Git at SmartThings

Shaun Jurgemeyer

What is Version Control?

A system for managing changes to a codebase.

How does it do this?

- Store changes in a 'repository'
- Give each change (called a 'commit') a unique identifier.
- Allow labelling and access to each of these commits
- Allow sharing of the repository and commits

What is Git?

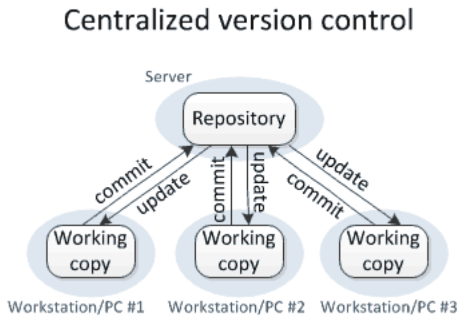
- One of many Version Control Systems out there.
- Others include Mercurial, Subversion, CVS
- Distributed Version Control vs Client Server

Distributed Version Control

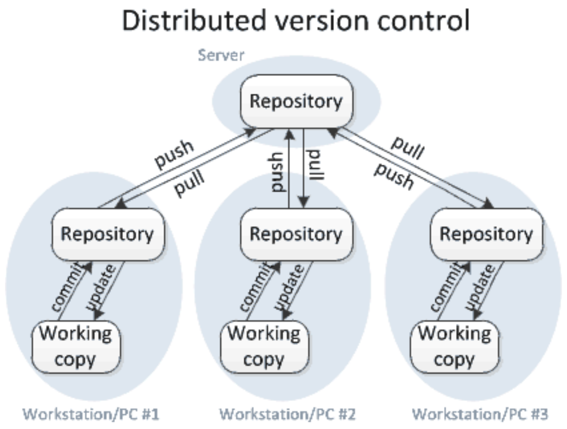
Distributed version control systems enable each user to have a complete repository.

- Does not require connection to network
- Common actions are much quicker since it does not need to be coordinated with a central repository.
- Communcaiton with central repository is only necessary when sharing changes.

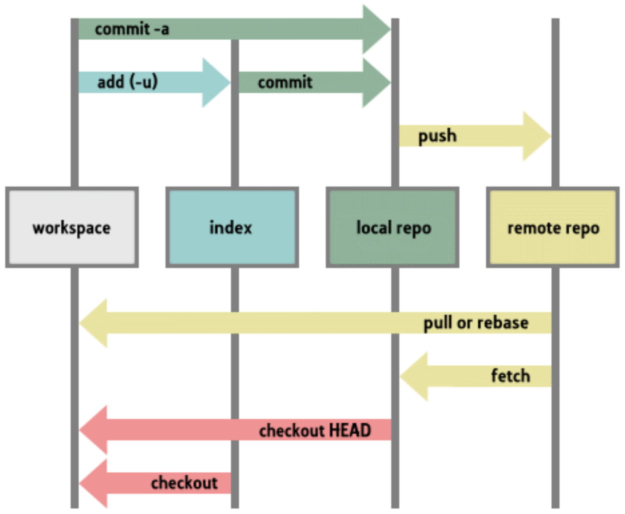
Centralized Version Control



Distributed Version Control



Commit creation



Git Index

- The git index is where you place files you want committed to the git repository.
- Before you commit files to the repository, you need to first place the files in the git index.
- Can be referred to as the index, staging area, or cache

Referencing commits

- By sha
- by short sha
- Relative to HEAD
- Branch name
- Tag name

Branches and Tags

Branches and Tags are both pointers to git commits

Branches typically move frequently where tags are usually fixed

Branches

Typically used to separate streams of work

Tags

Typically used for marking releases

Visualizations of commits

http://gitup.co/

Diffing commits

- diff between current code and last commit

- diff specific version

- diff tool can be configured in your local .gitconfig

Configuration

Global configuration

~/.gitconfig

Aliases

Ignoring files

Add patterns to the git ignore file to have git ignore the files altogether

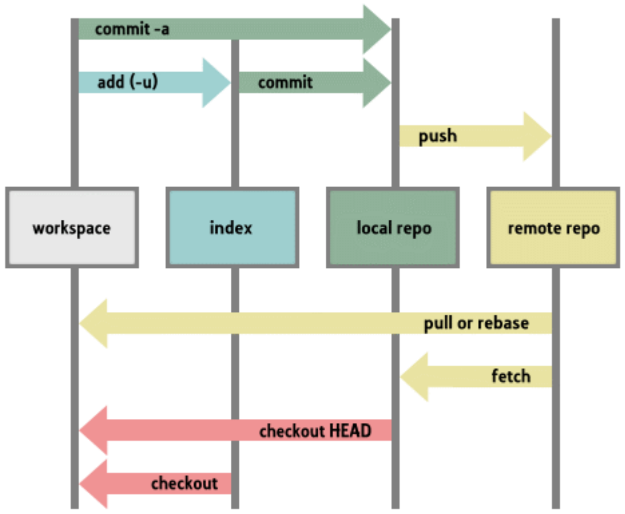
/.gitignore

```
```

Remote repositories

- Your commits are stored in your repository
- A remote repository is a copy of the entire repository stored elsewhere.
- You share your commits with others by 'pushing' your commits to the remote repository
- You get other people's commits, by 'fetching' commits from a remote repository

Repository interaction



Cloning a remote repository

- Copy of the entire repository is created on your local machine
- A remote repository is automatically configured. It is named 'origin'

Remote branches

- Branch pointers are also set on the remote repository
- They will not be changed unless you specifically 'push' that branch.
- You can create or modify branches locally without impacting the remote repository.

Getting changes from remote branches

Update your remote branch pointer

Add the changes from the remote branch to your local branch

Do both actions in one operation

Fast forward vs merge

Typically, commits have one parent. When you don't have any changes that aren't in the branch to be merged, git can just move the pointer

Otherwise, git must create a merge commit. Merge commits have 2 parents. This can make history more difficult to understand.

Rebasing

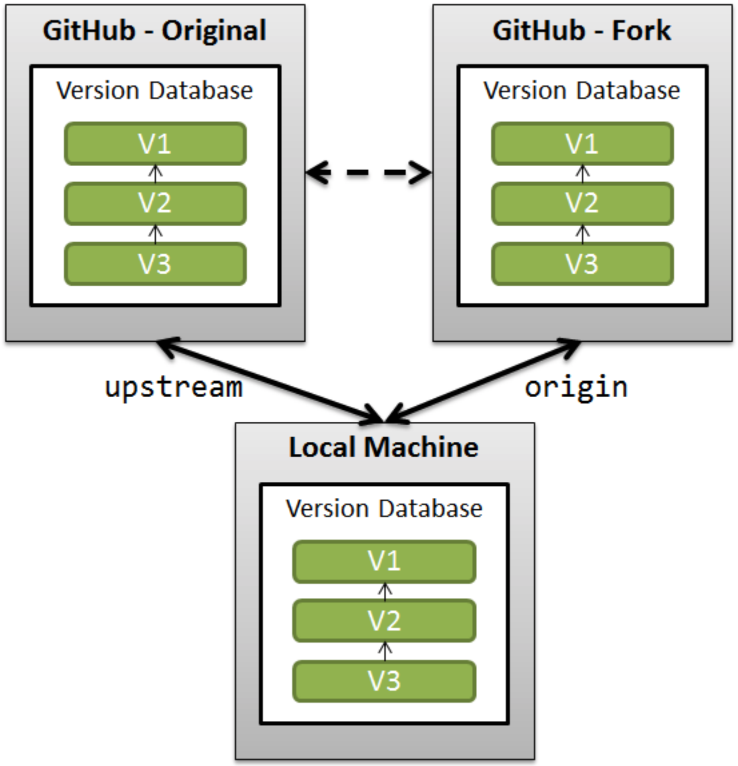
Rebasing allows you to avoid merge commits.

Merge conflicts

- When git can't figure out how to merge two sets of changes, you will get a merge conflict.
- In order to resolve this, you will have to manually decide how to resolve the conflicting changes.

Forking

- Copies another repository on github to your own copy in the cloud
- Allows you to push changes to a remote repository of your own without affecting the main project
- Makes pull requests easier to manipulate.


GitHub - Original
Version Database
V1
V2
V3
GitHub - Fork
Version Database
V1
V2
V3
upstream
origin
Local Machine
Version Database
V1
V2
V3

Suggested setup for repos

- Main repo e.g. PhysicalGraph/data-management
- Fork the main repo to your user e.g. sjurgemeyer/data-management
- Clone from your fork

- Add a remote repository

- Whenever you need to update.

- Whenever you need to create a PR

Pull Requests

- Allows other developers to review and comment on proposed changes
- Best practice is to push a branch to your fork and create a pull request to the main repo, master branch

Reviewing PRs

- Look for common coding errors, or problems you've run into in the past.
- Try to understand what the intent of the PR is and point out if anything could've been missed
- Point out confusing areas and ask for comments if necessary.
- Ask questions!
- If everything looks good to you and you feel comfortable merging the code, give a +1 or :thumbsup in the comments
- If you have looked at the PR, but don't feel confident in your knowledge of the code added or modified, give it an :eyes emoji.

SmartThings PR process

- Create a PR from your fork to upstream
- Wait for feedback! If you don't get any, solicit feedback directly or with @mentions.
- Ensure that the PR build is passing
- Update the PR as necessary.

Pull Request DO's and DON'Ts

- DON'T merge PRs without getting feedback - That's the whole point.
- DON'T merge PRs without a passing build. PR builds save other developers time.
- DON'T push straight to the main repo (upstream)
- DO review PRs on projects you are working on. Even if you don't have feedback, you can learn about what changes are being made.

SmartThings merging process

- When at least one other developer has given positive feedback (+1 or thumbsup), the PR is ready to merge.
- Communicate with the team in charge of the project in question about when it's good to merge.

Specific to Data Management

- upstream/master contains the most up to date code and is deployed to development.
- upstream/release contains the code that will be released to prod next and is currently in staging.
- Code pushed to these branches is automatically built and deployed to its respective environment
- Code that 'must go out in the next release' should be targeted at the release branch, but this should be rare.

Manipulating commits

- amend
- rebase -i
- reset
- cherry-pick

Force pushing

- If you attempt to push to a remote repository, but the commit cannot be fast-forwarded, you will get an error
- You can get around this by force pushing.
- DON'T FORCE PUSH TO upstream!!!
- Even if force pushing to your fork, do so with care

Stash

Allows storing the changes of the working directory and restoring the current branch Changes can be accessed and reapplied later

Reflog

- Shows history of commands that changed a reference (by default HEAD)
- Helps you get out of the worst situations

Resources

- Pro Git http://git-scm.com/doc
- Hub https://hub.github.com/
- SCM Breeze https://github.com/ndbroadbent/scm_breeze
- Git Up http://gitup.co
- Trailer.app http://ptsochantaris.github.io/trailer/
- Git Tower http://www.git-tower.com/