Workflow With Git

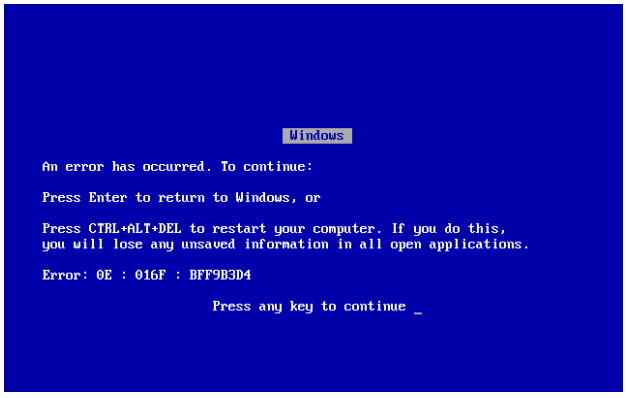
Kevin W. Gisi

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 technique for keeping track of projects it to simply create a copy of the files every so often for storage. These are typically packaged and compressed into small files, whose names include version numbers. While this works on a small level, there are two small issues: storage of redundant files, and managing version advances. If two individuals are working off of version 1.0, and they both create a new release at the same time, what versions are they?

- Compressed files
- Version stamps: rel_1.0.3.2554beta.tar.gz
- Locked away

Diff

An alternative to the file-backup approach is to use the GNU diff tool. This allows a user to view the differences between two files, and actually create a new "diff" file that represents only the changes between the two versions. This way, the diff files can be distributes and stored, rather than making space for large redundant files. However, we still run into the concurrent version issue.

```
*** /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

When developers decided it was time to automate the process, there were a few tasks that needed to be handled by any candidate versioning systems.
Challenges

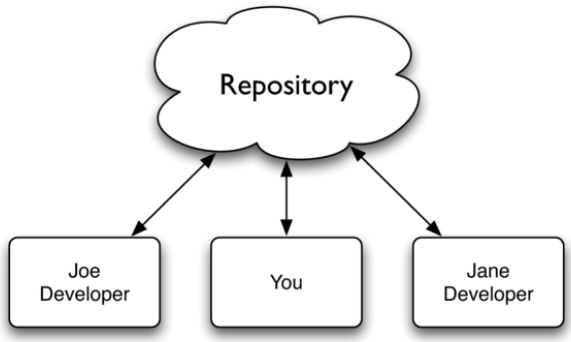
- Track changes over time
- Handle human concurrency issues
- Assist in merge conflicts Examples
- BitKeeper, CVS, Subversion

Beginnings of Git

Because of some licensing issues, the Linux gurus were forced to stop using BitKeeper to manage the source code for the Linux kernel. Linus Torvalds, the creator of the Linux kernel, set out to create his own versioning system that had a strong emphasis on handling merges, and supporting easy branching.

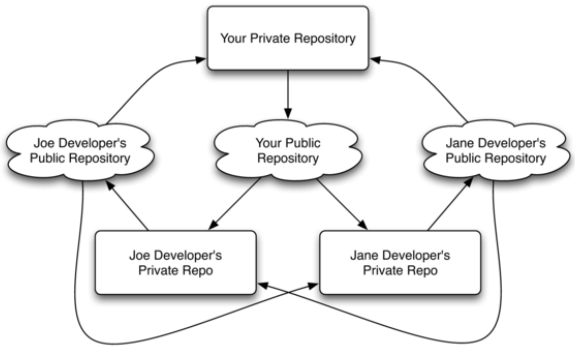
- Created by Linus Torvalds
- Tried to avoid conventional practices (CVS,BitKeeper)
- 1.0 release on December 21, 2005
- Used to Manage Linux kernel

Centralized Versioning



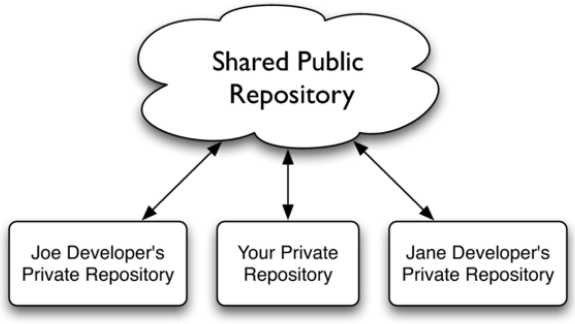
For some version control systems, the structure is centralized. This means that the code is actually hosted on a single server (or cluster of servers). Clients are able to able to pull down copies of the code, and push back changes, but any branching, merging, committing, is all done on a single versioning server. Subversion is an example of a centralized version control system.

Decentralized Versioning



Decentralized version control sytems such as Git follow a different philosophy. They, in fact, do away with clients. Instead, users are able to create local repositories, and commit, branch, merge all on their own machine. Once they are satisfied with their changes, they can push their repository's commits to another repository, but there is no requirement that there be any single repository that is considered the "master" repository. This has the added benefit of allowing users to make shorter commit cycles, and use the version control system almost like a word-processor's auto-save tool.

Git in Practice



While Git is a decentralized system, most users tend to use Git in a hybridized manner - each user has a local repository, but there is also a remote repository that acts as a single point from which all users pull updates and push changes. For many, Github is this shared repository.

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!
Git add tells Git that this file should be a part of the next commit. Unlike Subversion, once a file is in the repository, future commits will not explicitly include changes to these files. For each commit, the files must be re-added, or they will be listed as "Changed but not updated"

```
:$> echo "<p>Hello, world</p>"
> index.html
:$> 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

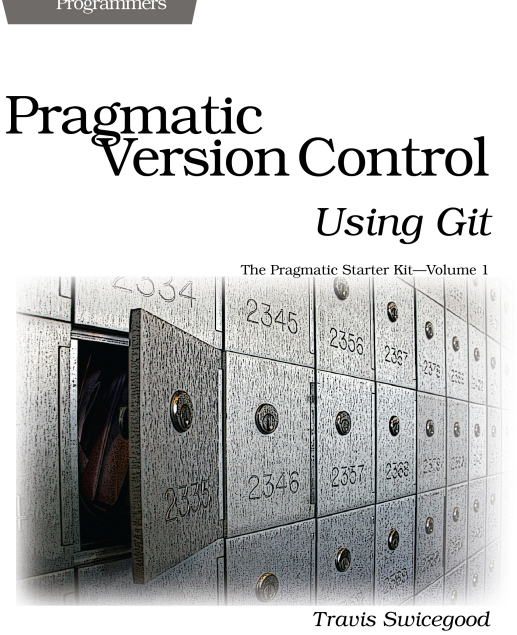
Push any new commits in our master branch to the remote origin

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

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

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

Additional Resources



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

Thanks!

Kevin W. Gisi
<kevin@kevingisi.com>

