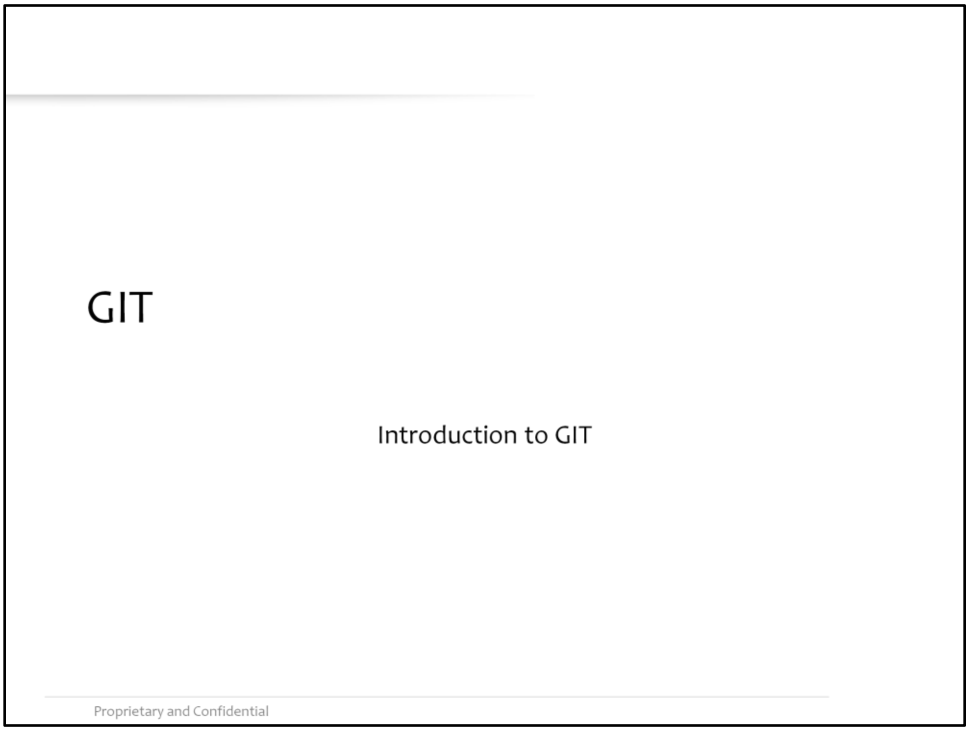
Instructor Notes:

Add instructor notes here.



GIT

Introduction to GIT

Proprietary and Confidential

Instructor Notes:

Add instructor notes here.

Lesson Objectives

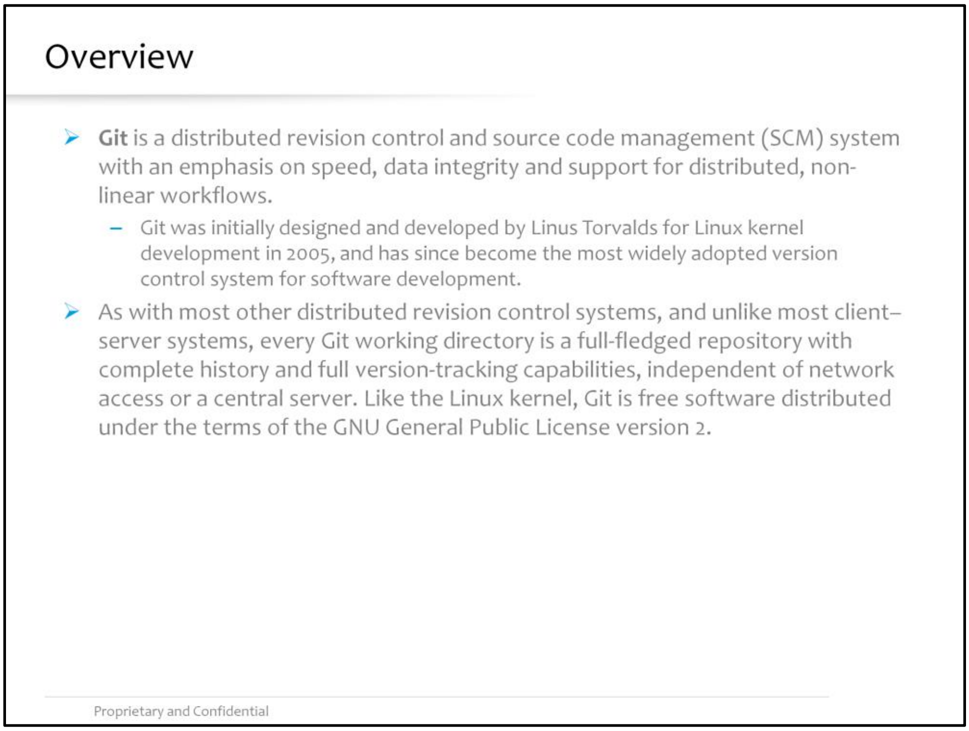
- **Overview of GIT**

Objectives

Proprietary and Confidential

Instructor Notes:

Add instructor notes here.



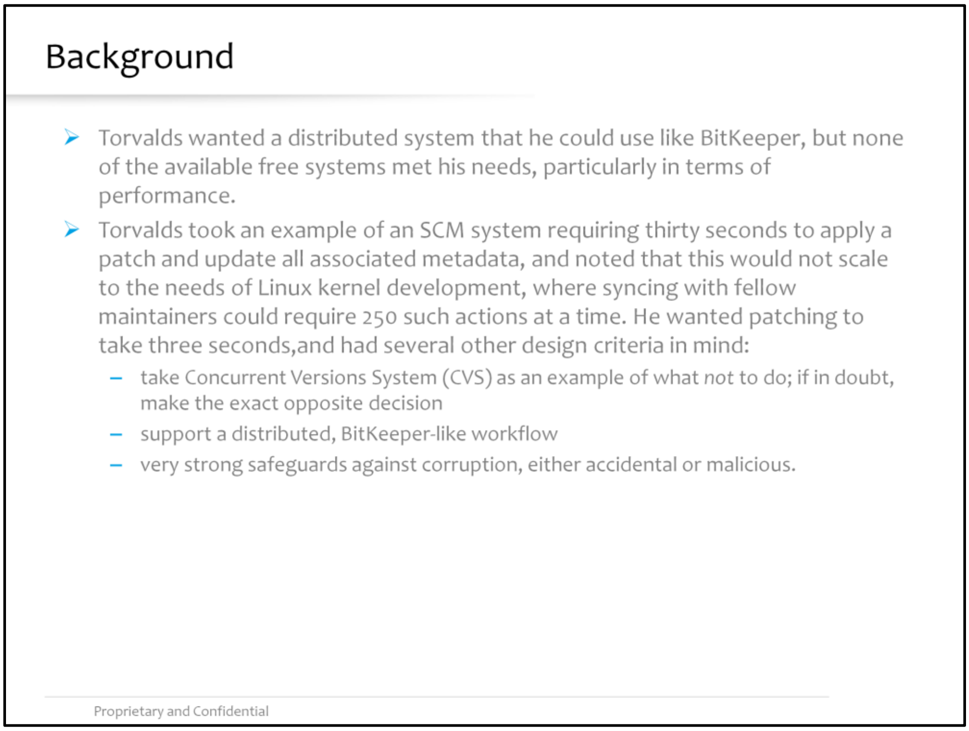
Overview

- **Git** is a distributed revision control and source code management (SCM) system with an emphasis on speed, data integrity and support for distributed, non-linear workflows.
  - Git was initially designed and developed by Linus Torvalds for Linux kernel development in 2005, and has since become the most widely adopted version control system for software development.
- As with most other distributed revision control systems, and unlike most client–server systems, every Git working directory is a full-fledged repository with complete history and full version-tracking capabilities, independent of network access or a central server. Like the Linux kernel, Git is free software distributed under the terms of the GNU General Public License version 2.

Proprietary and Confidential

Instructor Notes:

Add instructor notes here.



Background

- Torvalds wanted a distributed system that he could use like BitKeeper, but none of the available free systems met his needs, particularly in terms of performance.
- Torvalds took an example of an SCM system requiring thirty seconds to apply a patch and update all associated metadata, and noted that this would not scale to the needs of Linux kernel development, where syncing with fellow maintainers could require 250 such actions at a time. He wanted patching to take three seconds,and had several other design criteria in mind:
  - take Concurrent Versions System (CVS) as an example of what *not* to do; if in doubt, make the exact opposite decision
  - support a distributed, BitKeeper-like workflow
  - very strong safeguards against corruption, either accidental or malicious.

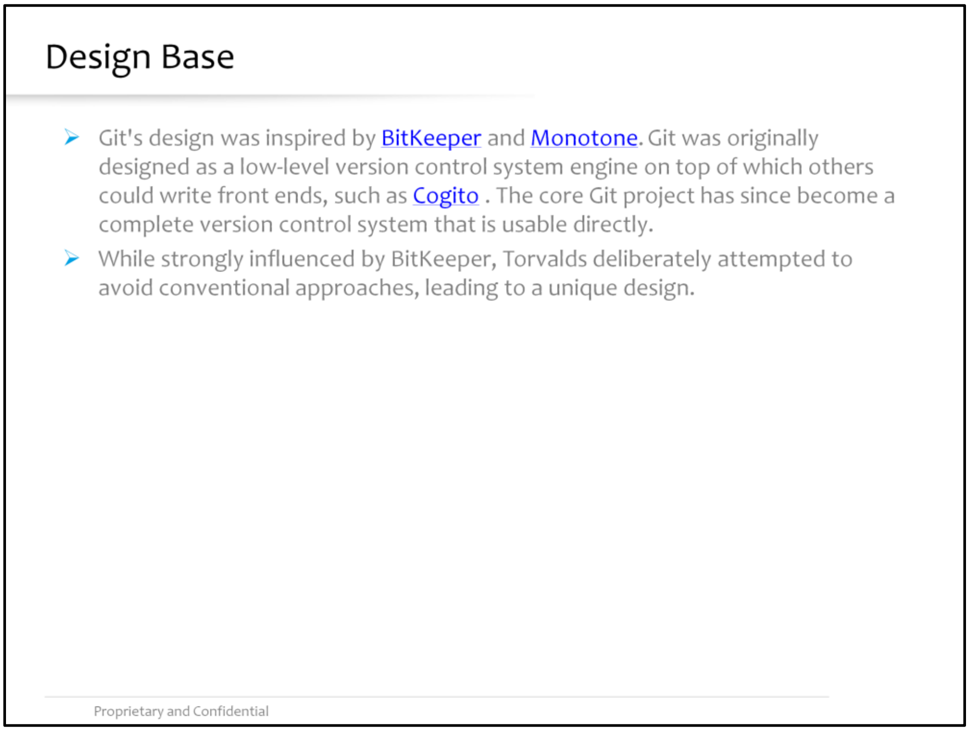
Proprietary and Confidential

Torvalds has quipped about the name *git*, which is British English slang roughly equivalent to "unpleasant person". Torvalds said: "I'm an egotistical bastard, and I name all my projects after myself.

The manual page of git describes git as "the stupid content tracker“

Instructor Notes:

Add instructor notes here.



Design Base

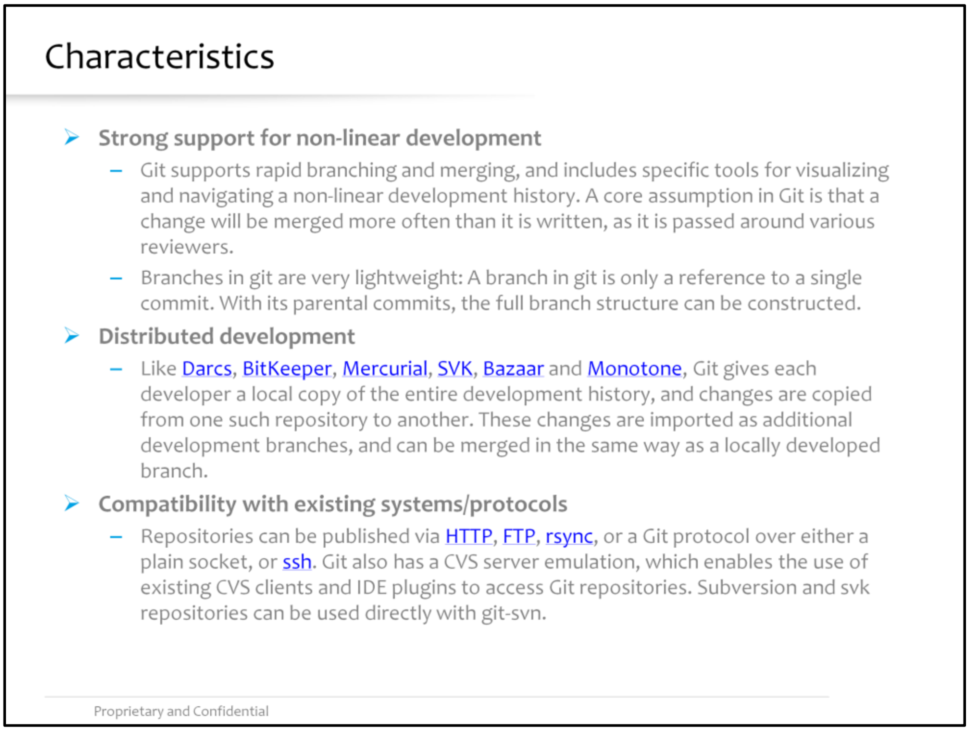
- Git's design was inspired by BitKeeper and Monotone. Git was originally designed as a low-level version control system engine on top of which others could write front ends, such as Cogito . The core Git project has since become a complete version control system that is usable directly.
- While strongly influenced by BitKeeper, Torvalds deliberately attempted to avoid conventional approaches, leading to a unique design.

Proprietary and Confidential

BitKeeper is a software tool for distributed revision control (configuration management,SCM, etc.) of computer source code
Monotone is an open source software tool for distributed revision control.
Cogito (originally git-pasky) is a revision control system layered on top of Git. It is historically the first Git frontend, which appeared in April 2005, just days after Git itself. While Git was initially meant just as the low-level interface, Cogito started with the stated goal of becoming a user-friendly front-end.

Instructor Notes:

Add instructor notes here.



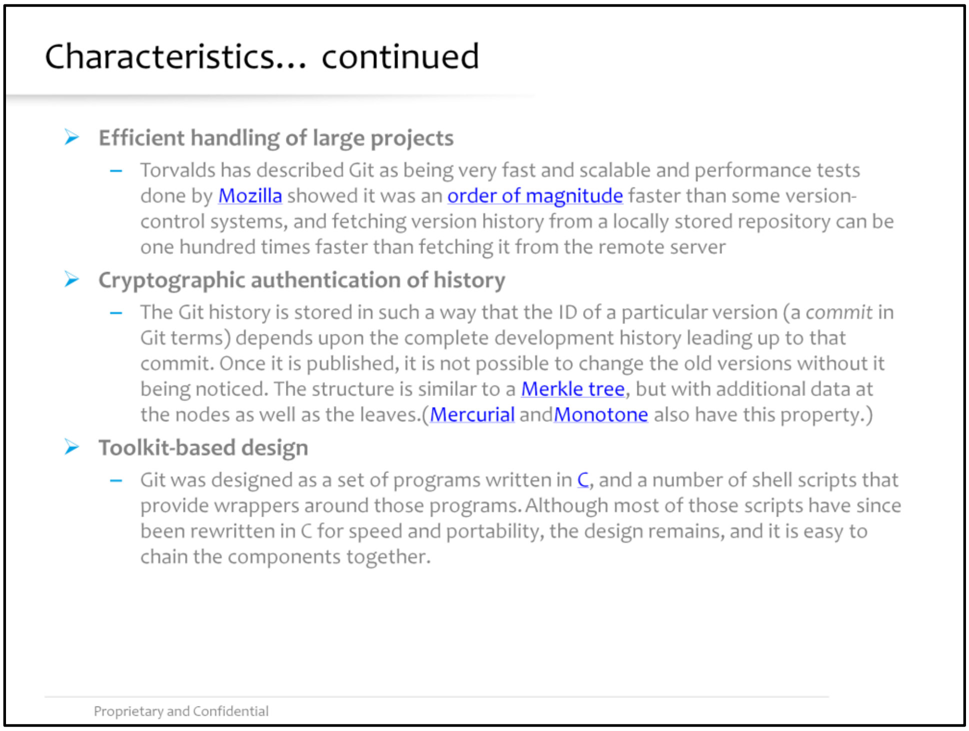
Characteristics

- **Strong support for non-linear development**
  - Git supports rapid branching and merging, and includes specific tools for visualizing and navigating a non-linear development history. A core assumption in Git is that a change will be merged more often than it is written, as it is passed around various reviewers.
  - Branches in git are very lightweight: A branch in git is only a reference to a single commit. With its parental commits, the full branch structure can be constructed.
- **Distributed development**
  - Like Darcs, BitKeeper, Mercurial, SVK, Bazaar and Monotone, Git gives each developer a local copy of the entire development history, and changes are copied from one such repository to another. These changes are imported as additional development branches, and can be merged in the same way as a locally developed branch.
- **Compatibility with existing systems/protocols**
  - Repositories can be published via HTTP, FTP, rsync, or a Git protocol over either a plain socket, or ssh. Git also has a CVS server emulation, which enables the use of existing CVS clients and IDE plugins to access Git repositories. Subversion and svk repositories can be used directly with git-svn.

Proprietary and Confidential

Instructor Notes:

Add instructor notes here.



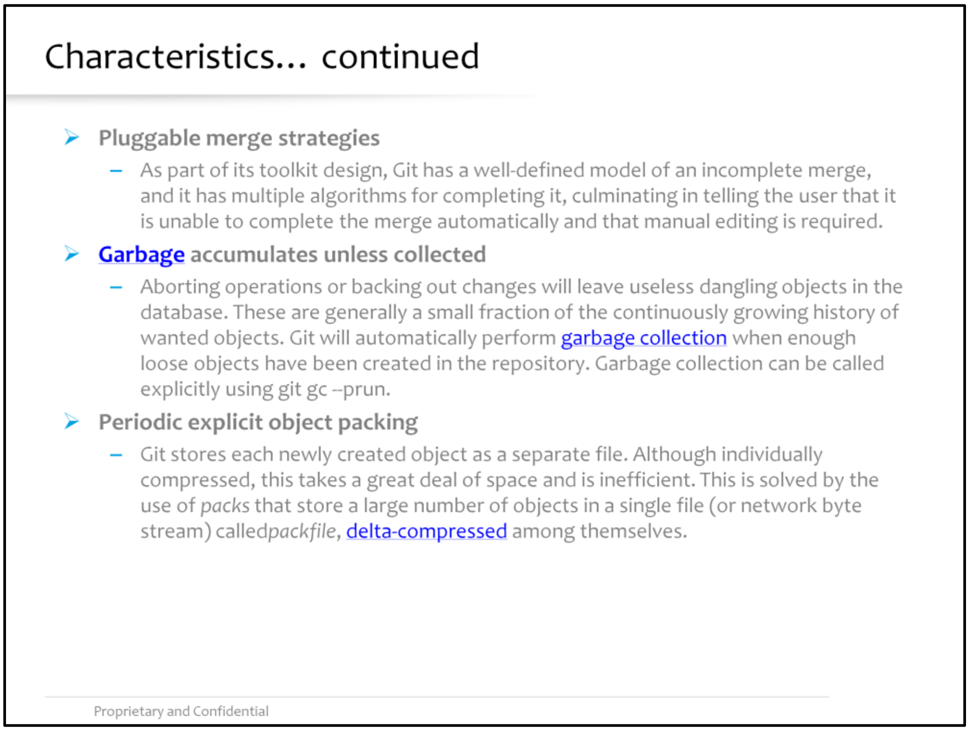
Characteristics… continued

- **Efficient handling of large projects**
  - Torvalds has described Git as being very fast and scalable and performance tests done by Mozilla showed it was an order of magnitude faster than some version-control systems, and fetching version history from a locally stored repository can be one hundred times faster than fetching it from the remote server
- **Cryptographic authentication of history**
  - The Git history is stored in such a way that the ID of a particular version (a *commit* in Git terms) depends upon the complete development history leading up to that commit. Once it is published, it is not possible to change the old versions without it being noticed. The structure is similar to a Merkle tree, but with additional data at the nodes as well as the leaves.(Mercurial andMonotone also have this property.)
- **Toolkit-based design**
  - Git was designed as a set of programs written in C, and a number of shell scripts that provide wrappers around those programs. Although most of those scripts have since been rewritten in C for speed and portability, the design remains, and it is easy to chain the components together.

Proprietary and Confidential

Instructor Notes:

Add instructor notes here.



Characteristics… continued

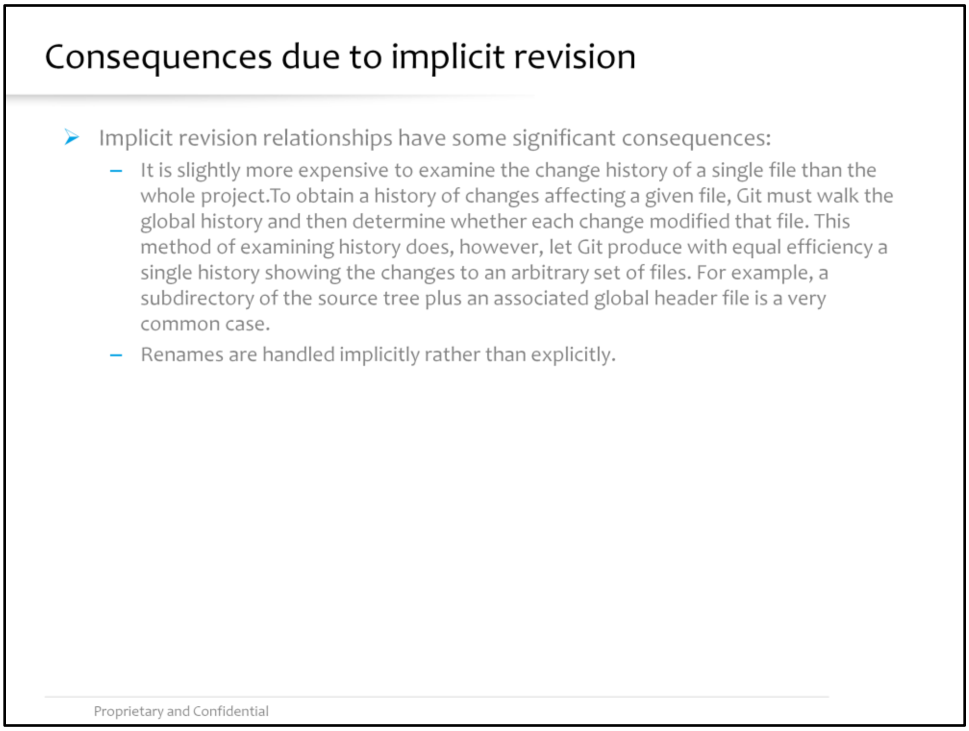
- Pluggable merge strategies
  - As part of its toolkit design, Git has a well-defined model of an incomplete merge, and it has multiple algorithms for completing it, culminating in telling the user that it is unable to complete the merge automatically and that manual editing is required.
- Garbage accumulates unless collected
  - Aborting operations or backing out changes will leave useless dangling objects in the database. These are generally a small fraction of the continuously growing history of wanted objects. Git will automatically perform garbage collection when enough loose objects have been created in the repository. Garbage collection can be called explicitly using git gc --prun.
- Periodic explicit object packing
  - Git stores each newly created object as a separate file. Although individually compressed, this takes a great deal of space and is inefficient. This is solved by the use of *packs* that store a large number of objects in a single file (or network byte stream) called*packfile*, delta-compressed among themselves.

Proprietary and Confidential

Packs are compressed using the heuristic that files with the same name are probably similar, but do not depend on it for correctness. A corresponding index file is created for each packfile, telling the offset of each object in the packfile. Newly created objects (newly added history) are still stored singly, and periodic repacking is required to maintain space efficiency. The process of packing the repository can be very computationally expensive. By allowing objects to exist in the repository in a loose, but quickly generated format, git allows the expensive pack operation to be deferred until later when time does not matter (e.g. the end of the work day). Git does periodic repacking automatically but manual repacking is also possible with the git gccommand. For data integrity, both packfile and its index have SHA-1 checksum inside, and also the file name of packfile contains a SHA-1 checksum. To check integrity, run the git fsck command.Another property of Git is that it snapshots directory trees of files. The earliest systems for tracking versions of source code, SCCS andRCS, worked on individual files and emphasized the space savings to be gained from interleaved deltas (SCCS) or delta encoding (RCS) the (mostly similar) versions. Later revision control systems maintained this notion of a file having an identity across multiple revisions of a project. However, Torvalds rejected this concept.Consequently, Git does not explicitly record file revision relationships at any level below the source code tree.

Instructor Notes:

Add instructor notes here.



Consequences due to implicit revision

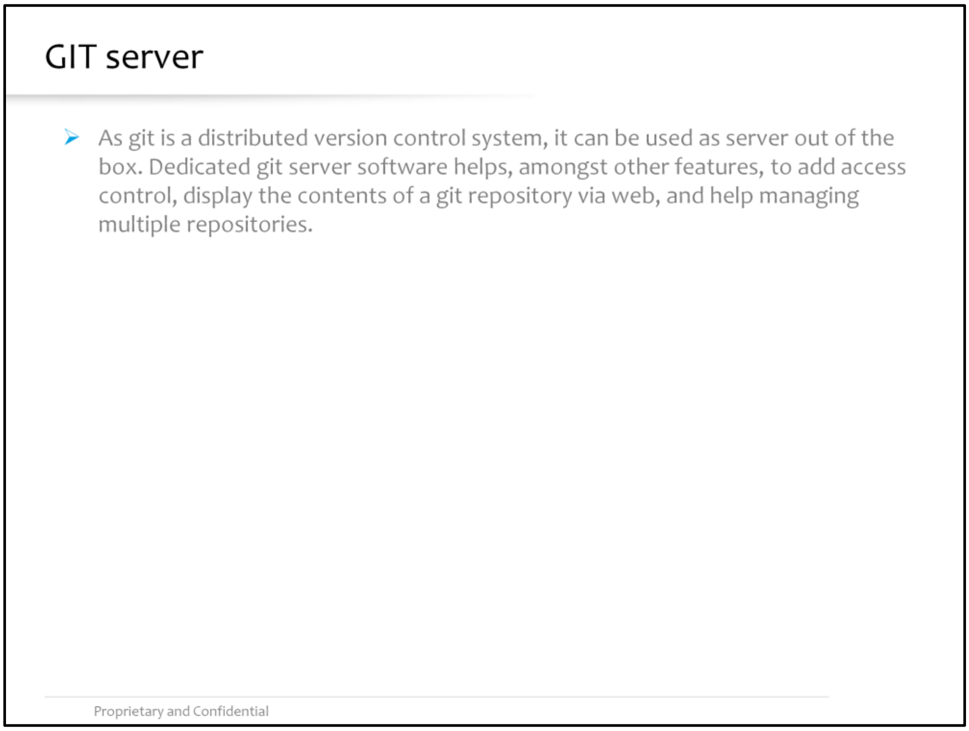
- Implicit revision relationships have some significant consequences:
  - It is slightly more expensive to examine the change history of a single file than the whole project.To obtain a history of changes affecting a given file, Git must walk the global history and then determine whether each change modified that file. This method of examining history does, however, let Git produce with equal efficiency a single history showing the changes to an arbitrary set of files. For example, a subdirectory of the source tree plus an associated global header file is a very common case.
  - Renames are handled implicitly rather than explicitly.

Proprietary and Confidential

Renames are handled implicitly rather than explicitly. A common complaint with CVS is that it uses the name of a file to identify its revision history, so moving or renaming a file is not possible without either interrupting its history, or renaming the history and thereby making the history inaccurate. Most post-CVS revision control systems solve this by giving a file a unique long-lived name (a sort ofinode number) that survives renaming. Git does not record such an identifier, and this is claimed as an advantage.Source codefiles are sometimes split or merged as well as simply renamed, and recording this as a simple rename would freeze an inaccurate description of what happened in the (immutable) history. Git addresses the issue by detecting renames while browsing the history of snapshots rather than recording it when making the snapshot.(Briefly, given a file in revision *N*, a file of the same name in revision*N*−*1* is its default ancestor. However, when there is no like-named file in revision *N*−*1*, Git searches for a file that existed only in revision *N*−*1* and is very similar to the new file.) However, it does require more CPU-intensive work every time history is reviewed, and a number of options to adjust the heuristics. This mechanism does not always work; sometimes a file that is renamed with changes in the same commit is read as a deletion of the old file and the creation of a new file. Developers can work around this limitation by committing the rename and changes separately.

Instructor Notes:

Add instructor notes here.

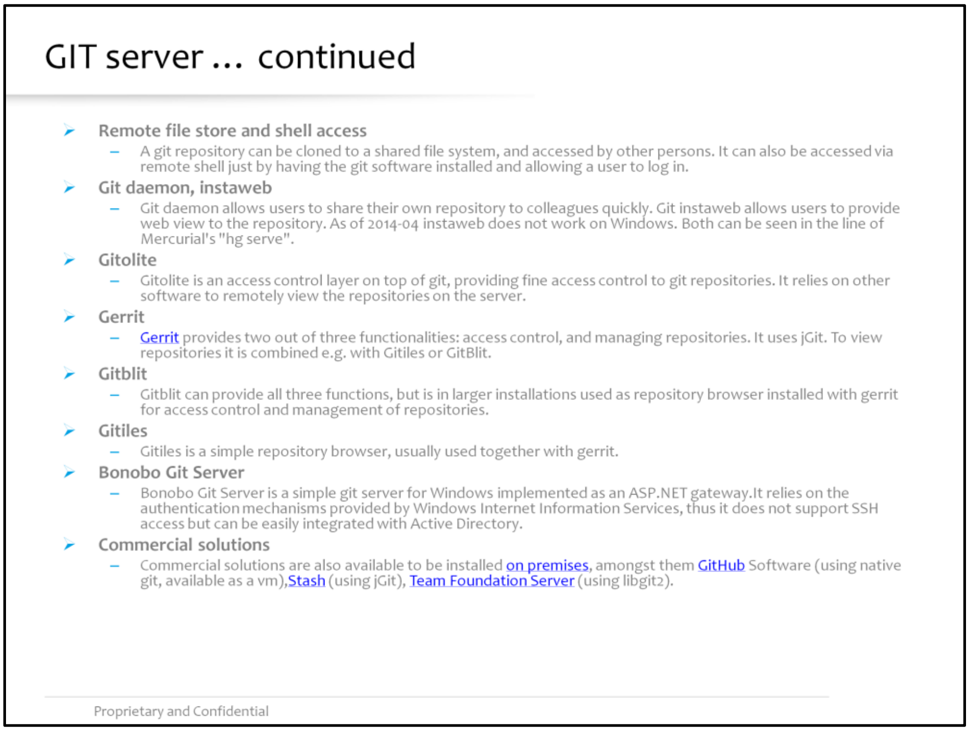


GIT server

- As git is a distributed version control system, it can be used as server out of the box. Dedicated git server software helps, amongst other features, to add access control, display the contents of a git repository via web, and help managing multiple repositories.

Instructor Notes:

Add instructor notes here.



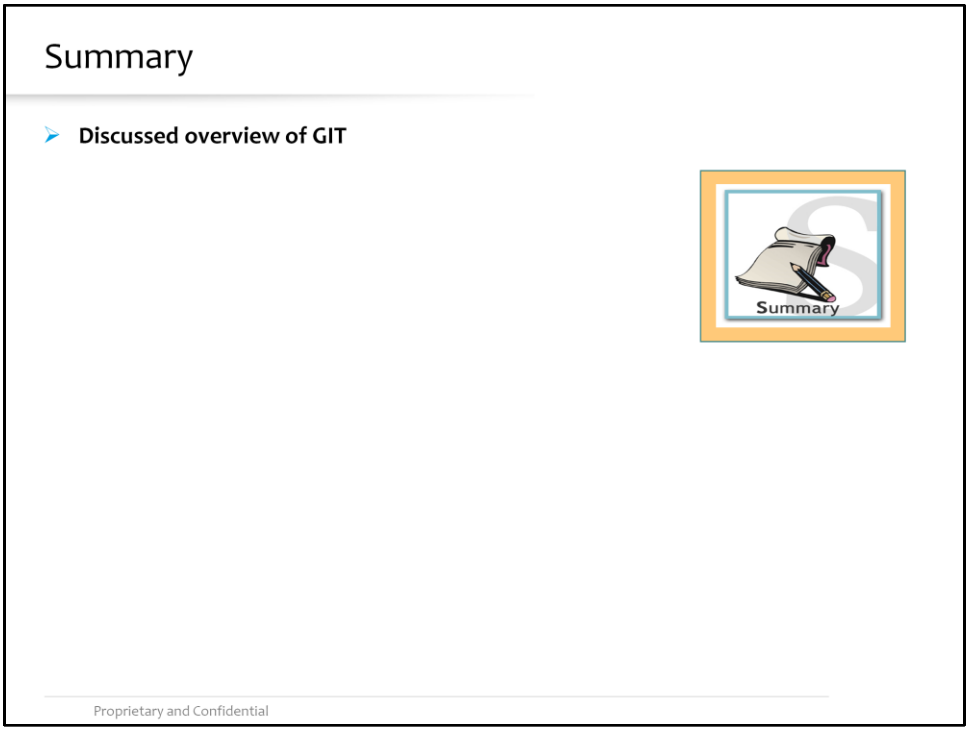
GIT server … continued

- Remote file store and shell access
  - A git repository can be cloned to a shared file system, and accessed by other persons. It can also be accessed via remote shell just by having the git software installed and allowing a user to log in.
- Git daemon, instaweb
  - Git daemon allows users to share their own repository to colleagues quickly. Git instaweb allows users to provide web view to the repository. As of 2014-04 instaweb does not work on Windows. Both can be seen in the line of Mercurial's "hg serve".
- Gitolite
  - Gitolite is an access control layer on top of git, providing fine access control to git repositories. It relies on other software to remotely view the repositories on the server.
- Gerrit
  - Gerrit provides two out of three functionalities: access control, and managing repositories. It uses jGit. To view repositories it is combined e.g. with Gitiles or GitBlit.
- Gitblit
  - Gitblit can provide all three functions, but is in larger installations used as repository browser installed with gerrit for access control and management of repositories.
- Gitiles
  - Gitiles is a simple repository browser, usually used together with gerrit.
- Bonobo Git Server
  - Bonobo Git Server is a simple git server for Windows implemented as an ASP.NET gateway.It relies on the authentication mechanisms provided by Windows Internet Information Services, thus it does not support SSH access but can be easily integrated with Active Directory.
- Commercial solutions
  - Commercial solutions are also available to be installed on premises, amongst them GitHub Software (using native git, available as a vm),Stash (using jGit), Team Foundation Server (using libgit2).

Proprietary and Confidential

Instructor Notes:

Add instructor notes here.



Summary

- Discussed overview of GIT

Proprietary and Confidential

Add the notes here.