Internet Applications

Introduction to Git

Suhel Hammoud

Git Tutorial Overview

- How to import projects into Git
- How to commit changes
- How to manage branches
- How to collaborate using Git

Getting Help in Git

```
$ man git-log
$ git help log
```

- man shows the manual page
- git help allows using any viewer (see git-help[1])

Setting up Your Identity

- Configures your Git username and email globally.

```
$ git config --global user.name "Your Name"
$ git config --global user.email "you@yourdomain.example.com"
```

Importing a New Project

```
$ tar xzf project.tar.gz
$ cd project
$ git init
  Initialized empty Git repository in .git/
```

- Project is now versioned under Git.

Making changes

Modify some files, then add their updated contents to the index:

```
$ git add file1 file2 file3
```

You can see what is about to be committed using git diff with the --cached option:

```
$ git diff --cached
```

Without --cached, git diff will show you any changes that you’ve made but not yet added to the index.

Status

You can also get a brief summary of the situation with git status:

```
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)

    modified:   file1
    modified:   file2
    modified:   file3
```

Commit Changes

- Commit your changes with:

```
$ git commit
```

- Will again prompt you for a message describing the change, and then record a new version of the project.
- Alternatively, instead of running `git add` beforehand, you can use

```
$ git commit -a
```

Good Commit Messages

- Start with a short title (≤50 characters)
- Then a blank line
- Then a detailed description

Example:

```
Fix login bug
Fixed session timeout issue causing user logout.
Added refresh token support.
```

Git Tracks Content, Not Files

- git add stages **new and modified contents**.
- Git does not track "file renames" explicitly — only content.

Ignoring Files

- Use .gitignore to tell Git which files/folders to ignore.

```
# Example .gitignore
*.log
node_modules/
.env
```

Viewing Project History

```
$ git log
commit 89abcde...
Author: Your Name <you@example.com>
Date:   Thu Apr 29 12:00 2025

    Updated file1, file2, file3

commit 0123456...
Author: Your Name <you@example.com>
Date:   Thu Apr 29 11:00 2025

    Initial commit
```

Managing Branches

```
$ git branch experimental
$ git branch
* main
  experimental
$ git switch experimental
    Switched to branch 'experimental'
```

- Alternatively, use `git checkout -b` to create a new branch.

```
$ git checkout -b experimental
    Switched to a new branch 'experimental'
```

Merging Changes

- Merged development lines into main.

```
$ git checkout main
$ git merge experimental
    2 files changed, 2 insertions(+), 1 deletion(-)
```

Resolving Merge Conflicts

- Conflicts occur when two branches modify the same part of a file.
- Git will mark conflicts in the file.
- Manually edit the file to resolve conflicts.
- After fixing:

```
git add <file>
git commit
```

Deleting Branches

```
$ git branch -d experimental
    Deleted branch experimental (was 89abcde).
$ git branch -D crazy-idea
    Deleted branch crazy-idea (was deadbeef).
```

Using Git for Collaboration 1/2

- Bob clones Alice's repository.

```
bob$ git clone 'url_to_alice_repo' myrepo
    Cloning into 'myrepo'...
    done.
```

- Bob makes changes.

```
bob$ git commit -a
    [main 89abcde] Bob's changes
```

Using Git for Collaboration 2/2

- Bob pushes his changes to online repository.

```
bob$ git push origin main
```

- Alice pulls Bob's changes.

```
alice$ git pull 'url_to_bob_repo' main
```

Inspecting Remote Changes

```
$ git fetch https://github.com/bob/repo.git main
$ git log -p HEAD..FETCH_HEAD
```

- Review remote changes before merging.

Defining Remote Shortcuts

```
$ git remote add bob https://github.com/bob/repo.git
$ git fetch bob
```

- Simplify repetitive fetches and pulls.

Pulling from Remote Branch

```
$ git pull . remotes/bob/main
```

- Merge remote-tracked changes into the local branch.
- pull = fetch + merge.

Exploring History

```
$ git show HEAD
    commit 89abcde...
```

```
$ git show HEAD^
    commit 0123456...
```

```
$ git show HEAD~4
```

- Traverse commit ancestry.

Tagging Commits

```
$ git tag v2.5 1b2e1d63ff
```

- Mark commits for releases and stable points.

Diffing Commits

```
$ git diff v2.5 HEAD
$ git branch stable v2.5
```

- View differences and create stable branches.
- Start a new branch named "stable" based at v2.5

Resetting to Previous Commits

```
$ git reset --hard HEAD^
HEAD is now at 0123456 Initial commit
```

- **Warning**: Destroys history after HEAD^.

Searching Content

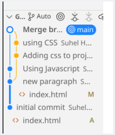
```
$ git grep "hello" v2.5
src/hello.c: printf("hello world!\n");
```

- Search for text across your repository.

Visualizing Git History

```
$ gitk
```

- Shows a full visual graph of branches, commits, merges.



```
G... Auto
Merge br... main
using CSS Suhel H...
Adding css to proj...
Using Javascript S...
new paragraph S...
index.html M
initial commit Suhel...
index.html A
```

Comparing Specific Files

```
$ git diff v2.5:Makefile HEAD:Makefile.in
```

```
$ git show v2.5:Makefile
```

- Compare different versions of individual files.

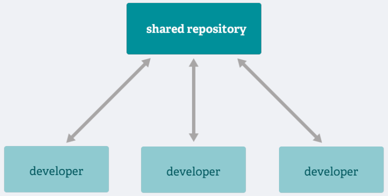
Distributed Git

- Git is **Distributed**.
- You do a **"clone"** of the entire repository.
- Each user can has a **full backup** of the main repository.
- If the server crashes, any user’s copy can restore it.
- No **single point of failure**, unless only one copy exists.
- **Endless workflows** can be implemented.
- Flexibility and robustness are core features.

Subversion-Style Workflow

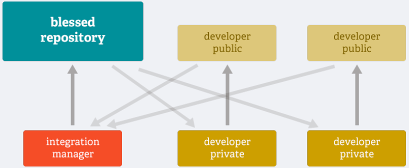
Centralized workflow for Git:

- All developers push to the same server.
- Git enforces synchronization: you can't push unless you've fetched new changes.



Integration Manager Workflow

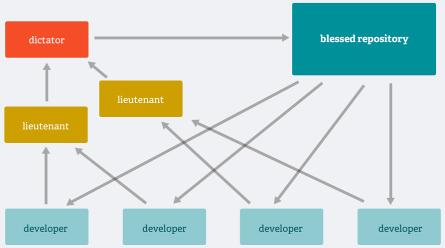
Popular in **open-source** projects:



- Developers clone the blessed repository.
- Push to their own repositories.
- Request the **Integrator** to pull changes.

Dictator and Lieutenants Workflow

Used for **large-scale projects** (e.g., Linux Kernel):



- **Lieutenants** manage subsystems.
- **Dictator** pulls from lieutenants.
- **Blessed repository** is updated for everyone to clone.

Summary

- Git tracks **content**, not files
- Branches and merges are cheap
- Collaboration is seamless
- Visualization tools (example: `gitk`) make Git easier to understand

References

- https://git-scm.com