GIT

Naveen Kumar K S
adith.Naveen@gmail.com

Agenda

- History of Git
- Distributed V.S Centralized Version Control
- GIT Architecture
- Getting started
- Branching and Merging
- Working with remote
- Summary

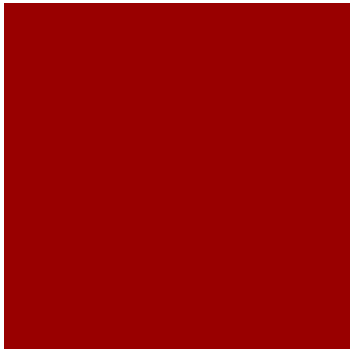
A Brief History

- Linus Torvalds uses BitKeeper to manage Linux code
- Ran into BitKeeper licensing issue
  - Liked functionality
  - Looked at CVS as how not to do things
- April 5, 2005 - Linus sends out email showing first version
- June 15, 2005 - Git used for Linux version control

GIT is not SCM

Never mind merging. It's not an SCM, it's a distribution and archival mechanism. I bet you could make a reasonable SCM on top of it, though. Another way of looking at it is to say that it's really a content-addressable filesystem, used to track directory trees.

Linus Torvalds, 7 Apr 2005



Centralized Version Control

- Traditional version control system
  - Server with database
  - Clients have a working version
- Examples
  - CVS
  - Subversion
  - Visual Source Safe
- Challenges
  - Multi-developer conflicts
  - Client/server communication

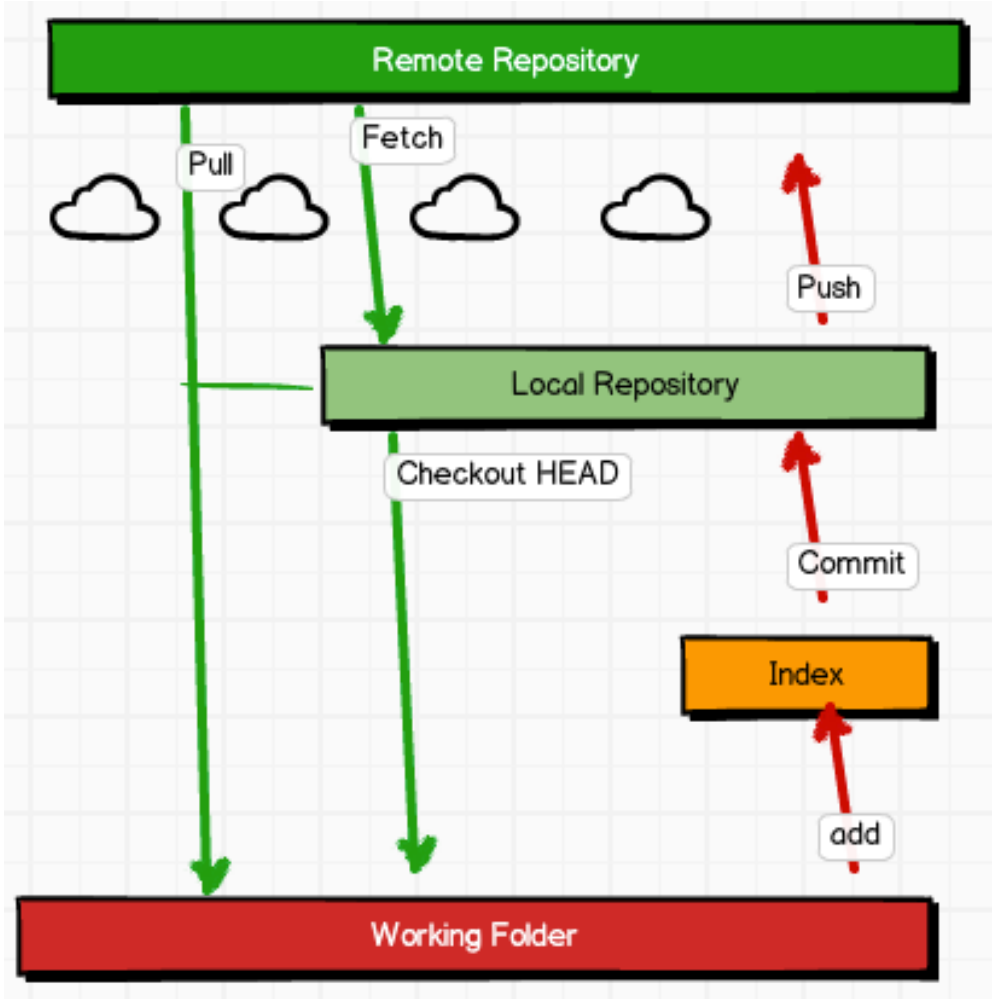
Advantages

- Resilience
  - No one repository has more data than any other
- Speed
  - Very fast operations compared to other VCS (I'm looking at you CVS and Subversion)
- Space
  - Compression can be done across repository not just per file
  - Minimizes local size as well as push/pull data transfers
- Simplicity
  - Object model is very simple
- Large userbase with robust tools

Disadvantages

- Definite learning curve, especially for those used to centralized systems
  - Can sometimes seem overwhelming to learn
    - Conceptual difference
    - Huge amount of commends

GIT Architecture



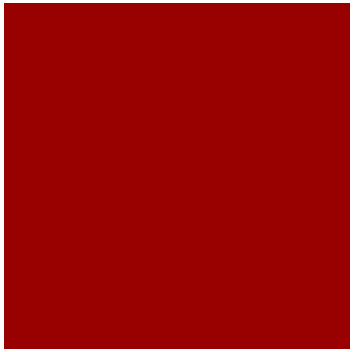
Before that lets Install

You can download GIT from

https://git-scm.com/downloads

Latest Version : 2.4.2

Getting Started



- A basic workflow
  - (Possible init or clone) Init a repo
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes

git init

- The **`git init`** command creates a new Git repository.
- It can be used to convert an existing, unversioned project to a Git repository or initialize a new empty repository.
- Executing `git init` creates a .git subdirectory in the project root, which contains all of the necessary metadata for the repo.
- View with `ls -la`

git init

```
git init <directory>
```

Above command will create an empty Git repository in the specified directory. Running this command will create a new folder called <directory> containing nothing but the .git subdirectory.

git bare

Syntax : `git init --bare <directory>`

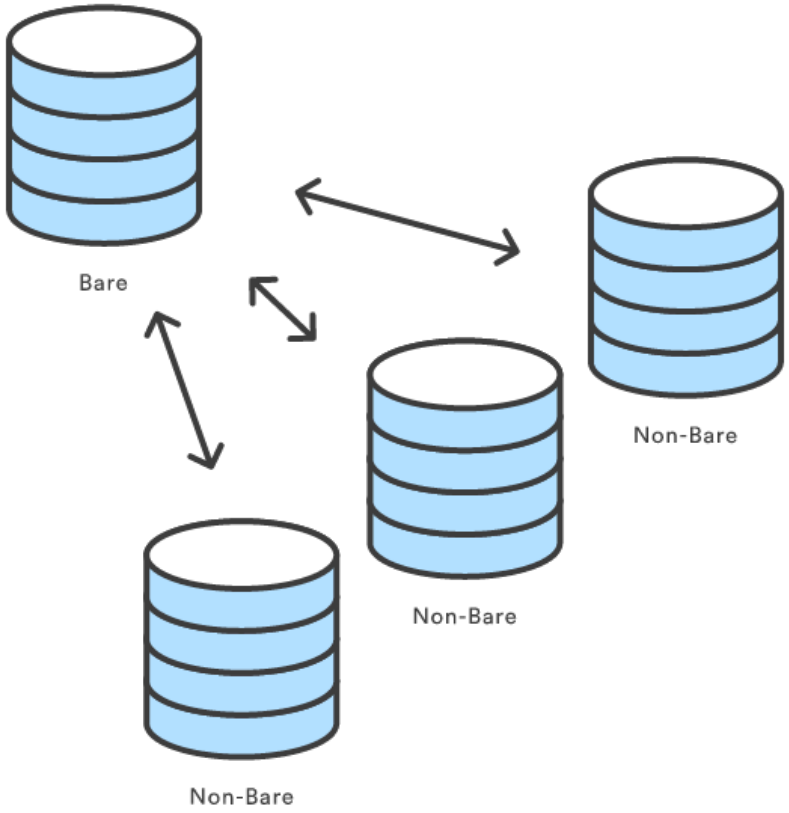
Example : `git init –bare my-project.git`

- Initialize an empty Git repository, but omit the working directory.
- Shared repositories should always be created with the --bare flag
- Conventionally, repositories initialized with the --bare flag end in .git.

Bare Repositories

- The --bare flag creates a repository that doesn't have a working directory, making it impossible to edit files and commit changes in that repository.
- Central repositories should always be created as bare repositories because pushing branches to a non-bare repository has the potential to overwrite changes
- Virtually all Git workflows, the central repository is bare, and developers local repositories are non-bare.

Bare Repositories



git clone

- The git clone command copies an existing Git repository(This is sort of like svn checkout).
- Git repository—it has its own history, manages its own files, and is a completely isolated environment from the original repository.

Usage : **`git clone <repo>`**

a remote machine accessible via HTTP or SSH.

`git clone <repo> <directory>`

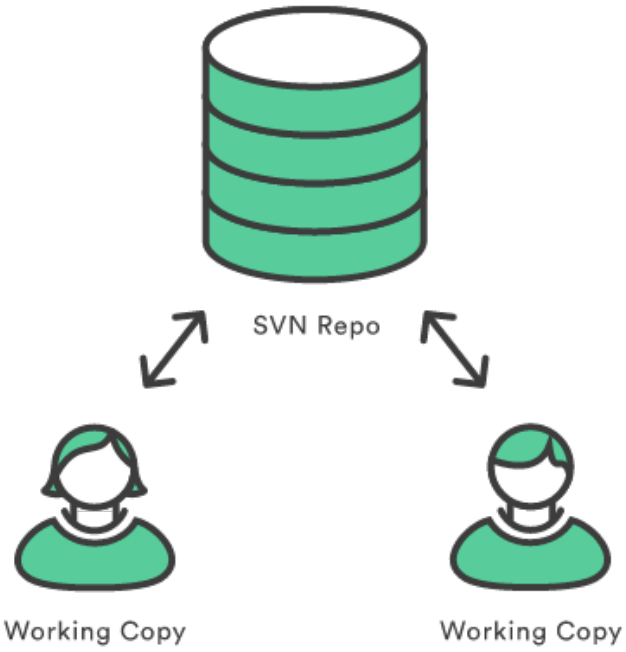
Clone the repository located at <repo> into the folder called <directory> on the local machine.

GIT vs SVN

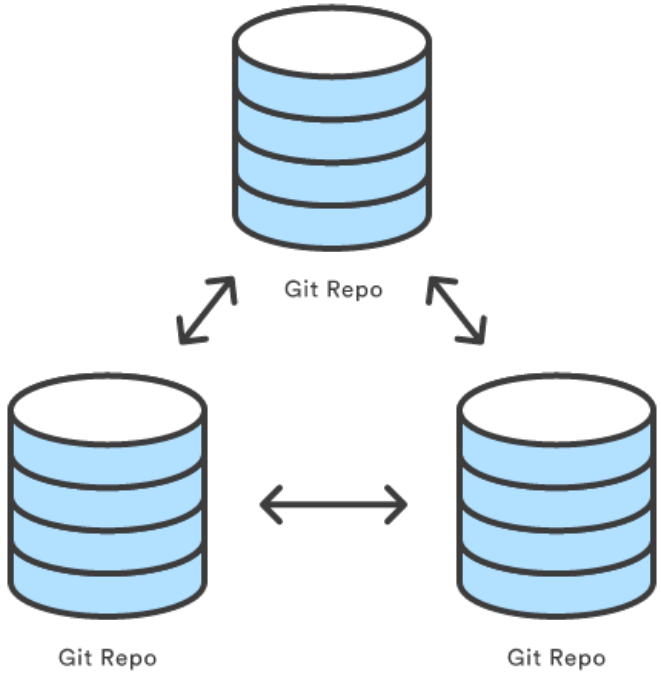
- Unlike SVN, Git makes no distinction between the working copy and the central repository—they are all full-fledged Git repositories.
- SVN depends on the relationship between the central repository and the working copy, Git's collaboration model is based on repository-to-repository interaction.
- In GIT you push or pull commits from one repository to another.

GIT vs SVN

Central-Repo-to-Working-Copy Collaboration



Repo-To-Repo Collaboration



git config

- he git config command lets you configure your Git installation (or an individual repository) from the command line.

Usage: **`git config user.name <name>`**

Define the author name to be used for all commits in the current repository.

Typically, you'll want to use the --global flag to set configuration options for the current user.

git config

`git config --global user.name <name>`

Define the author name to be used for all commits by the current user.

`git config --global user.email <email>`

Define the author email to be used for all commits by the current user.

`git config --global alias.<alias-name> <git-command>`

Create a shortcut for a Git command.

git config

```
git config --system core.editor <editor>
```

Define the text editor used by commands

Note: editor : vim, notepad.exe etc

```
git config --global --edit
```

Open the global configuration file in a text editor for manual editing.

git config

Git stores configuration options in three separate files, which lets you scope options to individual repositories, users, or the entire system:

- <repo>/.git/config – Repository-specific settings.
- ~/.gitconfig – User-specific settings. This is where options set with the --global flag are stored.
- $(prefix)/etc/gitconfig – System-wide settings.

git add

- The `git add` command adds a change in the working directory to the staging area.
- However, git add doesn't really affect the repository in any significant way—changes are not actually recorded until you run `git commit`.

Usage : `git add <file>`

Stage all changes in <file> for the next commit.

`git add <directory>`

Stage all changes in <file> for the next commit.

git add

Example

When you're starting a new project, git add serves the same function as svn import.

```
git add .
```
(dot for all files)

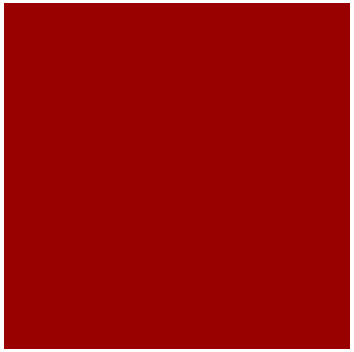
```
git commit
```

For Individual File

```
git add Hello.java
```

```
git commit
```

Stage Area



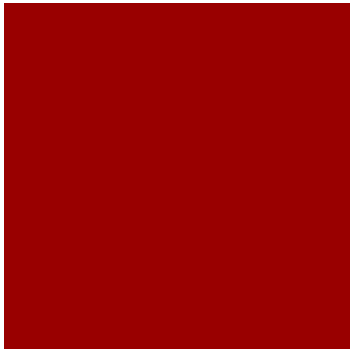
- It helps to think of it as a buffer between the working directory and the project history.

git commit

Snapshots are always committed to the local repository. This is fundamentally different from SVN, wherein the working copy is committed to the central repository.

Just as the staging area is a buffer between the working directory and the project history, each developer's local repository is a buffer between their contributions and the central repository.

Inspecting a repository



git status

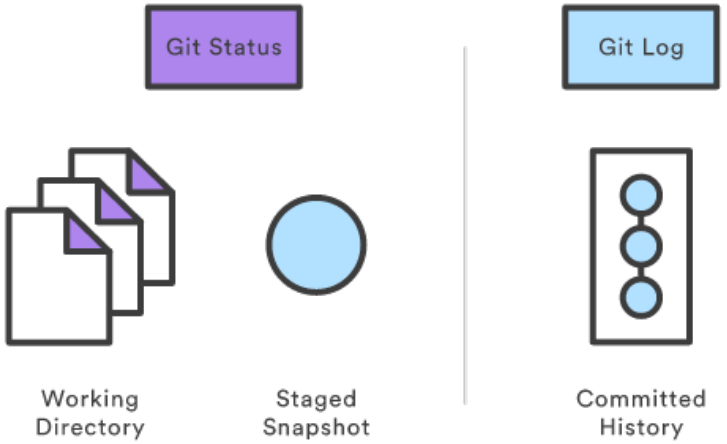
- The git status command displays the state of the working directory and the staging area.
- It lets you see which changes have been staged, which haven't, and which files aren't being tracked by Git.

Usage: `git status`

Inspecting a repository

git log

- The git log command displays committed snapshots.
- It lets you list the project history, filter it, and search for specific changes.
- While git status lets you inspect the working directory and the staging area.



Customized git log

```
git log
```

Display the entire commit history using the default formatting.

```
git log -n <limit>
```

Limit the number of commits by <limit>. For example, **`git log -n 3`** will display only 3 commits.

```
git log --oneline
```

Condense each commit to a single line.

Customized git log

```
git log --stat
```

Along with the ordinary git log information, include which files were altered

```
git log -p
```

Display the patch representing each commit.

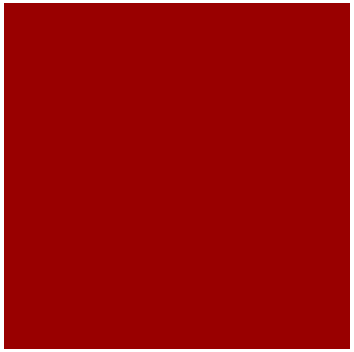
```
git log --author="<pattern>"
```

Search for commits by a particular author.

Customized git log

- `git log --grep="<pattern>"`
  - Search for commits with a commit message that matches <pattern>
- `git log <file>`
  - Only display commits that include the specified file.
- `git log --graph --decorate –oneline`
  - –graph flag that will draw a text based graph of the commits on the left hand side of the commit messages.
  - –decorate adds the names of branches or tags of the commits that are shown.

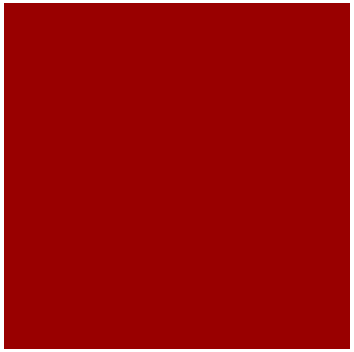
Viewing old commits



git checkout

- The git checkout command serves three distinct functions: checking out files, checking out commits, and checking out branches.
- Lets concentrate only on two, later on branches

Viewing old commits



Usage: git checkout master

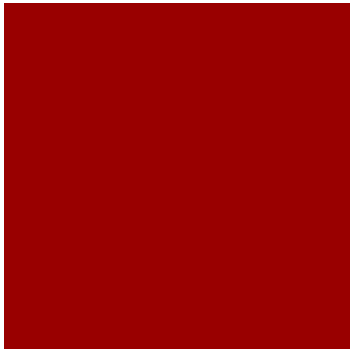
Return to the master branch.

Usage: `git checkout <commit> <file>`

Example: `git checkout a1e8fb5 Hello.java`

Check out a previous version of a file.

Viewing old commits



Usage: `git checkout <commit>`

Example: `git checkout a1e8fb5`

Update all files in the working directory to match the specified commit.

You can check out the most recent version with the following:

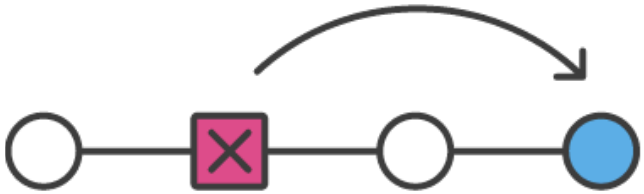
Example: `git checkout HEAD Hello.java`

git revert

Usage: `git revert <commit>`

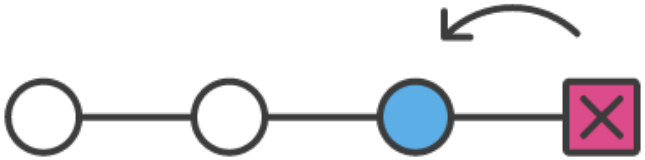
Generate a new commit that undoes all of the changes introduced in <commit>, then apply it to the current branch.

Reverting



Note: It's important to understand that git revert undoes a single commit, as shown in next slide

Resetting



git revert

Example

The following example is a simple demonstration of git revert. It commits a snapshot, then immediately undoes it with a revert.

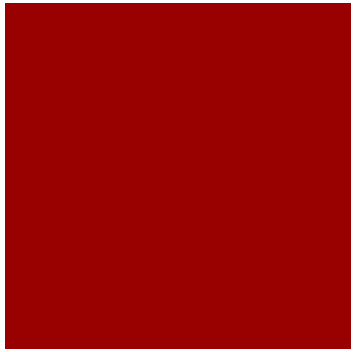
Edit some tracked files

Commit a snapshot

```
git commit -m "Make some changes that will be undone”
```

Revert the commit we just created

```
git revert HEAD
```



git revert

`git revert –abort`

To abort the revert process

git reset

- If `git revert` is a “safe” way to undo changes, you can think of `git reset` as the dangerous method.
- When you undo with `git reset` there is no way to retrieve the original copy—it is a permanent undo.

`git reset --hard <commit>`

Move the current branch tip backward to <commit> and reset both the staging area and the working directory to match. This obliterates not only the uncommitted changes, but all commits after <commit>, as well.

git clean

`git clean –n`

Perform a “dry run” of git clean. This will show you which files are going to be removed without actually doing it.

`git clean -f`

Remove untracked files from the current directory. The -f (force) flag is required unless the clean

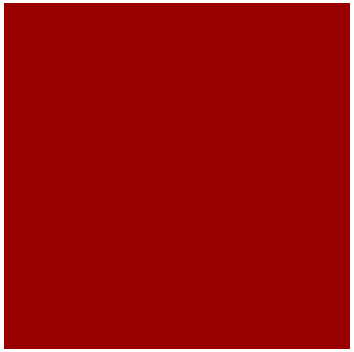
git clean

```
git clean -f <path>
```

Remove untracked files, but limit the operation to the specified path.

```
git clean -df
```

Remove untracked files and untracked directories from the current directory.



git clean

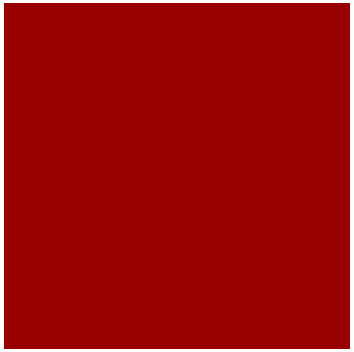
```
git clean -xf
```

Remove untracked files from the current directory as well as any files that Git usually ignores.

Rewriting history

- We discusses some of the most common reasons for overwriting committed snapshots and shows you how to avoid the pitfalls of doing so.

--amend

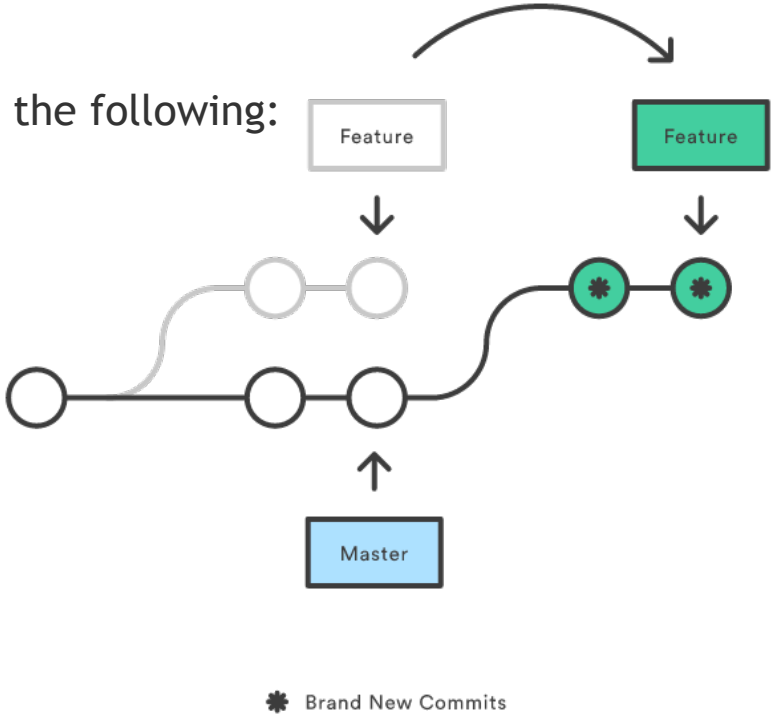


- The `git commit --amend` command is a convenient way to fix up the most recent commit.
- It lets you combine staged changes with the previous commit instead of committing it as an entirely new snapshot.
- It can also be used to simply edit the previous commit message without changing its snapshot.

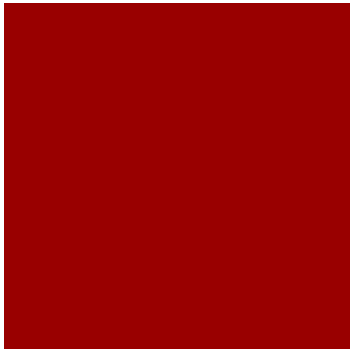
git rebase

- Rebasing is the process of moving a branch to a new base commit.

The general process can be visualized as the following:



git rebase



- Rebasing really is just moving a branch from one commit to another
- But internally, Git accomplishes this by creating new commits and applying them to the specified base—it's literally rewriting your project history.

Usage

```
git rebase <base>
```

Example - rebase

```
# Start a new feature

git checkout -b new-feature master

# Edit files

git commit -a -m "Start developing a
feature"
```

In the middle of our feature, we realize there's a security hole in our project

```
# Create a hotfix branch based off of master
git checkout -b hotfix master
# Edit files
git commit -a -m "Fix security hole"
# Merge back into master
git checkout master
git merge hotfix
git branch -d hotfix
```

After merging the hotfix into master, we have a forked project history. Instead of a plain git merge, we'll integrate the feature branch with a rebase to maintain a linear history:

```
git checkout new-feature
git rebase master
```

This moves new-feature to the tip of master, which lets us do a standard fast-forward merge from master:

```
git checkout master
git merge new-feature
```

git reflog

- Git keeps track of updates to the tip of branches using a mechanism called reflog.

Usage: `git reflog`

Show the reflog for the local repository.

`git reflog --relative-date`

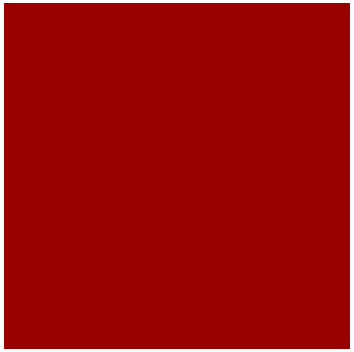
Show the reflog with relative date information (e.g. 2 weeks ago).

Syncing

SVN vs GIT

- SVN uses a single central repository to serve as the communication
- This is different from Git's collaboration model, which gives every developer their own copy of the repository, complete with its own local history and branch structure.
- Git lets you share entire branches between repositories.

git remote



- The git remote command lets you create, view, and delete connections to other repositories.

Thank yOU