Workflow With Git

Kevin W. Gisi

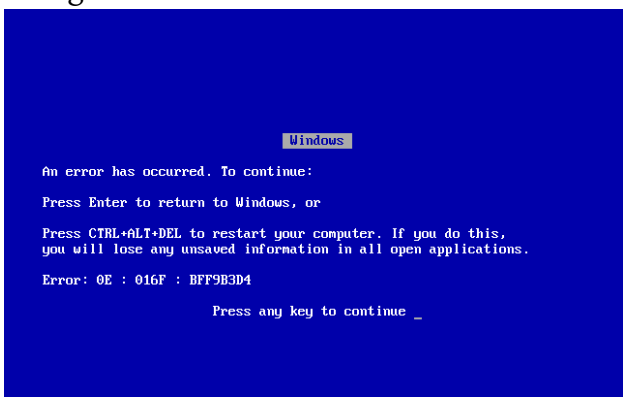
ECRuby Meeting—April 8, 2010

Summary

Regardless of what language you prefer, version-control is an essential tool for getting things done - especially when collaborating with others. Git, an open-source tool, has quickly become one of the most widely-used versioning systems, mainly thanks to the ability to branch and merge with relative ease. We'll take a look at how you can begin to integrate Git into your current workflow.

Remember Windows 95?

Regardless of your choice in programming language, computer errors are bound to happen. This is precisely why we need to have a way to back up our files. However, there's also an immense benefit to having a way of storing changes in files over time.



Archives

One way to manage this need is to simply take a snapshot of the project at various times, and store these on some backup device. A primitive method, it is nonetheless still often used. Individuals can simply zip, or tar up their project directories, throw a version number on it, and lock it away. This mechanism starts to break down with multiple collaborators, because it becomes nearly impossible to determine what version you're working with. It's also terribly inefficient, as far as data storage goes.

- Compressed files
- Version stamps: rel_1.0.3.2554beta.tar.gz
- Locked away
- Becomes challenging with multiple contributors

Diff

A space-saving alternative to storing a new version for each file, is to simply store the changes. The GNU "diff" tool provides a means of recording the differences between two files, and even write a new file that simply holds the changes. This new file can then be sent to collaborators, who can apply it to their current version. The issue of determining order of diffs is still an issue, but at the very least, storage is now reduced to maintaining the *changes* in a file, rather than the entire file itself.

```
*** /path/to/original
''timestamp''
--- /path/to/new
''timestamp''
***************
*** 1,3 ****
--- 1,9 ----
+ This is an important
+ notice! It should
+ therefore be located at
+ the beginning of this
+ document!
...
```

Primitive Version Control

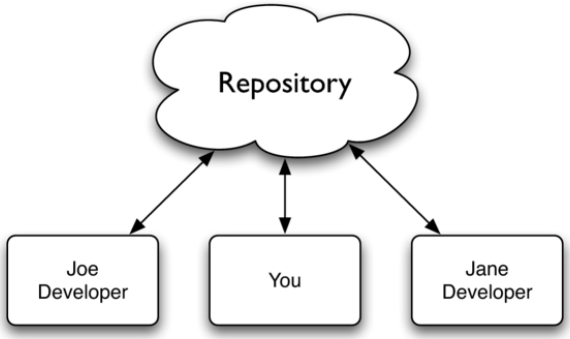
In order to automate the process, there are a few things that would be necessary in software designed to manage versions.
Goals of Versioning Software

- Track changes over time
- Handle human concurrency issues
- Assist in merge conflicts Examples
- BitKeeper, CVS, Subversion

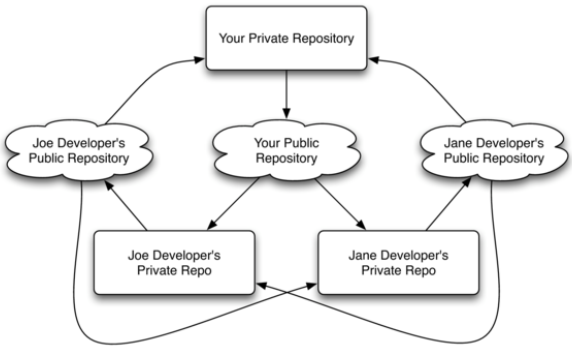
Beginnings of Git

- Created by Linus Torvalds
- Tried to avoid conventional practices (CVS,BitKeeper)
- 1.0 release on December 21, 2005
- Used to Manage Linux kernel

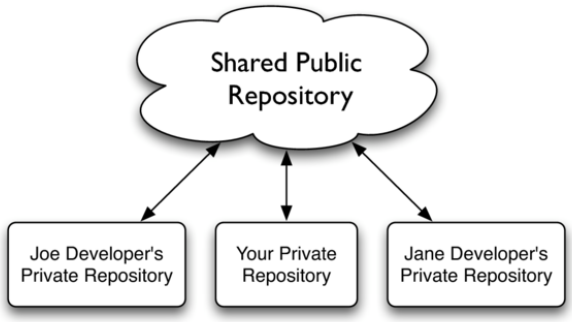
Centralized Versioning



Decentralized Versioning



Git in Practice



Let's Create a Repository!

```
:$> ls
about.html  contact.html
index.html  style.css

:$> git init
Initialized empty Git
repository in /home/gisikw/
project
```

Adding initial files

```
:$> git add .

:$> git commit -m "Initial
Commit"
Created initial commit ed3ec5b:
Initial commit
 0 files changed, 0
insertions(+), 0 deletions(-)
 create mode 100644 about.html
 create mode 100644 contact.html
 create mode 100644 index.html
 create mode 100644 style.css
```

Git Add

Git add doesn't do what you think it does!

```
:$> echo "<p>Hello, world</p>"
> index.html
:$> git commit -m "Added
initial text to index page"
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to
update what will be committed)
#
#       modified:   index.html
#
no changes added to commit (use
"git add" and/or "git commit
-a")
```

Git Add and Commit

Automatically add changes to files that are currently tracked:

```
$:> git commit -a -m "Added
initial text to index page"
Created commit 94bc6d3: Added
initial text to index page
 1 files changed, 1
insertions(+), 0 deletions(-)
```

Ignoring files

```
:$> ls
about.html ~about.html
contact.html index.html
style.css test.html
```

.gitignore

```
~*.html
test.html
```

Git Revisions

Revision names:

- HEAD - the latest commit
- HEAD~ - the second-to-last-commit
- HEAD~2 - you get the idea

Explicit Git Revisions

```
:$> git log
commit
fa8db86872c83fa62efa420548d8afe3
Author: Kevin W. Gisi <=>
Date:   Thu Apr 8 04:35:06 2010
-0500

    Adjusted index to display
standards-complient headers

commit
c4e43f114f9c441ae20d51bf5277044d
Author: Kevin W. Gisi <=>
Date:   Thu Apr 8 04:34:26 2010
-0500

    Added help.html file to
display usage and FAQ
information
```

Reverting a Commit

```
:$> git revert HEAD
```

Create a new commit which undoes the changes most recently made.

Tagging a Release

It's release time!

```
:$> git tag 1.0
```

Show current tags

```
:$> git tag
1.0
```

Feature Branches

```
:$> git branch html5
:$> git checkout html5
Switched to branch "html5"
:$> git mv index.html start.html
:$> git commit -am "Moved
index.html to start.html"
Created commit 9c56488: Moved
index.html to start.html
 1 files changed, 0
insertions(+), 0 deletions(-)
 rename index.html =>
start.html (100%)
```

Master Stays Un-touched

```
:$> git checkout master
Switched to branch "master"
:$> ls
about.html  contact.html
help.html  index.html  style.css
```

Merging Branches

Navigate to the branch you want to work on

```
:$> git merge html5
```

Merge the other branch into the current branch.

Remote Repositories

Often, we want to push our commits to another repository. We need to register the repository first.

```
:$> git remote add origin
git@github.com:gisikw/
samply-application.git
```

We tell Git a local name to call the remote, and the url where it can be accessed.

Pushing and Pulling from Remotes

Push any new commits in our master branch to the remote origin

```
:$> git push origin master
```

Pull any new commits on the remote origin into our master branch

```
:$> git pull origin master
```

Additional Resources

The Pragmatic Programmers

Pragmatic Version Control

Using Git

Travis Swicegood

Edited by Susannah Davidson Pfalzer

- Pragmatic Version Control Using Git by Travis Swicegood
- http://help.github.com
- GitCasts

Thanks!

Kevin W. Gisi
<kevin@kevingisi.com>