

From Solo Git to Team Git

Solo Git –

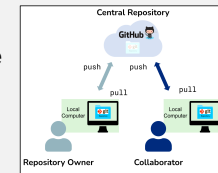
- one person, one *local* repository
- workflows
 - one linear history – commit after commit, no conflicts possible
 - branches to protect main – *conflicts never a real risk* because no one else ever updates main while you are working on a branch

Team Git –

- multiple people, one *shared* repository
- workflows
 - branches to protect main – *conflicts a real risk* because others may update main while you are working on a branch

Shared Repositories

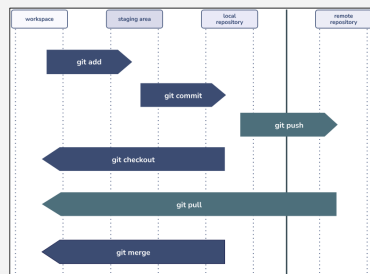
- Git's fundamental model: everyone works from their own local repository
 - commits only ever go to the local repository
- teams utilize a central shared repository that everyone syncs to
 - repositories *can* sync peer-to-peer, but the common pattern is everyone syncing to one central repository
 - GitHub is a common host for a central repository, especially for open-source projects but also for private team projects



Shared Repositories

New steps and terminology –

- *pull* to bring the shared repository's commits into your local working area
- *push* to send your local commits to the shared repository



Team Workflow

Branching and merging –

- create a branch, committing your work to the branch (locally) as you go
- merge your branch into main

Branching and merging, with a shared repository –

- *pull* the latest shared history into your local repository before starting new work
- create a branch, committing your work to the branch (locally) as you go
- *push* your branch to the shared repository so others can see it
- *pull* again before merging
 - someone else's work may have reached the shared repository while you were working
- merge your branch into the shared main, then *push* the result

Commit Hygiene and Conflicts

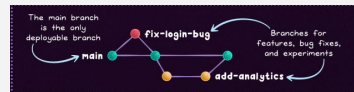
- *commit hygiene* matters more on a team
 - a vague commit message is now confusing to someone other than you
 - commit comments are important for locating points for revert or reset to should that be needed
 - a clear incremental commit history is part of the trail that makes your code understandable to someone who didn't write it
- a *merge conflict* happens when two branches changed the same lines differently
 - git can't guess which change should win
- *resolve a conflict* by reading both versions and deciding what the final code should be
 - a human judgment call
 - key is understanding what each version is trying to accomplish

Branching Strategies

- *feature branching*
 - isolate each piece of work in its own branch
 - merge into the main line when complete
- *trunk-based development*
 - minimize branch lifetime
 - integrate small changes into the main line frequently
- tradeoffs
 - isolation and safety vs integration speed and integrity
 - the longer two branches diverge, the more they may have changed the same code in incompatible ways
 - lose a private space to build things that aren't yet working
 - must break all work into small enough pieces to be able to commit frequently without breaking main
- a spectrum, not a binary choice
 - teams can adopt an appropriate commit frequency consistent with their needed workflow

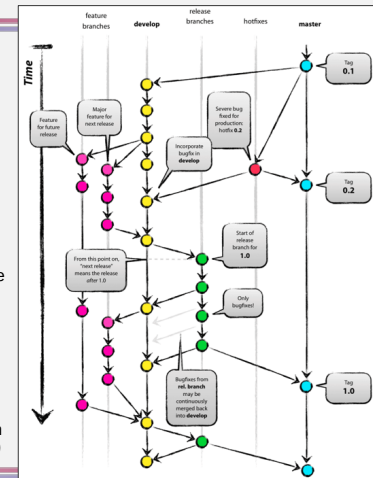
GitHub Flow

- designed for continuous delivery without a separate release line
 - closer to trunk-based development but with room for a lightweight review step
- key features
 - one main branch, always deployable
 - short-lived feature branches merged directly into main
- requires work broken into smallish chunks
 - commits within a feature branch don't need to be ready for main
 - still requires the whole chunk to be completely and reasonably small so the branch doesn't live long enough to drift far from main
- suitable for small-to-medium projects without a release-oriented deployment model



Other Git Workflows

- Git Flow
 - designed for software with formal, versioned release cycles
 - allows patching a live production release while also developing features for a new version
 - two long-lived branches
 - main – stable, production-ready code
 - develop – where the next release is prepared (integration branch for features)
 - other branches
 - feature – for individual features (branched from develop)
 - release – for preparing for a release (branched from develop)
 - hotfix – for fixing critical issues in production (branched from main)



Other Git Workflows

- GitLab Flow

- designed as a middle ground between GitHub Flow and Git Flow
 - adds environment branches to support a deployment pipeline

- branches

- main – stable, production-ready code
- staging/production/... – branches for specific steps of the deployment pipeline
- feature – for individual features

