Git Source Control: For the Rest of Us

Nolan Erck

South of Shasta Consulting

Obligatory “About Me” Stuff

- Owner, Chief Consultant at South of Shasta
  - Software Development
- ColdFusion, C++, Java, jQuery, PHP, .NET, HTML5, Android, SQL, etc
- Manager, SacInteractive User Group
- Reformed Video Game Developer (Grim Fandango, SimPark, StarWars Rogue Squadron, etc)
- Music Junkie


I'M REAL PROUD OF MY RECORD COLLECTION

Today's Agenda

- What is “source control”?
- When/why do I need it?
- How to use Git from a GUI.
- The difference in Git and GitHub.
- Next steps:
- branching, merging, reverting.
- Demos of how to do this.
- Other info.
- Questions.

Quick show of hands

(It's okay to raise your hand, I promise.)

- Who is brand new to source control?
- Who is NOT currently using any source control in your projects?
- Who uses .OLD or .BAK files as a way to “save your place” before making a change?
- Who makes ZIP files to backup the last version of your code?

Scenario 1

- You're about to make a change to your website.
- “I should save a copy of this somewhere before I break anything.”
- Windows Explorer
- AboutUs.html ---> AboutUs.BAK
- That's good, right?

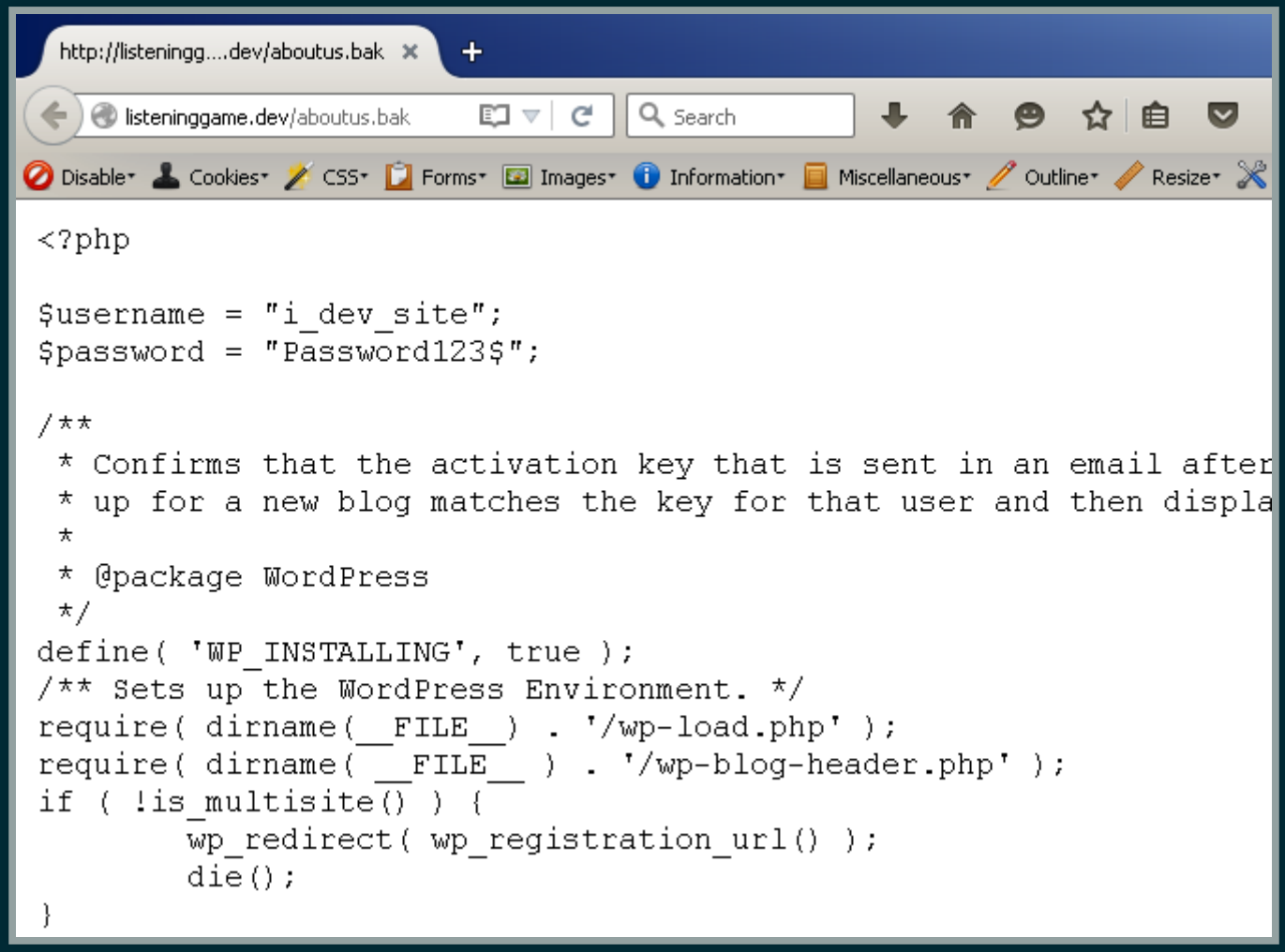
Scenario 1 (cont)

- What about when you make a second change to AboutUs.html?
- Rename the file again
- AboutUs.HTML ---> AboutUs.OLD
- Fast-forward 6 months
- Which is the “most recent backup”, BAK or OLD?
- Can't remember, have to “diff” by hand. Boo.

But wait, it gets worse!

- Do you use FTP to update the site?
- And move /wwwroot up all at once?
- AboutUs.BAK and .OLD are in now in Production.
- Http://sitename/AboutUs.OLD Is a valid URL.
- Won't be processed by ColdFusion (or Ruby, PHP, etc)
- Original source code gets sent to the browser as plain text!
- Security issue!

Don't believe me? Here's proof.




```
<?php

$username = "i_dev_site";
$password = "Password123$";

/**
 * Confirms that the activation key that is sent in an email after
 * up for a new blog matches the key for that user and then displa
 *
 * @package WordPress
 */
define( 'WP_INSTALLING', true );
/** Sets up the WordPress Environment. */
require( dirname( __FILE__ ) . '/wp-load.php' );
require( dirname( __FILE__ ) . '/wp-blog-header.php' );
if ( !is_multisite() ) {
        wp_redirect( wp_registration_url() );
        die();
}
```

Scenario 2

- George and I are splitting a project.
- I'm working in contact.html
- George emails me a copy of contact.html with changes of his own.
- Which lines changed? Did he change them before/after my edits?
- I have to manually “diff” the 2 files. Boo.

Ever had this phone call?

Client: “You MUST have made a change to the site today! Everything was working FINE yesterday and now there is a bug.
What did YOU break?!”

Or this one?

Client: “Remember that huge change you made to the site last week? Well I need you to undo it. Today. Right now. Can't you just CLICK something and make that happen?”

There's gotta be a better way!

- There is.
- Source Control.
- It doesn't have to be Git.
- For today's talk we'll focus on Git with a GUI client.
- (Others: Subversion, CVS, Team Foundation Server, Visual SourceSafe, PerfForce, Mercurial.)

Source control...

- Acts as a “librarian” and “hall monitor” for your site assets (code, images, config files, etc).
- Is software that tracks all the changes made to files on a project (by me, Ringo, you, that guy, everyone).
- Let's us share files safely.
- No more emailing files!
- Let's us change files safely too.
- No more BAK and OLD! No more ZIPs!
- Is platform agnostic.
- Works for Windows, Mac, Linux development.
- For any kind of code you write: CF, C++, Java, iPhone, Ruby, COBOL, PHP, QBASIC, InstallShield, whatever.

Two pieces to the software

- Client
- Server

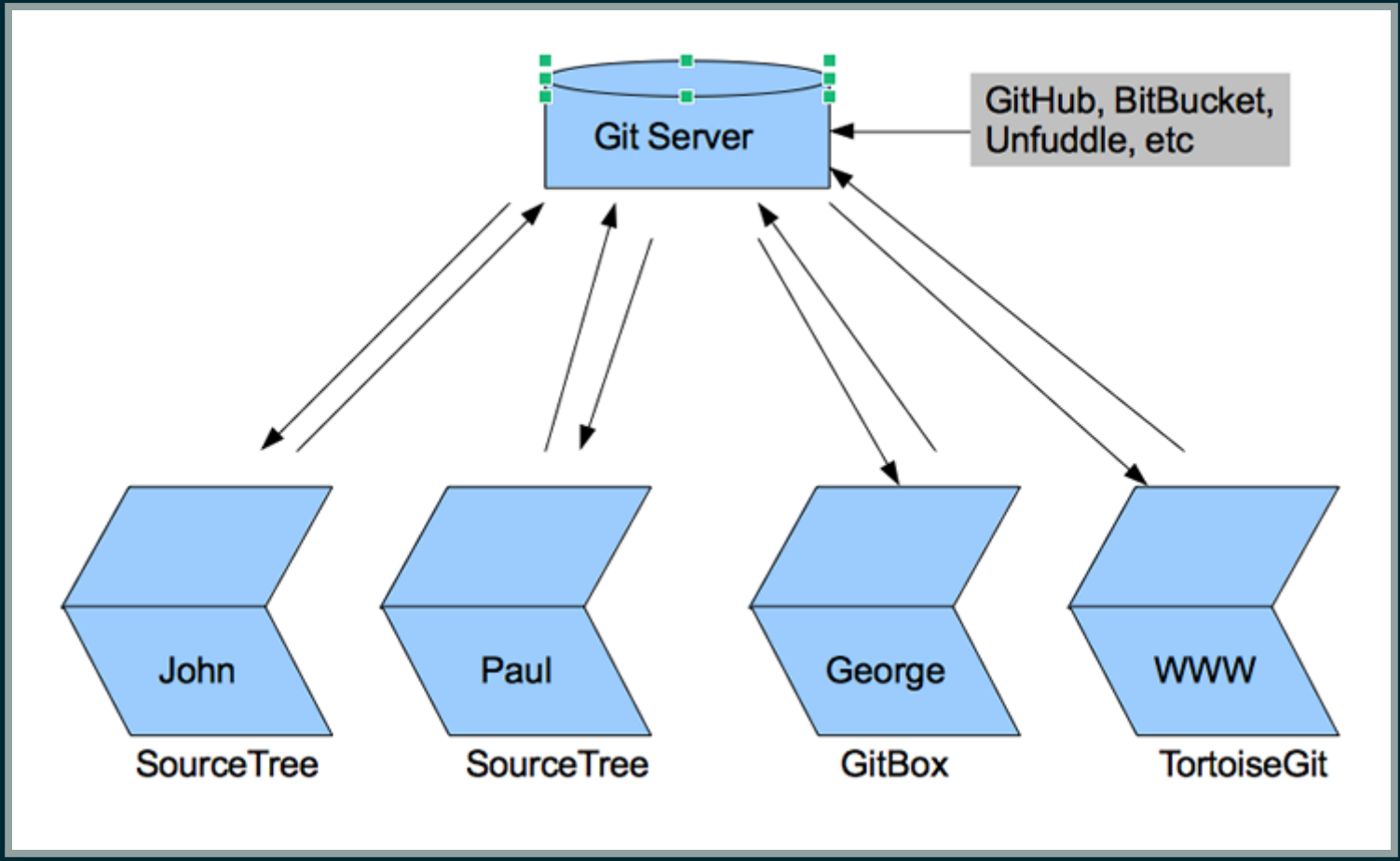
Client

- Desktop app, lives on your Dev box.
- OSX: Tower, GitBox, SourceTree, SmartGit, etc.
- Windows: TortoiseGit, SmartGit, SourceTree, etc.
- Can use any one you want.
- Can mix/match among team members.
- Free to $50-ish.
- Can change at any time, don't have to be “married” to 1 particular client.

Server

- So all team members can share files.
- A central place to do the “librarian” work.
- Keep track of who changes what, when, etc.
- GitHub, Unfuddle, BeanStalk, host your own, etc.
- Can use any one you want (doesn't have to be GitHub, that's just 1 of the vendors available).
- Git != GitHub
- Git = the protocol under the hood (e.g. CVS, Subversion)
- GitHub = a popular choice for your Git server

Server + Clients



(Poor Ringo...always left out.)

Adding a file to a Git Repo

- AKA the “git add” command.
- “I want Git to watch this file for changes”.
- Do it once, after you create the file.
- Just because you make a file, that doesn't mean Git watches it for changes.
- You must “add” the file to the Git repo.
- Let's look at an example....

Committing a file

- “git commit”
- Git: take a snapshot of what this file looks like right now, for later reference.
- Do this instead of making .BAK files.
- The “committed” version of the file is stored locally.
- Let's look at an example...

Pushing a file

- “git push”
- Git: “take the snapshot you made and copy it up to the Git server.”
- Do this whenever the file(s) is ready to be shared with your team, or ready for inclusion in the latest build, etc.
- Let's look at an example...

Reverting a file

- AKA the “git revert” command.
- “Ack! I just made a huge mistake in this file. How do I get it back to what it used to look like?”
- Do this any time you messed up and need to undo a change.
- Under the hood, Git protocol is the same regardless of GUI client. The actual “thing” you press will vary between GUI clients.
- Here's an example in SourceTree...

Branching

- “git branch”
- “I'm about to make a change to the site. I want a safe way to add my changes, but undo them quickly if something goes wrong.”
- Do this: Always.
- If you're making a change to the site, put it in its own branch.
- Yes, even for that tiny change that's only in 1 file.
- Example time again...

Merging Branches

- “git merge”
- “My new feature has been blessed by QA as bug free and is ready to be put on the Live server”.
- (Or combined with other features for the next build.)
- Let's look at an example...

Common Practices

- Make a branch called “Master” for what the Production server should look like.
- Don't actually write code in this branch.
- Write in other branches, then merge them into Master.
- Anytime I “pull from Master”, I get an exact copy of what's on the Live server.
- Make a branch called “QA”. As bugs are fixed, merge them into QA.
- When everything in QA looks correct, merge it into “master”.
- Don't be afraid to make “temporary branches” to try out new ideas. No harm in making and deleting new branches.

Next things to learn

- Merge Conflicts
- Reverting
- Rebasing
- Cherry Picking
- Various other weird features (you probably won't need unless something goes horribly wrong)

Other Resources

- Pro Git Book
  - FREE!
  - http://git-scm.com/book
  - Good info, examples are command-line.
- Tim Cunningham's Git presos and blog entries (cfmumbojumbo.com)
  - CF Hour Podcast, episode 118
- Lots of (short) tutorial videos on YouTube
- My Git Class at CF Camp 2016!
- Google, DuckDuckGo, etc


Questions? Comments?
• southofshasta.com
• nolan@southofshasta.com
• Twitter: @southofshasta
• Github: nolanerck
Thanks!