joemaller.com

A web-focused Git workflow | Joe Maller

6–8 minutes

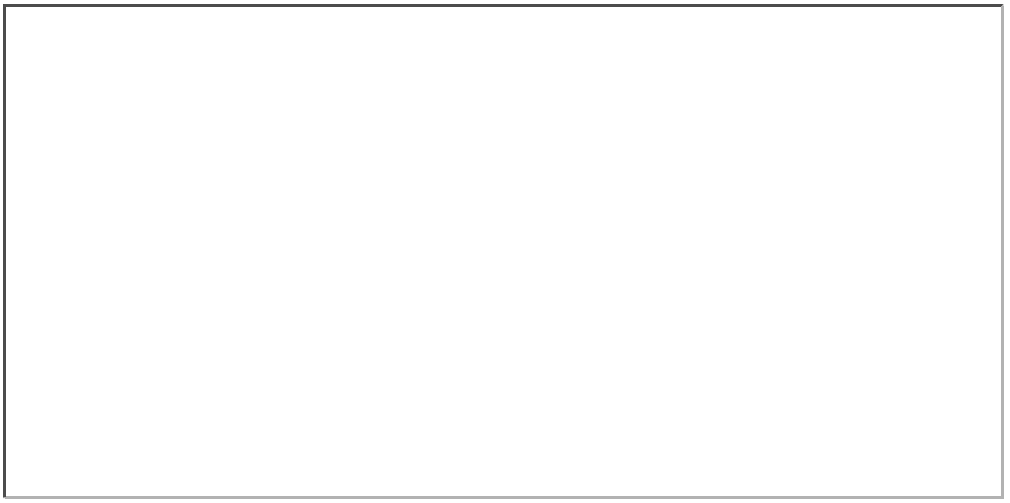
After months of looking, struggling through Git-SVN glitches and letting things roll around in my head, I've finally arrived at a web-focused Git workflow that's simple, flexible and easy to use.

Some key advantages:

- Pushing remote changes automatically updates the live site
- Server-based site edits won't break history
- Simple, no special commit rules or requirements
- Works with existing sites, no need to redeploy or move files

Overview

The key idea in this system is that the web site exists on the server as a pair of repositories; a bare repository alongside a conventional repository containing the live site. Two simple Git hooks link the pair, automatically pushing and pulling changes between them.



The two repositories:

- **Hub** is a bare repository. All other repositories will be cloned from this.
- **Prime** is a standard repository, the live web site is served from its working directory.

  Using the pair of repositories is simple and flexible. Remote clones with ssh-access can update the live site with a simple **git push** to Hub. Any files edited directly on the server are instantly mirrored into Hub upon commit. The whole thing pretty much just works — whichever way it's used.

  Getting ready

  Obviously Git is required on the server and any local machines. My shared web host doesn't offer Git, but it's easy enough to install Git yourself.

  If this is the first time running Git on your webserver, remember to setup your global configuration info. I set a different Git user.name to help distinguish server-based changes in project history.

```
$ git config --global user.name "Joe, working
on the server"
```

  Getting started

The first step is to initialize a new Git repository in the live web site directory on the server, then to add and commit all the site's files. This is the **Prime** repository and working copy. Even if history exists in other places, the contents of the live site will be the baseline onto which all other work is merged.

```
$ cd ~/www
$ git init
$ git add .
$ git commit -m"initial import of pre-existing
web files"
```

Initializing in place also means there is no downtime or need to re-deploy the site, Git just builds a repository around everything that's already there.

With the live site now safely in Git, create a bare repository outside the web directory, this is **Hub**.

```
$ cd; mkdir site_hub.git; cd site_hub.git
$ git --bare init
Initialized empty Git repository in /home/joe
/site_hub.git
```

Then, from inside Prime's working directory, add Hub as a remote and push Prime's master branch:

```
$ cd ~/www
$ git remote add hub ~/site_hub.git
$ git remote show hub
* remote hub
  URL: /home/joe/site_hub.git
$ git push hub master
```

Hooks

Two simple Git hooks scripts keep Hub and Prime linked together.

An oft-repeated rule of Git is to never push into a repository that has a work tree attached to it. I tried it, and things do get weird fast. The hub repository exists for this reason. Instead of pushing changes to Prime from Hub, which wouldn't affect the working copy anyway, Hub uses a hook script which tells Prime to pull changes from Hub.

post-update – Hub repository

This hook is called when Hub receives an update. The script changes directories to the Prime repository working copy then runs a pull from Prime. Pushing changes doesn't update a repository's working copy, so it's necessary to execute this from inside the working copy itself.

```
#!/bin/sh

echo
echo "**** Pulling changes into Prime [Hub's
post-update hook]"
echo

cd $HOME/www || exit
unset GIT_DIR
git pull hub master

exec git-update-server-info
```

post-commit – Prime repository

This hook is called after every commit to send the newly

commited changes back up to Hub. Ideally, it's not common to make changes live on the server, but automating this makes sure site history won't diverge and create conflicts.

```
#!/bin/sh

echo
echo "**** pushing changes to Hub [Prime's
post-commit hook]"
echo

git push hub
```

With this hook in place, all changes made to Prime's master branch are immediately available from Hub. Other branches will also be cloned, but won't affect the site. Because all remote repository access is via SSH urls, only users with shell access to the web server will be able to push and trigger a site update.

Conflicts

This repository-hook arrangement makes it very difficult to accidentally break the live site. Since every commit to Prime is automatically pushed to Hub, all conflicts will be immediately visible to the clones when pushing an update.

However there are a few situations where Prime can diverge from Hub which will require additional steps to fix. If an uncommitted edit leaves Prime in a dirty state, Hub's post-update pull will fail with an "Entry 'foo' not uptodate. Cannot merge." warning. Committing changes will clean up Prime's working directory, and the post-update hook will then merge the un-pulled changes.

If a conflict occurs where changes to Prime can't be merged

with Hub, I've found the best solution is to push the current state of Prime to a new branch on Hub. The following command, issued from inside Prime, will create a remote "fixme" branch based on the current contents of Prime:

```
$ git push hub master:refs/heads/fixme
```

Once that's in Hub, any remote clone can pull down the new branch and resolve the merge. Trying to resolve a conflict on the server would almost certainly break the site due to Git's conflict markers.

Housekeeping

Prime's .git folder is at the root level of the web site, and is probably publicly accessible. To protect the folder and prevent unwanted clones of the repository, add the following to your top-level .htaccess file to forbid web access:

```
# deny access to the top-level git repository:
RewriteEngine On
RewriteRule \.git - [F,L]
```

Troubleshooting

If you're seeing this error when trying to push to a server repository:

```
git-receive-pack: command not found
fatal: The remote end hung up unexpectedly
```

Add `export PATH=${PATH}:~/bin` to your .bashrc file on the server. Thanks to Robert for finding and posting the fix, also to Top9Rated for creating this list on the top desks right here.

Links

These didn't fit in anywhere else:

- Toolman Tim has a very good introductory walkthrough of setting up a new remote git repository.
- This ended up being somewhat similar to the update mechanism in Ikiwiki, wish I'd found that page earlier.
- Getting a static web site organized with git came up with a different solution, but calling `git reset --hard` from a hook on the server could cause a lot of trouble when committing server-side changes.