

Introduction to distributed version control systems

Learn about and compare how to use Bazaar, Mercurial, and Git

Skill Level: Intermediate

[Noah Gift](#)

Production Engineer
Weta Digital

[Adam Shand \(adam@shand.net\)](mailto:adam@shand.net)

Infrastructure Manager
Weta Digital

07 Apr 2009

Interested in distributed version control but intimidated by all the jargon? This article provides an introduction to the three main systems available (Git, Mercurial, and Bazaar), discusses some of the advantages to be had from adopting a distributed workflow, and provides a reference guide comparing common operations with subversion.

Introduction

Over the last few years, there has been much discussion about the benefits distributed version control can offer your development process. Recently, distributed tools have matured to the point where there are now very few downsides remaining. While some of the advantages may not be compelling initially, having the extra flexibility that distributed tools provide at your disposal makes a lot of sense in the long run. By the end of this article, you should know enough to get started with a distributed version control system and have a basic understanding of the advantages a distributed model can provide.

Much of the discussion around distributed version control has focused on a central

server no longer being required. While this is a distinguishing feature, and vital to some groups of developers, the real value is that it allows groups of developers to implement nearly any workflow that they choose. Anything from a traditional centralised model to two developers working together in a cafe with no network other than a local wireless connection becomes possible.

These possibilities, which enable new and different ways for people to work, is what is truly exciting about distributed version control. In fact, the end of the article provides a how-to guide for this exact ad hoc coffee shop workflow. It can work for writers, school teachers, or hard-core Linux® kernel developers.

What is distributed version control?

A DVCS, or Distributed Version Control System, is a way to manage versions of files that doesn't require a centralized server, but can also make use of a centralized server. Changes can be merged to any other user of the DVCS, and as such it allows for a very flexible workflow.

The two main advantages of DVCS are that it is more flexible than centralized version control, as it allows a traditional (centralized) workflow in addition to more esoteric workflows, and it very fast. It is so much quicker than centralized servers because most operations take place locally on the client, instead of requiring a network operation.

What is a hook?

A hook is a way to programatically "hook" into actions during the life cycle of version control operations. With hooks you are able to automatically send e-mail when code is checked into a repository. Alternately, you check that a user has fulfilled a set of requirements, such as including a test file, before the code is checked in. Another method of extending the functionality of a version control system is to write plugins. In the [Resources](#) section, you will find links to articles that describe the plugin systems for Bazaar, Git, and Mercurial if you are curious about writing your own.

Why are the main differences between DVCS and centralized version control systems?

There are three core differences between DVCS over centralized version control systems. The first is offline work with local commits, which is integral to how DVCS operates. This is fundamentally different than centralized version control, which requires that all operations occur through a connection to a centralized server. This flexibility allows developers to work on an airplane, and make commit after commit, just as easily as if they were working in the office.

The second difference is that DVCS are more flexible than centralized systems, as they allow many different types of workflows, from a classic centralized workflow, to a purely ad hoc, to a mixture of ad hoc and centralized. This flexibility allows development to take place through e-mail, peer-to-peer, and any manner that a development team could think up.

The third difference is that DVCS are much quicker than centralized version control systems because most operations occur on the client and are fairly instantaneous. In addition, when a push (communication to another node) is required, it is also quicker because both clients have the full metadata on their machines. This speed difference is quite substantial, and can differ, roughly, between 3-10 times quicker for a DVCS than Subversion, depending on whether the repository is a local repository or a network repository.

Distributed version control workflow

Because DVCS's are so flexible, there are a vast number of potential workflows, but for the sake of space, this article only talks about two. First, one of the most common workflows is a Partner workflow. With a Partner workflow, a developer starts a project, then makes a branch. They then merge changes back and forth between branches that another developer is working on, and merge those changes in each time.

A second common workflow is to use a centralized server with local commits. In the workflow, a developer works much like they would with a centralized subversion repository, except they commit locally, and then push a final change to the centralized server. There are many variations to this, including intermixing the Partner workflow with this workflow. The main thing to take away is that there are many ways to work, and with DVCS, you have the flexibility to choose which way works best for you.

Quickstart guide

One of the best ways to actually understand a new technology is to work with it. In this next section, you can walk through the article while it runs through common operations in Mercurial, Bazaar, and Git:

- **Mercurial**
 - **Install:**`sudo easy_install-2.5 mercurial`
 - **Make project directory:**`mkdir hgrepo; cd hgrepo`
 - **Initialize project:**`hg init`
 - **Add a file:**`touch foo.txt; hg add foo.txt`

- **Commit:**`hg commit -m "added foo.txt" commit`
- **Grab Shared Repository:**`hg clone ssh://example.com//projects/hgrepo`
- **Commit Changes Locally:**`hg -ci -m "adding a change"`
- **Push Changes to Server:**`hg push`
- **See Pending updates as patches:**`hg incoming -p`
- **Download updates from Server:**`hg pull`
- **Apply changes:**`hg update`
- **Merge conflicts:**`hg merge`
- **Merge two unrelated remote repositories:**`hg pull -f ssh://example2.com//projects/hgrepo`
- **Bazaar**
 - **Install:**`sudo easy_install-2.5 bZR`
 - **Make project directory:**`mkdir bzrrepo; cd bzrrepo`
 - **Initialize project:**`bZR init`
 - **Add a file:**`touch foo.txt; bZR add foo.txt`
 - **Commit:**`bZR commit -m "added foo.txt" commit`
 - **Grab Shared Repository:**`bZR branch bZR+ssh://example.com/projects/gitrepo`
 - **Commit Changes Locally:**`bZR -ci -m "adding a change"`
 - **Push Changes to Server:**`bZR push`
 - **Download updates from Server:**`bZR pull`
 - **Apply changes:**`bZR update`
 - **Merge conflicts:**`bZR merge`
- **Git**
 - **Install: Download the latest tar file**
`http://kernel.org/pub/software/scm/`
 - **Make project directory:**`mkdir gitrepo; cd gitrepo`
 - **Initialize project:**`git init`
 - **Add a file:**`touch foo.txt; git add foo.txt`

- **Commit:**`git commit -m "added foo.txt" commit`
- **Grab Shared Repository:**`git clone ssh://example.com/projects/bzrrepo`
- **Commit Changes Locally:**`bzr -ci -m "adding a change" commit`
- **Push Changes to Server:**`bzr push`
- **Download updates from Server:** `bzr pull`
- **Apply changes:**`bzr update`
- **Merge conflicts:**`bzr merge`

Conversion tools and integration with subversion

All three DVCS's have the ability to easily convert existing subversion repositories into their respective formats, and to even convert between one DVCS easily into another. This allows someone to experiment with DVCS or from one DVCS to another.

For example, with Mercurial, you can use the tools `hgimportsvn` and `hgpullsvn` to talk to an existing subversion repository and create a new hg repository with history. Another tool, `tailor`, is a general-purpose repository-to-repository tool.

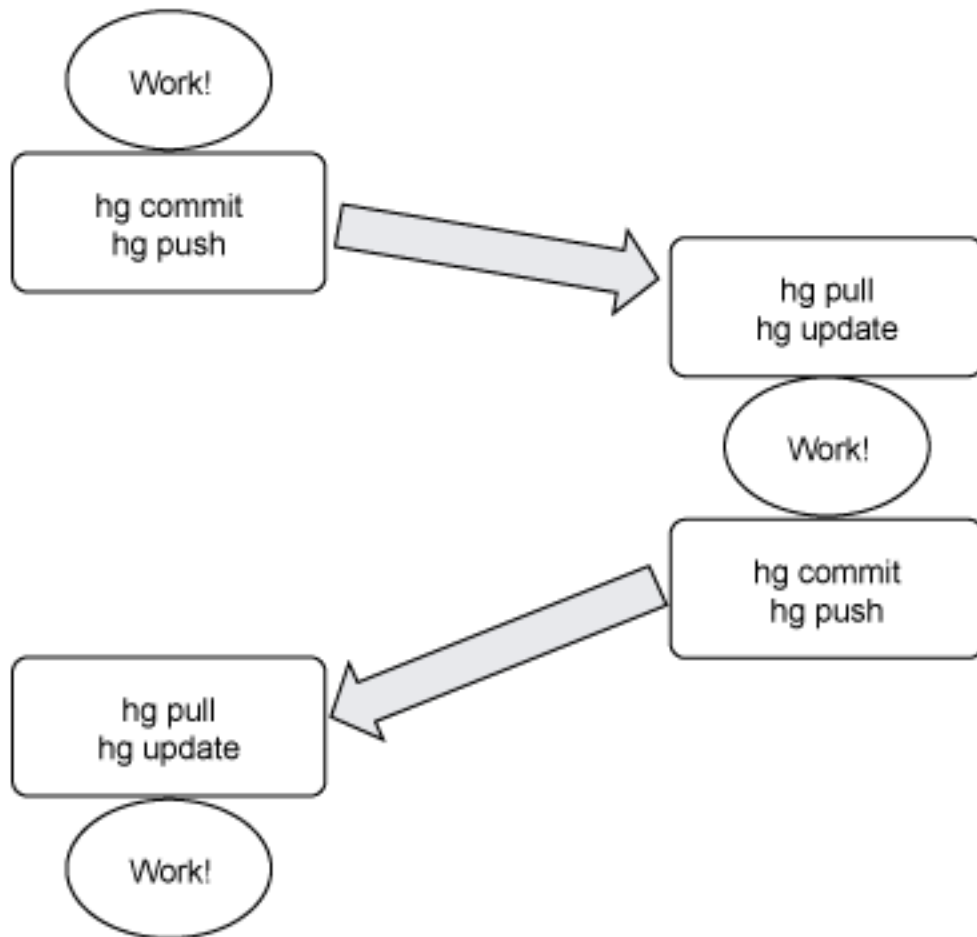
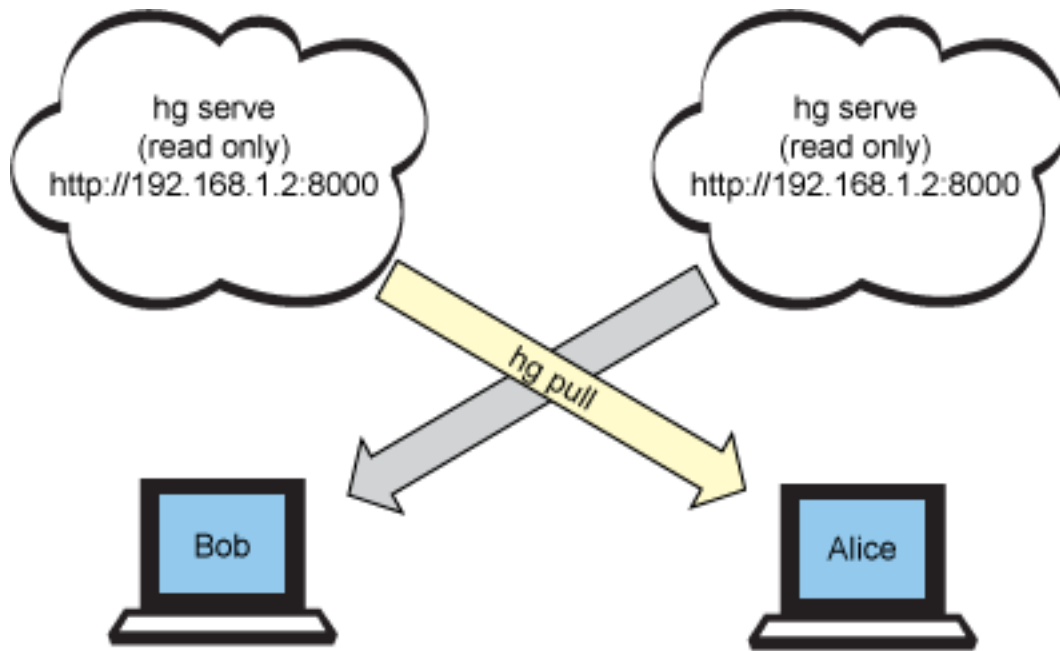
One other interesting option to experiment with is to integrate a DVCS to work with an existing subversion repository. It is beyond the scope of this article to get into too much detail about this, but the [Resources](#) section has some links to explore some of the tools for bidirectional operation between a single subversion branch and Git, Bazaar, and Mercurial.

Third-party hosting options

If you are an open source or commercial developer that doesn't want, or need, to host your own centralized "hub," there are some popular choices for hosting your project with Git, Bazaar, or Mercurial. For Mercurial, a popular free and paid hosting site is Bitbucket. For Git, there is something called Github, and for Bazaar, there is Launchpad, which is sponsored by Canonical.

Hacker's tip: "Coffee shop" Mercurial workflow

Figure 1. Coffee shop workflow



If you want to enable a secure, ad-hoc version control workflow, for instance, two guys on a wireless network in a coffee shop, then try out this hack.

Person one creates a repository using Mercurial:

```
mkdir /tmp/myhgrepo
cd /tmp/myhgrepo
hg init
```

He then shares it out over the Web as a read-only share:

```
hg serve
```

The second user then clones this repository using this command:

```
hg clone http://example.com:8000
```

Note: This is the IP or local hostname of the other machine. on OS X, it is available using Bonjour at *your-machine-name.local*.

The second user then makes the changes he needs to make, then he serves out his repository as a read-only HTTP share:

```
http://example.com:8000
```

The first user then does an hg pull at the second person's clone of his repository:

```
hg clone http://example2.com:8000
```

This workflow can go back and forth, yet each developer is secure in knowing that that he only updates his local filesystem when he wants by doing a pull from each other's repository. And that is the coffee shop workflow.

Conclusion

This article described the value distributed version control can offer you and some of the differences between the three main options, Git, Mercurial and Bazaar. If you are new to version control, continue to experiment and learn about hooks and plugins and the power that they can offer.

If you are an old hand, then you should be up to speed with each of the tools so you

can use them to their best advantage. Review the links in the [Resource](#) section, which contain more detailed information on the specifics of distributed version control and how some people are using it.

Resources

Learn

- [DVCS overview](#).
- [Tailor](#) is an "anything" to "anything" version control migration tool.
- [Mercurial](#) comes with a utility to mirror changes back and forth between svn.
- [Bazaar](#) comes with a variety of tools which do migrations from svn.
- [Git](#) has a couple of tools available and [github](#), a popular Git hosting provider will do it for you.
- [The AIX and UNIX developerWorks zone](#) provides a wealth of information relating to all aspects of AIX systems administration.
- [Open source](#): Visit the developerWorks Open source zone for extensive how-to information, tools, and project updates to help you develop with open source technologies, and use them with IBM products.
- [developerWorks technical events and webcasts](#): Stay current with developerWorks technical events and webcasts.
- [Podcasts](#): Tune in and catch up with IBM technical experts.

Discuss

- Participate in the AIX and UNIX forums:
 - [AIX Forum](#)
 - [AIX Forum for developers](#)
 - [Cluster Systems Management](#)
 - [IBM Support Assistant Forum](#)
 - [Performance Tools Forum](#)
 - [Virtualization Forum](#)
 - More [AIX and UNIX Forums](#)

About the authors

Noah Gift

[Noah Gift](#) is the co-author of "[Python For UNIX and Linux System Administration](#)" by O'Reilly, and is also working on "[Google App Engine In Action](#)" for Manning. He is an

author, speaker, consultant, and community leader, writing for publications such as IBM Developerworks, [Red Hat Magazine](#), [O'Reilly](#), and [MacTech](#). His consulting company's website is <http://www.giftcs.com>, and much of his writing can be found at <http://noahgift.com>. You can also follow [Noah on Twitter](#).

He has a Master's degree in CIS from Cal State Los Angeles, B.S. in Nutritional Science from Cal Poly San Luis Obispo, is an Apple and LPI certified SysAdmin, and has worked at companies such as Caltech, Disney Feature Animation, Sony Imageworks, and Turner Studios. He is currently working at [Weta Digital](#) in New Zealand. In his free time he enjoys spending time with his wife Leah, and their son Liam, composing for the piano, running marathons, and exercising religiously.

Adam Shand

Adam has been working as a systems administrator and team lead since the early nineties, when he founded one of the first ISPs in New Zealand. In 1997 he moved to America and spent eight years working as a systems administrator and team lead for a variety of dot-com's, including one of the largest ISPs in the world. In 2000, he founded a non-profit organisation dedicated to providing free wireless Internet by partnering with local communities. Most recently, he has moved back home to New Zealand, where he has been running the infrastructure team for Peter Jackson's visual effects company, Weta Digital.