GIT

Lesson 03 – Repositories and Branching

Capgemini

Lesson Objectives

To understand …
- Git Repositories and Branching
- Git Working Tree

2

Add instructor notes here.

Repositories and Branches

Repositories:

- It is a collection of refs together with an object database containing all objects which are reachable from the refs, possibly accompanied by meta data from one or more porcelains. A repository can share an object database with other repositories via alternates mechanism.

What to store in repositories?

- Anything, however any sort of editable files are preferred.

How to get GIT repository:

- $ git clone git://git.kernel.org/pub/scm/git/git.git

    It does approx. 225 MB download.

Ref : A 40-byte hex representation of a SHA1 or a name that denotes a particular object. They may be stored in a file under $GIT_DIR/refs/ directory, or in the $GIT_DIR/packed-refs file

Reachable: All of the ancestors of a given commit are said to be "reachable" from that commit. More generally, one object is reachable from another if we can reach the one from the other by a chain that follows tags to whatever they tag, commits to their parents or trees, and trees to the trees or blobs that they contain.

Porcelain: Cute name for programs and program suites depending on core git, presenting a high level access to core git. Porcelains expose more of a SCM interface than the plumbing

Alternate object database: Via the alternates mechanism, a repository can inherit part of its object database from another object database, which is called "alternate

Git repositories

Creating repositories :
- at default location
  - git init
- at particular location
  - git init c:/testGIT
- Bare repository
  - git init --bare

A bare repository is normally an appropriately named directory with a .git suffix that does not have a locally checked-out copy of any of the files under revision control. That is, all of the git administrative and control files that would normally be present in the hidden .git sub-directory are directly present in the repository.git directory instead, and no other files are present and checked out. Usually publishers of public repositories make bare repositories available.

master The default development branch. Whenever you create a git repository, a branch named "master" is created, and becomes the active branch. In most cases, this contains the local development, though that is purely by convention and is not required.

Repositories and Branches

Branches:
- A "branch" is an active line of development.
- The most recent commit on a branch is referred to as the tip of that branch.
- The tip of the branch is referenced by a branch head, which moves forward as additional development is done on the branch.
- A single git repository can track an arbitrary number of branches, but your working tree is associated with just one of them (the "current" or "checked out" branch), and HEAD points to that branch

Add instructor notes
here.

Repositories and Branches

Getting different versions of project:

- Git is best thought of as a tool for storing the history of a collection of files. It stores the history as a compressed collection of interrelated snapshots of the project's contents. In git each such version is called a commit.
- Those snapshots aren't necessarily all arranged in a single line from oldest to newest; instead, work may simultaneously proceed along parallel lines of development, called branches, which may merge and diverge.
- A single git repository can track development on multiple branches. It does this by keeping a list of heads which reference the latest commit on each branch; the git-branch(1) command shows you the list of branch heads:

- $ git branch * master
- A freshly cloned repository contains a single branch head, by default named "master", with the working directory initialized to the state of the project referred to by that branch head.

- Most projects also use tags. Tags, like heads, are references into the project's history, and can be listed using the git-tag(1) command:
- $ git tag -l

HEAD: The current branch. In more detail: Your working tree is normally derived from the state of the tree referred to by HEAD. HEAD is a reference to one of the heads in your repository, except when using a detached HEAD, in which case it directly references an arbitrary commit.

detached HEAD: Normally the HEAD stores the name of a branch. However, git also allows you to check out an arbitrary commit that isn't necessarily the tip of any particular branch. In this case HEAD is said to be "detached".

Add instructor notes here.

Understanding History

Commits:

- Every change in the history of a project is represented by a commit. The git-show(1) command shows the most recent commit on the current branch:
- $ git show
- Every commit (except the very first commit in a project) also has a parent commit which shows what happened before this commit. Following the chain of parents will eventually take you back to the beginning of the project.
- However, the commits do not form a simple list; git allows lines of development to diverge and then reconverge, and the point where two lines of development reconverge is called a "merge". The commit representing a merge can therefore have more than one parent, with each parent representing the most recent commit on one of the lines of development leading to that point.
- The best way to see how this works is using the gitk(1)command; running gitk now on a git repository and looking for merge commits will help understand how the git organizes history.
- In the following, we say that commit X is "reachable" from commit Y if commit X is an ancestor of commit Y. Equivalently, you could say that Y is a descendant of X, or that there is a chain of parents leading from commit Y to commit X.
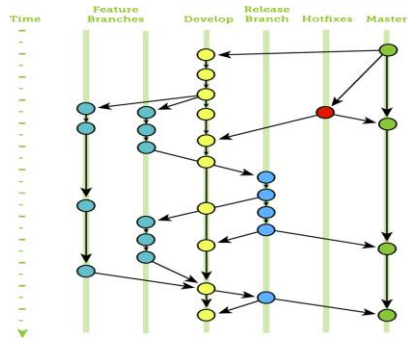
Add instructor notes here.

Understanding History

History Diagrams:
- We will sometimes represent git history using diagrams like the one below:

Add instructor notes here.

Working Trees

- The working tree is your current view into the repository. It has all the files from your project: the source code, build files, unit tests, and so on.

- Some VCSs refer to this as your working copy. People coming to Git for the first time from another VCS often have trouble separating the working tree from the repository. In a VCS such as Subversion, your repository exists "over there" on another server.

- In Git, "over there" means in the .git/ directory inside your project's directory on your local computer. This means you can look at the history of the repository and see what has changed without having to communicate with a repository on another server.

Summary

In this lesson, you have learnt
- Git Repositories and Branching
- Git Working Tree

Summary

Add the notes here.