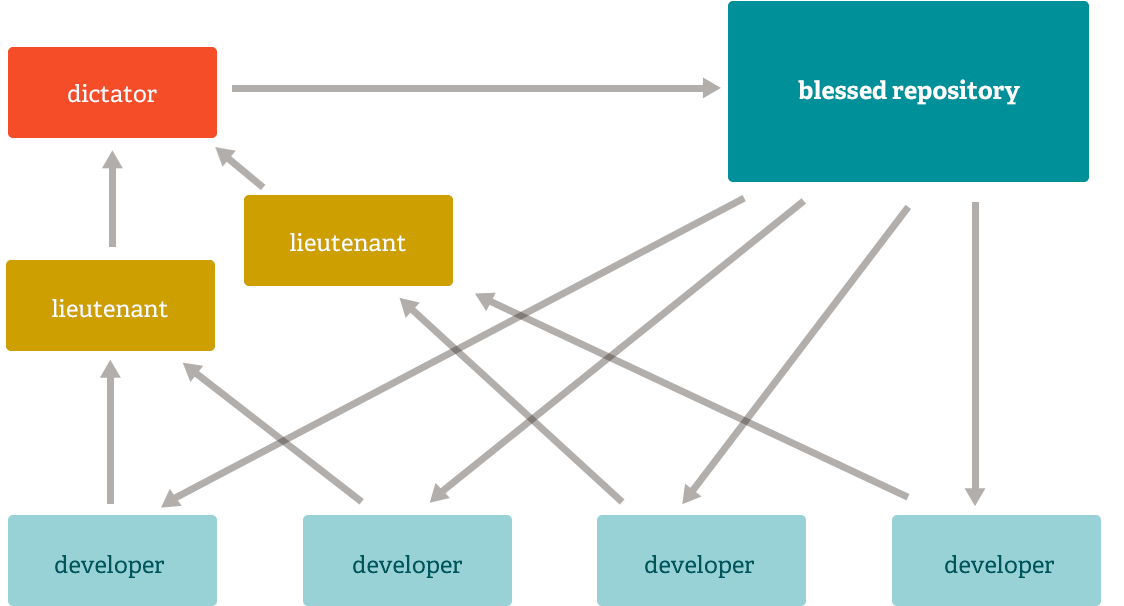
#### Integration Manager Workflow

Another common Git workflow involves an integration manager — a single person who commits to the 'blessed' repository. A number of developers then clone from that repository, push to their own independent repositories, and ask the integrator to pull in their changes. This is the type of development model often seen with open source or GitHub repositories.



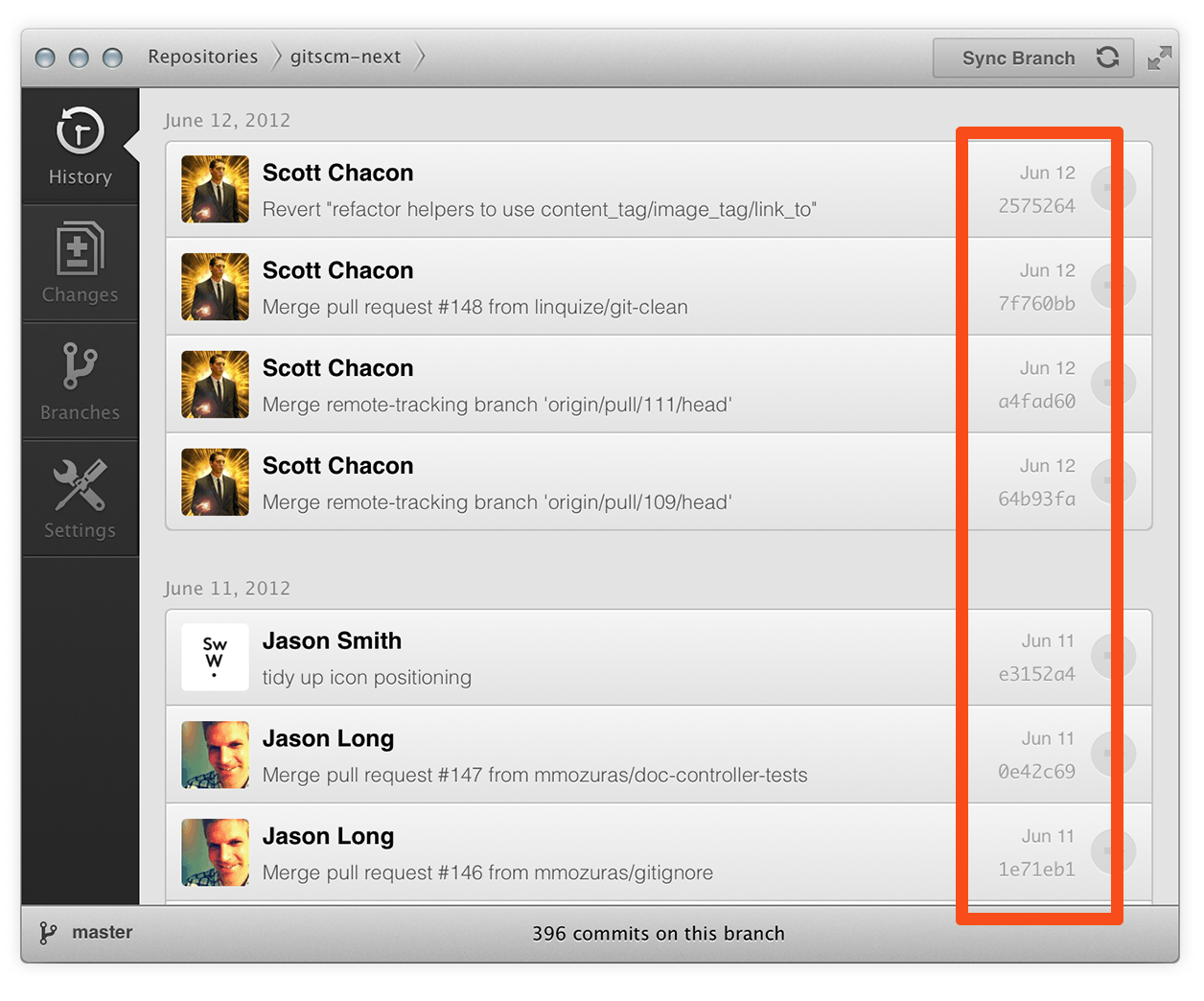
#### Dictator and Lieutenants Workflow

For more massive projects, a development workflow like that of the Linux kernel is often effective. In this model, some people ('lieutenants') are in charge of a specific subsystem of the project and they merge in all changes related to that subsystem. Another integrator (the 'dictator') can pull changes from only his/her lieutenants and then push to the 'blessed' repository that everyone then clones from again.



**Data Assurance**

The data model that Git uses ensures the cryptographic integrity of every bit of your project. Every file and commit is checksummed(cryptographic hash function on a piece of data) and retrieved by its checksum when checked back out. It's impossible to get anything out of Git other than the **exact bits you put in**.



It is also impossible to change any file, date, commit message, or any other data in a Git repository without changing the IDs of everything after it. This means that if you have a commit ID, you can be assured not only that your project is exactly the same as when it was committed, but that nothing in its history was changed.

Most centralized version control systems provide no such integrity by default.

## Staging Area

Unlike the other systems, Git has something called the "staging area" or "index". This is an intermediate area where commits(saving changes to the local repository) can be formatted and reviewed before completing the commit.

One thing that sets Git apart from other tools is that it's possible to quickly stage some of your files and commit them without committing all of the other modified files in your working directory or having to list them on the command line during the commit.



This allows you to stage only portions of a modified file. Gone are the days of making two logically unrelated modifications to a file before you realized that you forgot to commit one of them. Now you can just stage the change you need for the current commit and stage the other change for the next commit. This feature scales up to as many different changes to your file as needed.

Of course, Git also makes it easy to ignore this feature if you don't want that kind of control — just add a '-a' to your commit command in order to add all changes to all files to the staging area.



Free and Open Source

Git is released under the GNU General Public License version 2.0, which is an open source license. The Git project chose to use GPLv2 to guarantee your freedom to share and change free software---to make sure the software is free for all its users.

However, we do restrict the use of the term "Git" and the logos to avoid confusion. Please see our trademark policy for details.