Git remote usage

Dr Peadar Grant

February 1, 2024

Contents

1 Use cases S.2

2 Remotes S.5

3 Starting out S.6

4 Workflow S.7

5 Good commits S.8

1 Use cases

1. **As a single developer:**
    - Code on multiple machines
    - Deploy to server(s) or other production environment
    - Use as input to Continuous Integration (later!)
2. **Working with others:**
    - Easily and systematically share code amongst team members
    - Accomodates on-site, hybrid, fully remote, disconnected work patterns
    - Attribution of code changes
    - Controlled pull and push of changes amongst members

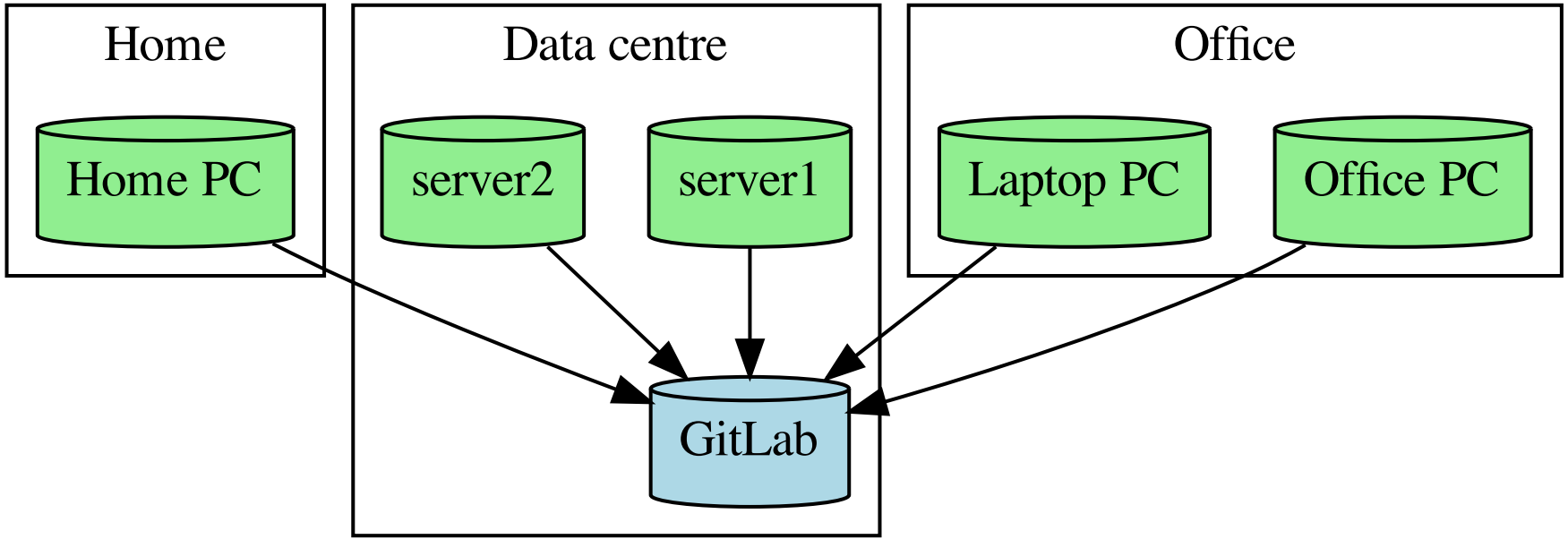

Figure 1: Researcher using git across multiple machines

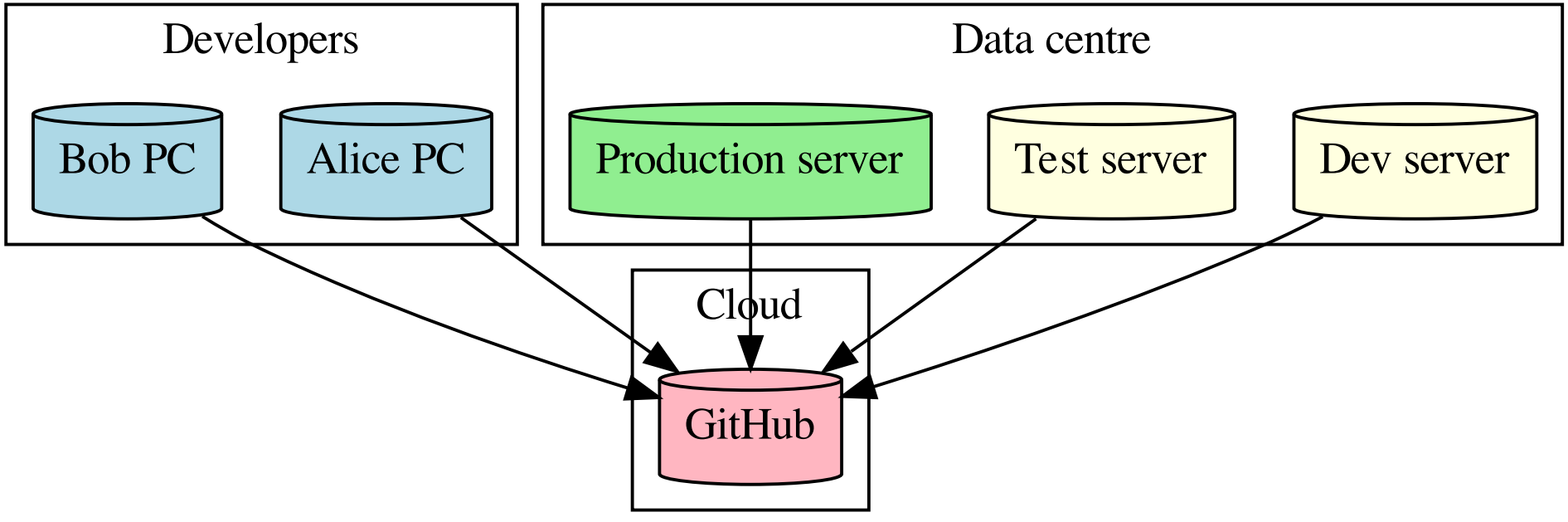

Figure 2: Team Git usage

2 Remotes

1. Technically any git repository can function as a remote.
2. Git can use a number of different protocols to share commits among repositories.
3. Most people nowadays use remotes provided by dedicated services (e.g. GitHub, GitLab)
   - Web interface for viewing code.
   - Access control for multiple developers
   - Value-added features like Continuous Integration
   - Integration point for new AI features (e.g. CoPilot)
   - Easy “plug-in” of other tools (e.g. IRC, Slack)

3 Starting out

Most common scenarios:

Remote repository already has code (easiest)

1. Use `git clone` to get *working copy* of remote.

Remote repository is empty (harder)

1. Add a *remote* using `git remote add`
2. Upload using `git push -u`

4 Workflow

1. **Before starting work** you `pull` changes in.
2. Use normal `add`, `commit` commands.
3. **When finished** you `push` commits to remote.

5 Good commits

Commits should be:

1. **Specific:** don't bundle a load of unrelated changes into a single commit.
    - There is no penalty for more commits vs larger commits!
2. **Tested:** don't commit code that you haven't tested.
    - In some very limited circumstances there may be justifications for doing this.
    - Some tools later on may help (e.g. branching)
3. **Complete:** make sure *all* files relevant are added:
    - Particularly important when adding new file(s)

5.1 Good commit messages

Commit messages should:

1. **Explain what** is being changed on a high-level.
2. **Explain why** something was done if necessary.
3. Clarify whether it is a **fix or a new feature**.
4. **Not re-state code changes** which we can see from `git diff`.
5. Explain why a particular course of action was taken, particularly if the changes aren't obvious.