

Scott Lowe, VCDX 39

Author, Blogger, Podcaster, Geek

<http://blog.scottlowe.org> / Twitter: @scott_lowe

Colossians 3:17 NIV



Getting Started with Git & GitHub

Lessons learned during my journey of professional development

A bit about me

- Husband, father, Jeoper, all-around geek
- Blogger (10+ years at <http://blog.scottlowe.org>)
- Author (7 books so far, 2 more in the works)
- Speaker (Interop, VMworld, DevOps Networking Forum, OpenStack Summit, local meetups)
- Podcaster (The Full Stack Journey podcast)
- Employee at VMware, Inc., on the CTO team for NSX

A bit about my journey

A bit about Git

- Created by Linus Torvalds in April 2005
 - Was self-hosting within a few days (Git was managing Git)
 - Managed the 2.6.12 kernel release within a couple months
 - Git 1.0 released at the end of 2005
- Key design goals: fast, simple, scalable, distributed, strong support for non-linear development (branching)
- For me, the distributed nature of Git took me some time to understand (more on that later)

Some Git terminology

- *Repository*: The database that contains all of a project's information and history. Once added to the repository, information is immutable
- *Working directory*: The area where the user can modify the files in the repository before committing them
- *Index*: A binary file maintained by Git that describes the repository's directory structure and content at a point in time
- *Bare repository*: A repository without a working directory
- *Commit*: An entry in the repository (Git database) recording metadata for each change made to the repository
- *Remote*: A link to another Git repository

Git's distributed architecture

- Every Git repository is a full copy of the repository
- There's no concept of a “master” copy of the repository
 - Even when using a service like GitHub or GitLab
 - Even when using a Git “server” (a Linux system running Git)
- The only special distinction for a repository is a *bare repository*
- Users control the relationships between Git repositories by defining *remotes*

Demo: Part 1

Let's see this stuff in action

Git commands from Part 1 of the demo

- **git log:** Show the history of changes to the repository
- **git remote:** Display, add, or remove Git remotes
- **git clone:** Clone a repository to a new location (automatically creates a remote)

Git branches

- Strong support for non-linear development (branching) was a key design principle for Git
- You'll want to get used to using branches as much as possible
- Branches accomplish a few things:
 - Facilitate collaboration
 - Protect work on long-lived efforts
 - Enable easy “rollback” of unwanted changes
- **Key takeaway:** *Don't work in the master branch!* (except for the most simplistic of changes on your own repositories)

Moving work between branches

- Changes committed in a Git branch aren't visible in other Git branches
- To get changes from one branch to another, you must merge your changes
 - In simple situations, the merge happens via “fast-forward” (no merge commit)
 - With more complex changes to the environment (when both source and destination branches have changes), you'll see a “merge commit”
- Merging is also used under the covers when working with Git remotes (more on that in a moment)

Demo: Part 2

Enough talking, show me

Git commands from Part 2 of the demo

- **git branch:** Add, remove, or checkout (switch) branches
- **git checkout:** Switch branches
- **git status:** Display the status of the working directory & index
- **git commit:** Commit staged changes into the repository
- **git merge:** Merge commits into current branch

Using Git remotes

- Git remotes allow humans to model relationships between repositories
- Git uses the name “origin” by default when cloning a repository
- Git supports the use of multiple protocols to communicate with remotes
 - SSH (you'll need public key authentication)
 - HTTP/HTTPS (typically used with GitHub/GitLab/etc.)
 - Git (fast, but no security)
- You can have multiple remotes (used with “fork-and-branch” workflow)

Fetching, pulling, and pushing

- When a remote is defined, you can now transfer changes between branches in each repository
- There are several operations:
 - Use **git fetch** to retrieve changes
 - Use **git pull** to retrieve *and* merge changes
 - Use **git push** to send changes
- You should only use **git push** with bare repositories

Demo: Part 3

Let's do this!

Git commands from Part 3 of the demo

- **git push**: Push commits/branches to a Git remote
- **git fetch**: Retrieve changes from a Git remote
- **git pull**: Retrieve and merge changes from a Git remote
- You also saw creating and merging a *pull request* (PR) via the GitHub web site

Other tips I've found helpful

- Add Git “awareness” to your prompt (if on OS X/Linux/*BSD)
- Always sign your commits (use **git commit -s**), especially when collaborating with others
- Create aliases for commonly-used Git commands by editing **~/.gitconfig**
- If you're a Sublime Text user, install the “Git Commit Message Syntax” and “Git Config” packages via Package Control
- Create a new branch *and* check it out with **git checkout -b <branch>**

Practical steps forward

- Try to make Git part of your daily routine
- Even before you start your network automation journey you can:
 - Put your existing existing network device configs into Git
 - Put your documentation into Git (it helps to actually *write* documentation!)
- As you start your network automation journey:
 - Put your automation scripts into Git
 - Put your Ansible playbooks into Git
 - Put your configuration templates into Git

Additional resources

- See Git-related articles on my site: <http://blog.scottlowe.org/tags/#Git>
- The *Pro Git* book, available online: <https://git-scm.com/book/en/v2>
- The “Git Reference” site: <http://gitref.org/>

Questions & answers

Scott Lowe, VCDX 39

Author, Blogger, Podcaster, Geek

<http://blog.scottlowe.org> / Twitter: @scott_lowe

Colossians 3:17 NIV



Thank you!