CS 4350: Fundamentals of Software Engineering
CS 5500: Foundations of Software Engineering

Lesson 7.2 Using `git` in Teams

Jon Bell, John Boyland, Mitch Wand

Khoury College of Computer Sciences

Outline of this lesson

1. Review of `git` basics:
   a) clone, commit, push
   b) pull, stash
   c) branch, fetch, merge
2. How to collaborate using branches.
3. How to collaborate using forks.

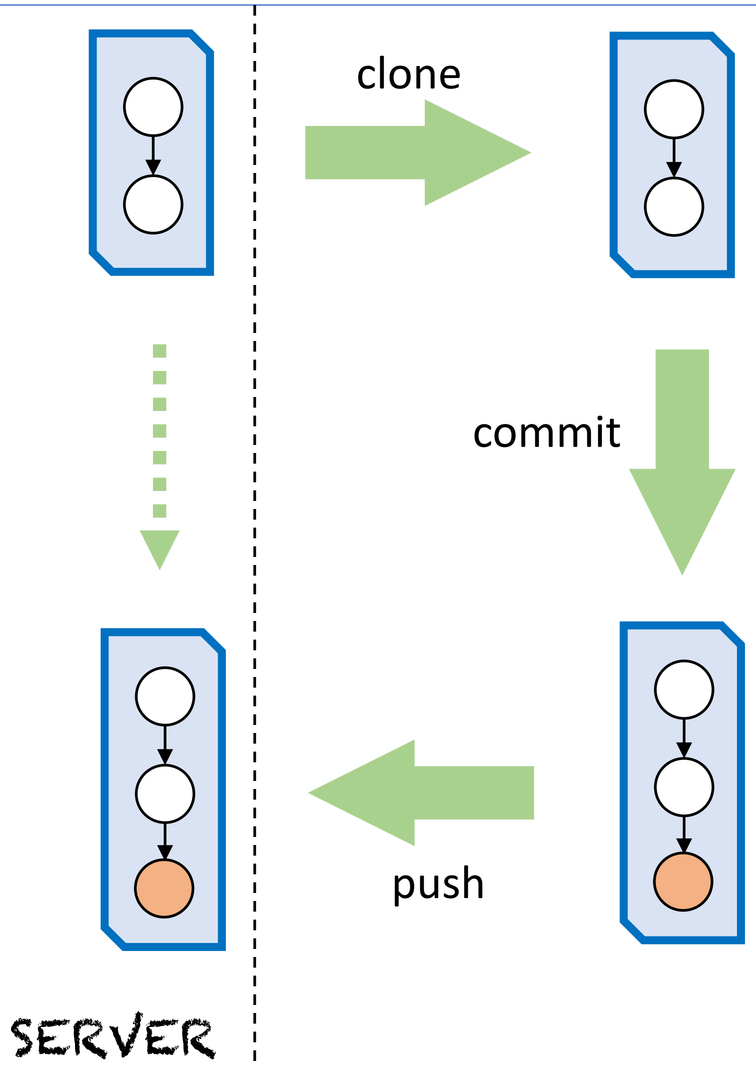
Learning Objectives for this Lesson

- By the end of this lesson, you should be able to:
  - Compare branches and forks on github;
  - Explain how branches and forks can be used for collaboration;
  - Describe the lifetime of a “pull request.”

Review: `git` Basics (1 of 3)

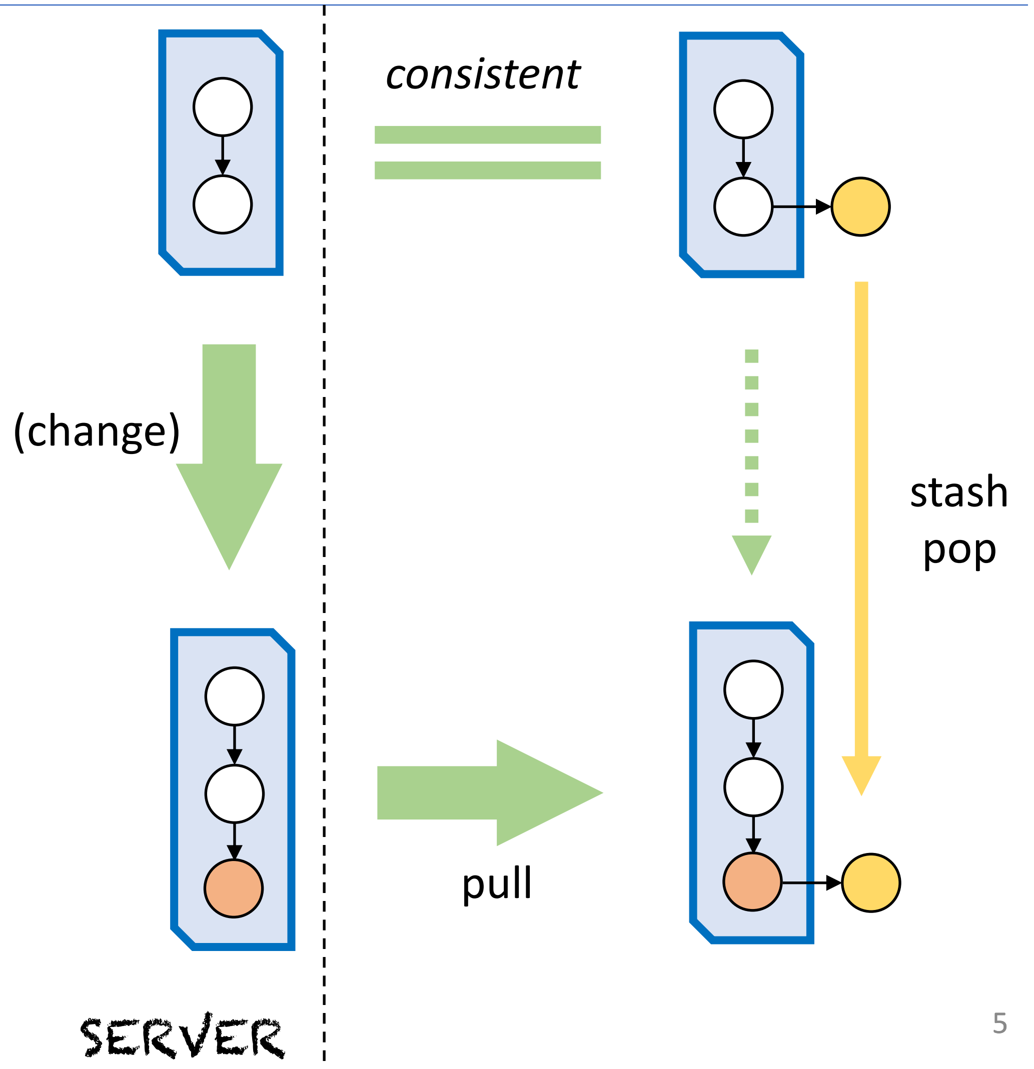
- Using a repo from a server:
  1. Clone the repo locally.
  2. Perform changes:
     - Modify files;
     - Add files;
     - Delete files.
  3. Commit locally
     - Creates a new version.
  4. Push change back to server.

Single user



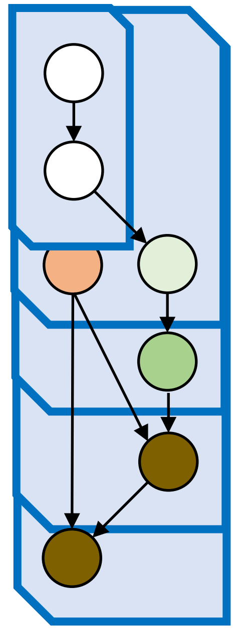
Review: `git` Basics (2 of 3)

- If a change happens on server:
  - We can "pull" it over, …
  - … as long our repo is consistent.
- If local change not committed
  - We can first "stash",
    - (saving our changes)
  - Then "pull" to update local repo,
  - And then "stash pop"
    - (restoring our changes)
- If local change committed
  - We will need to "merge" commits.



Review: `git` Basics (3 of 3)

- Development in a branch:
  - “fetch” to update repo;
  - Delay merges indefinitely.
- Merge main into branch:
  - Update branch to reflect changes;
  - Easier sooner.
- Merge into main:
  - Use work of branch;
  - Best if branch already up-to-date.



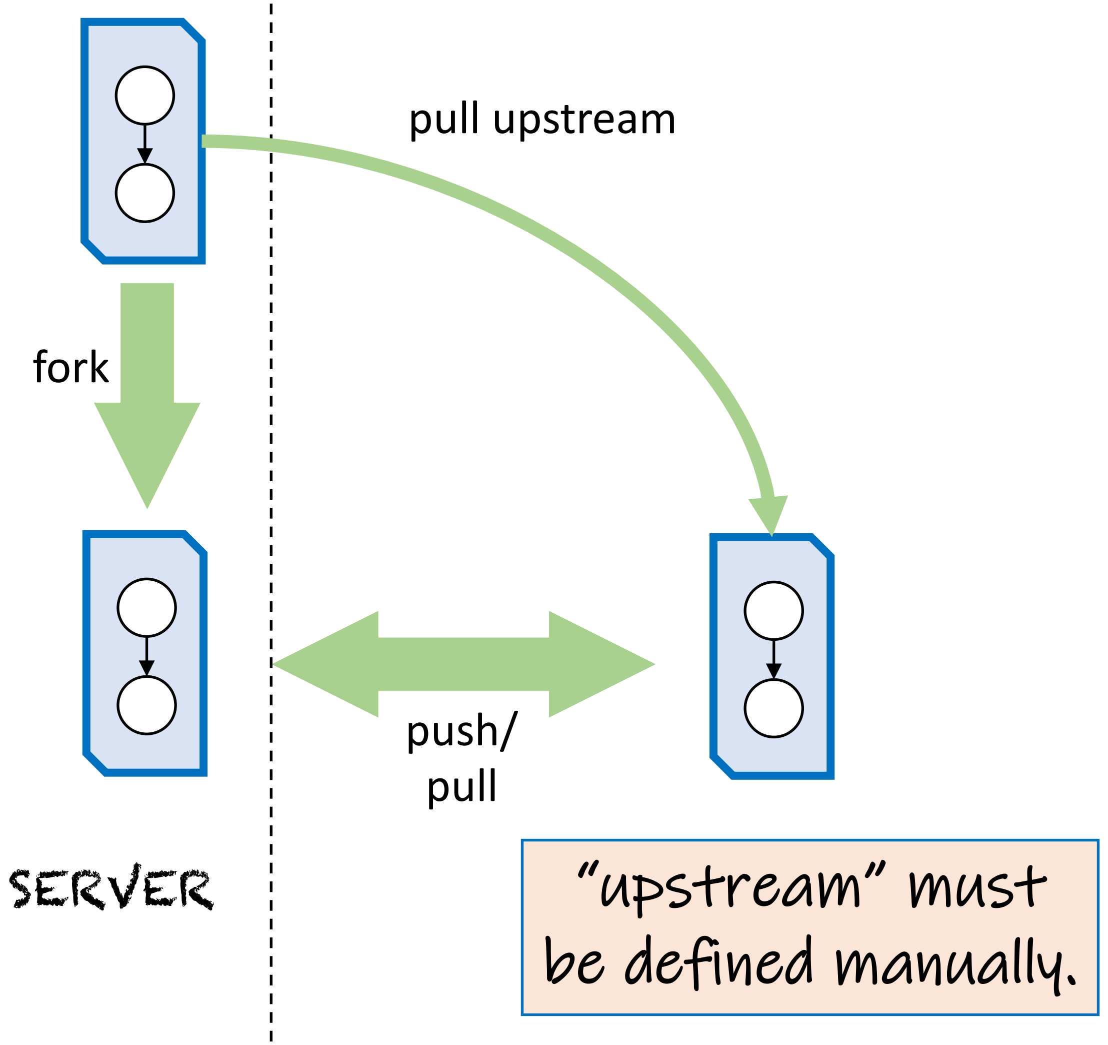
“main” sometimes called “master”

Using branches for collaboration

- Branches can be made by "insiders":
  - Branches can be pushed to the original repo;
  - Branches enjoy relative isolation;
  - Visible to other developers:
    - But usually extended only by a single developer;
  - They can be merged in or abandoned.
- Use a github "pull request" to request feedback:
  - Alert other developers of a change;
  - Courtesy only, since you could merge into main.

"pull request"
is a misnomer here

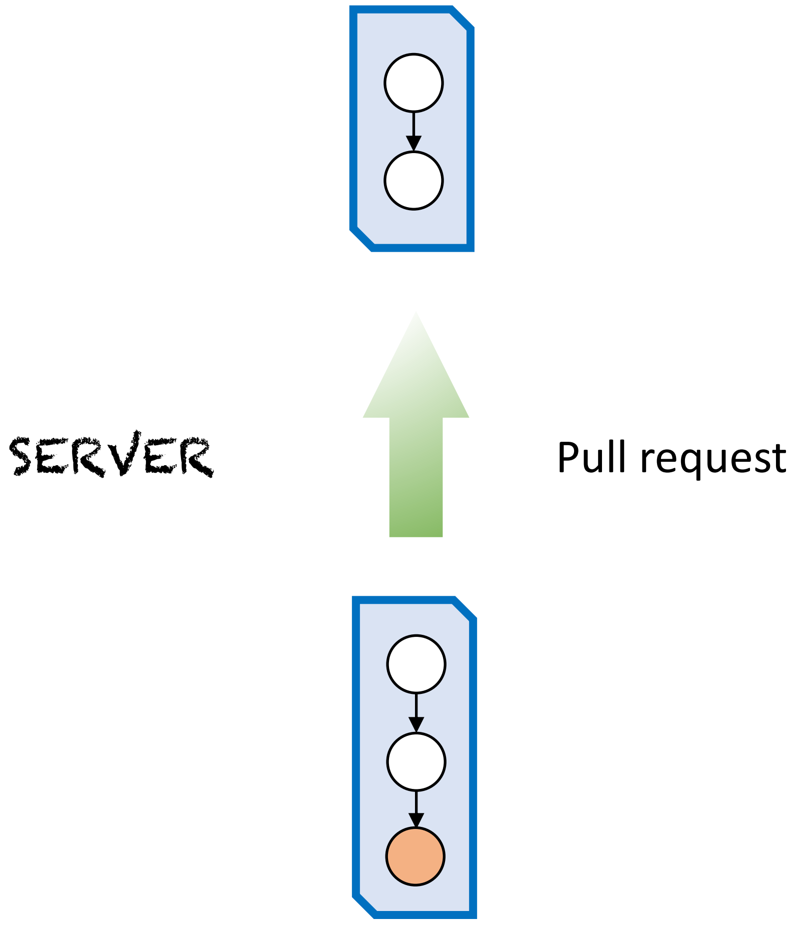
“Forks” are made by outsiders



“upstream” must be defined manually.

- A “fork” is a copy of a repo:
  - You don’t change original;
  - You can change your copy:
    - “push” changes you make.
  - Updates must happen through local clone pulling from upstream.
- Local clone has two “remotes”:
  - “upstream” is original repo;
  - “origin” refers to the fork.
- You can’t ”push” to the original!

“Pull Request” of the Copy

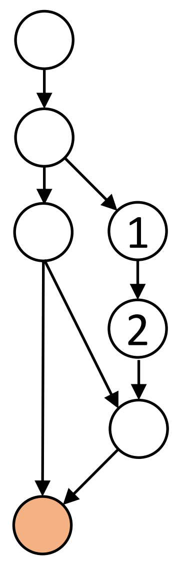


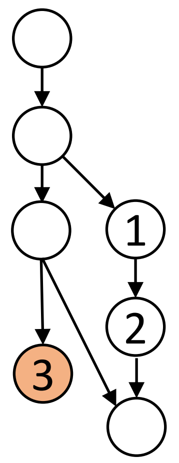
- The owner of the fork requests that the owner(s) of the original repo “pull” in changes.
- All done on the server
  - (e.g., github).
- Owner of fork should
  - Make sure fork is up-to-date;
  - Explain the reason for the change in the pull request.

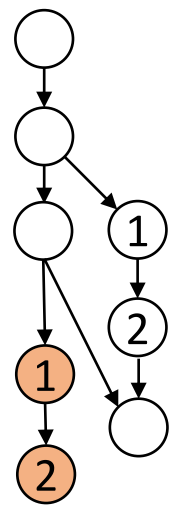
What to Do With a Pull Request

- The original owner(s) can
  - Examine all the changes;
  - Send comments back to the requester;
  - Request changes in the fork before approving;
  - Approve the request;
    - (Approval has no effect (yet) on the repo)
  - Close the request;
    - (effectively refusing the request).
- Or the pull request can be ignored
  - (not always the polite option).

Three Ways to Accept Pull Request







MERGE SQUASH REBASE

Normally "squash" is the best choice.

Branch problems

- Working in a branch helps avoid worrying about what’s going on in ”main,” but …
  - Delaying merges makes them harder;
  - Other developers might depend on your branch,
    - Or branch off your branch!
- So, keep branches short and merge back in quickly.
- Or, use the “monorepo” approach:
  - Do everything in “main” (no development branches)

  (Recommended in SE@Google, see Chapter 16)

Review: Learning Objectives for this Lesson

- You should now be able to:
  - Compare branches and forks on github;
  - Explain how branches and forks can be used for collaboration;
  - Describe the lifetime of a “pull request.”

Next steps…

- In our next lesson, we’ll talk about “Code Reviews.”