Git & GitHub Workflow

Before we start ...

Join The Course GitHub Organization

- **Important:** If you haven't registered to join our course GitHub organization, Do it now!
  - You must submit the form **by 10 pm tonight (Monday, Sep 19)**.
    If you fail to do so, you will not be able to work on A1 and receive a **mark of 0**.
  - You will get an email invite to join our organization by Tuesday morning.
    You must accept the invite **by 10 pm tomorrow (Tuesday, Sep 20)**.
    Once again, if you fail to do so, you will not be able to work on A1 and receive a **mark of 0**.
  - I will publish the assignment on Tuesday night.
- Please note that these deadlines are strict!
  The assignment is due this Friday, Sep 23, at 4 pm.
  Therefore, we cannot wait much longer with publishing the assignment.

Team Project

- During the next two weeks, you will form teams
  - 5-7 students per team
  - Can mix between sections, as long as you can make it to tutorials
  - I will publish a sign-up sheet later this week.
    (After we finish the individual registration to our GitHub organization)
  - **By Monday, Oct 3rd**, everybody should have a team.

Team Project

- Start thinking of ideas and/or look for people with interesting ideas
  - Focus on domains that you understand and/or are excited about.
  - Try to solve a real problem that you encountered in your daily/professional life.
  - Try to build something that you will actually use.
  - Try to be realistic (i.e. you need to be able to build a prototype).
  - Use the discussion board (or any other communication tool) to exchange ideas.
- Let your instructor/TA’s know if there is anything we can do to help.

Last week ...

Git - Review

- The basics
  - Unlike SVN, there is no single official repo.
  - Commits are snapshots (not just deltas)
- You start with one of the two options:
  - `git init` - Create a new repo
  - `git clone` - Create a clone of an existing repo

Git - Review

- Basic commands:
  - `git add` - Add a file to the staging area
  - `git commit` - Commit all added changes
- These commands run locally
  - You can commit (i.e. save) changes, even if you are not connected to the Internet.
- Getting info:
  - `git status`
  - `git log`

Git/GitHub Workflow - Review

Last week we saw the following workflow:

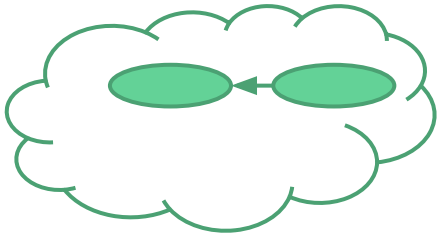
- In order to contribute to a public repo, you
  - *Fork* the public repo
  - *Clone* the fork to you local machine
  - *Commit* changes locally
  - *Push* them to your fork
  - Create a *pull-request*
  - Somebody (with write permission to the original repo) *merges* your pull-request.

This week ...

Git/GitHub Workflow

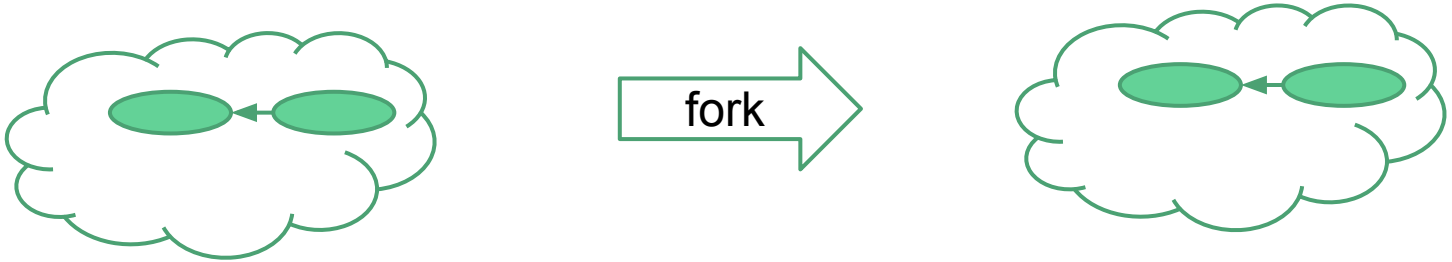
Let’s see a few more common workflows …

Another Git/GitHub Workflow



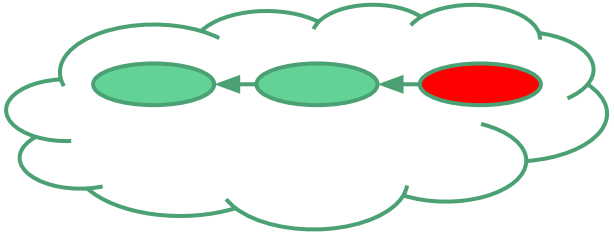
Project repo, publically readable, only maintainers and core devs can write (in the case of your assignment, only the instructors can write)

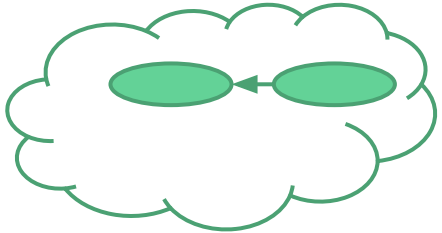
Another Git/GitHub Workflow



Fork (hosted on GitHub servers) is readable and writable by you.

Another Git/GitHub Workflow





Suppose the original repo changes.

Ex: The instructor finds a bug in the assignment's starter code, and push commits into your handout repo.

Your fork is out of date. Now what?

Another Git/GitHub Workflow

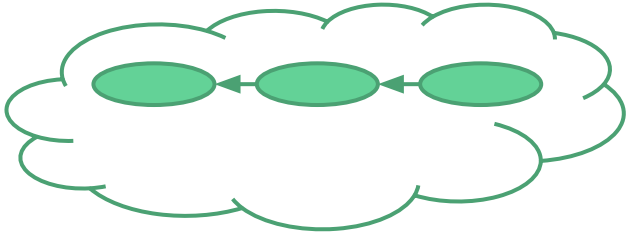
You can create a pull-request (from the project repo to your fork)

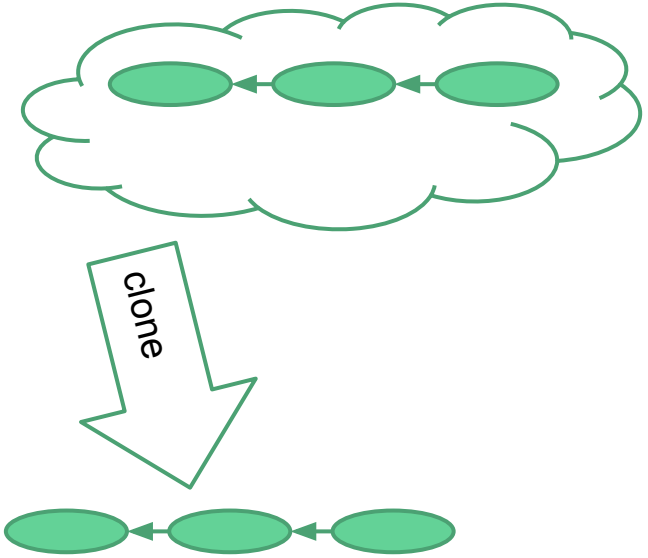
Another Git/GitHub Workflow

- You have write permission ⇒ you can merge the PR
- This case is easy:
  - No conflicts (you didn't change your fork)
  - Git can easily merge the pull-request by simply adding the red commit to the fork.
  - Can be done entirely using GitHub's web UI.
- **Important:** This does NOT mean that your code won't break!
  - Example: method name change

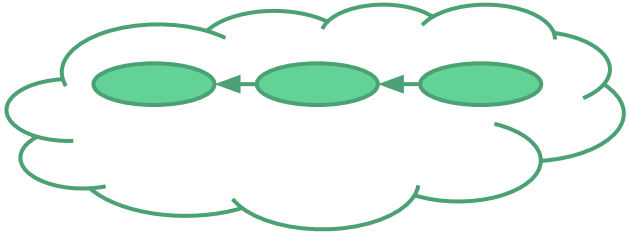
Git/GitHub Workflow

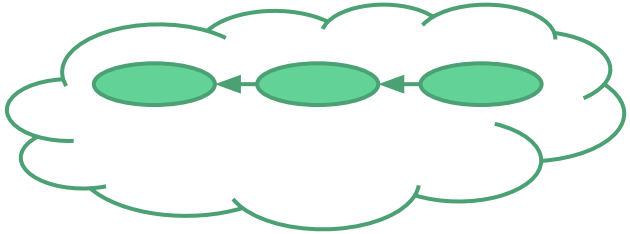
Let’s look at a slightly less simple case, where you start working on the code locally ...



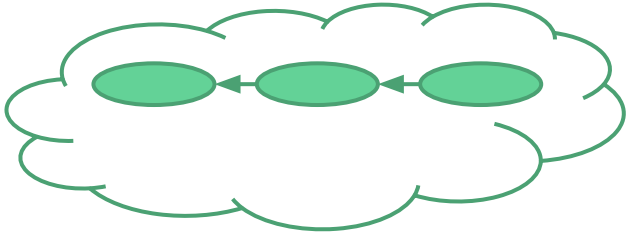


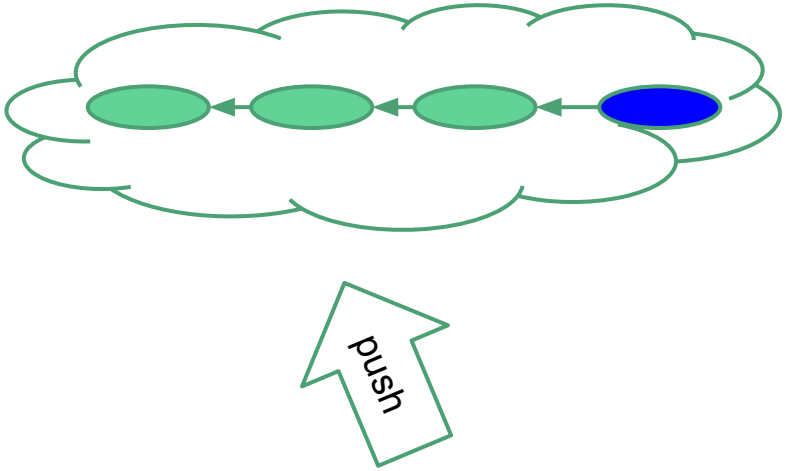
Local clone of our fork. (Now we can work on our code)





Great! Local commits.
Making progress!



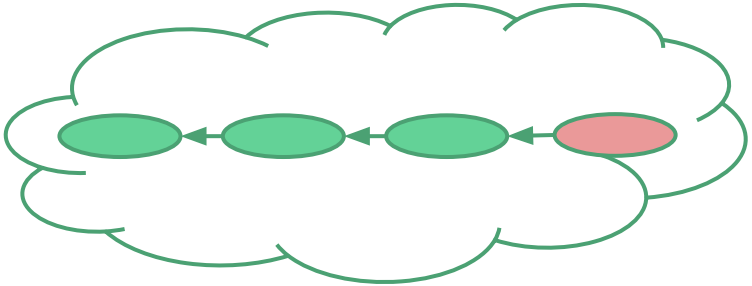


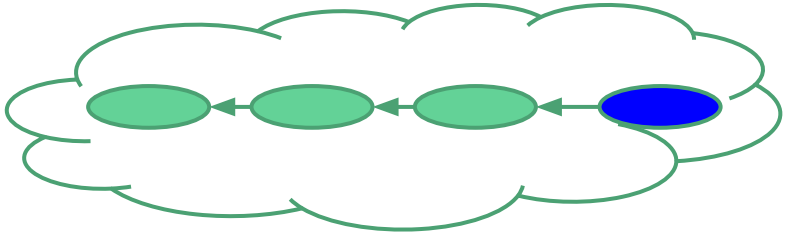
Push WIP (Work-In-Progress) to your fork as you commit - Use your fork as a backup.

Handout changes again..

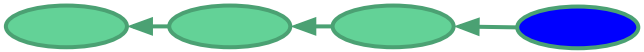
- Once you have pushed commits into your fork, dealing with upstream changes can become more complicated

Handout changes again..



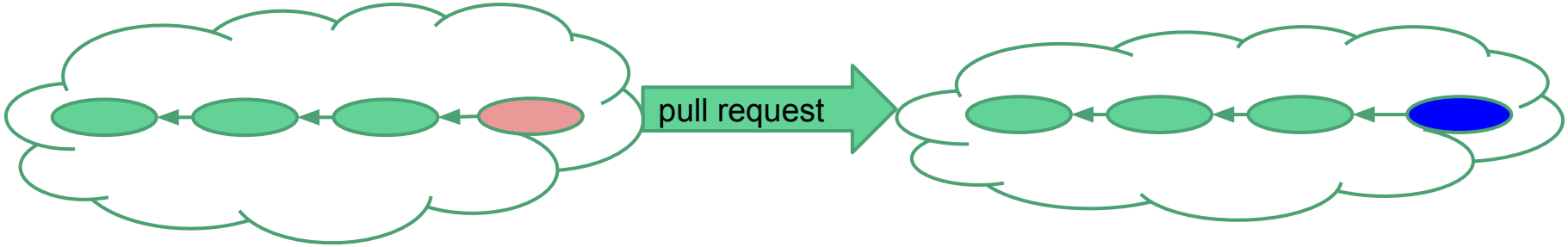


Can we submit a pull-request?

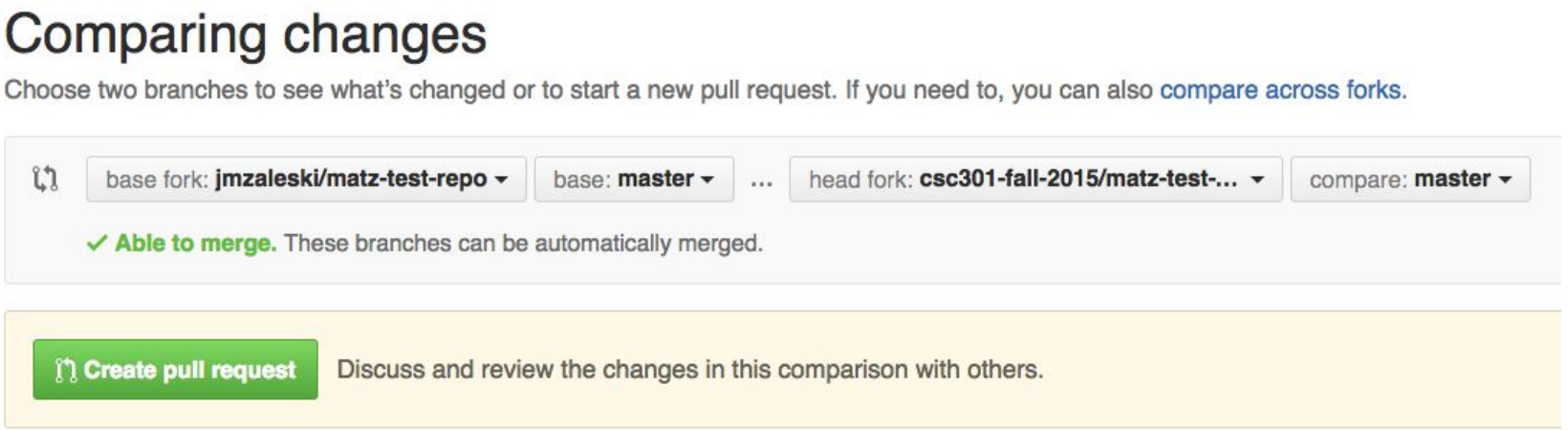


Case 1 - No Conflicts

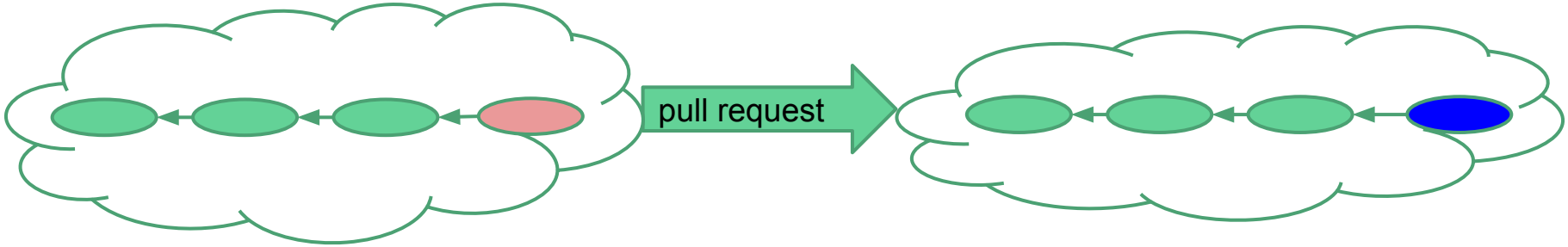
- Suppose the red and pink commits (from the previous slide) change different files.
- Totally separate changes ⇒ no chance of conflict
- This is the easy case.



Here is what it looks like on GitHub, when you go to create the pull-request:

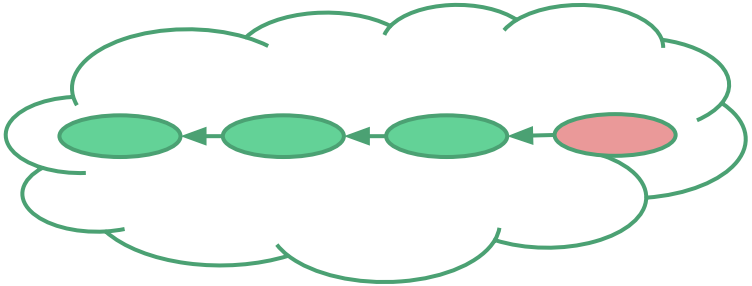


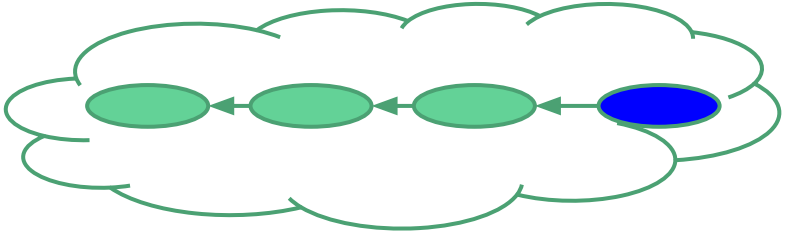
NOTICE: On GitHub, the “destination repo” is on the left, and the “source repo” is on the right. (Personally, I find it a bit confusing)



Because there are no conflicts, you can merge the PR using GitHub's web UI:

Case 2 - Conflicts

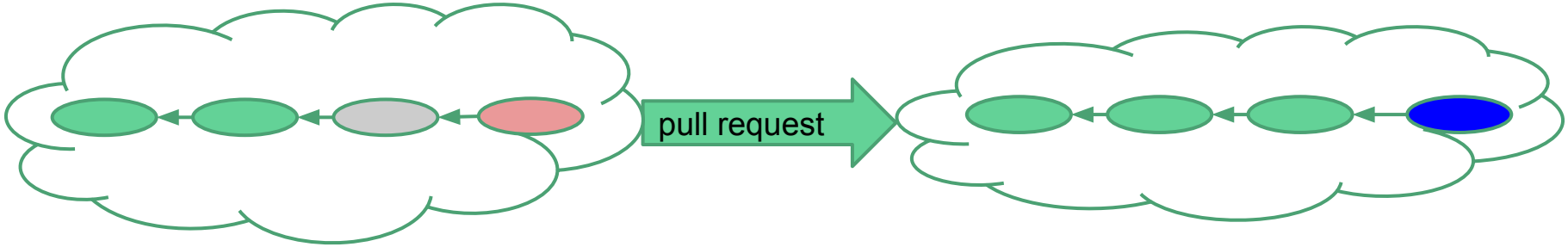




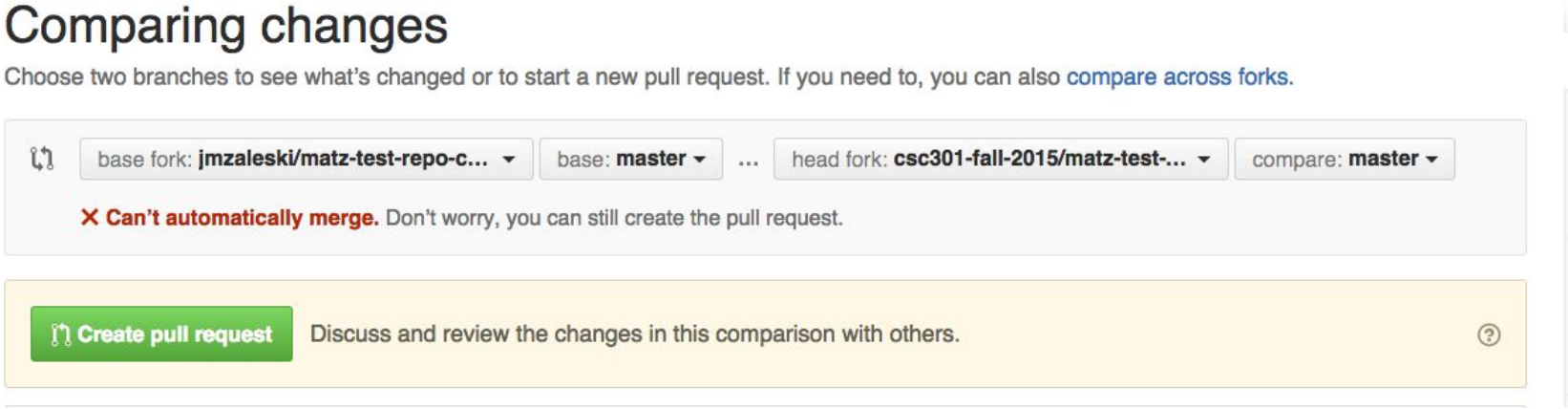
Suppose the pink and blue commits make different changes to the same line of the same file.

Case 2 - Conflicts

- Conflict = Two commits change the same line(s) in the same file(s).
- Requires human intervention
  - Decide “who wins”
  - perhaps combine the two somehow

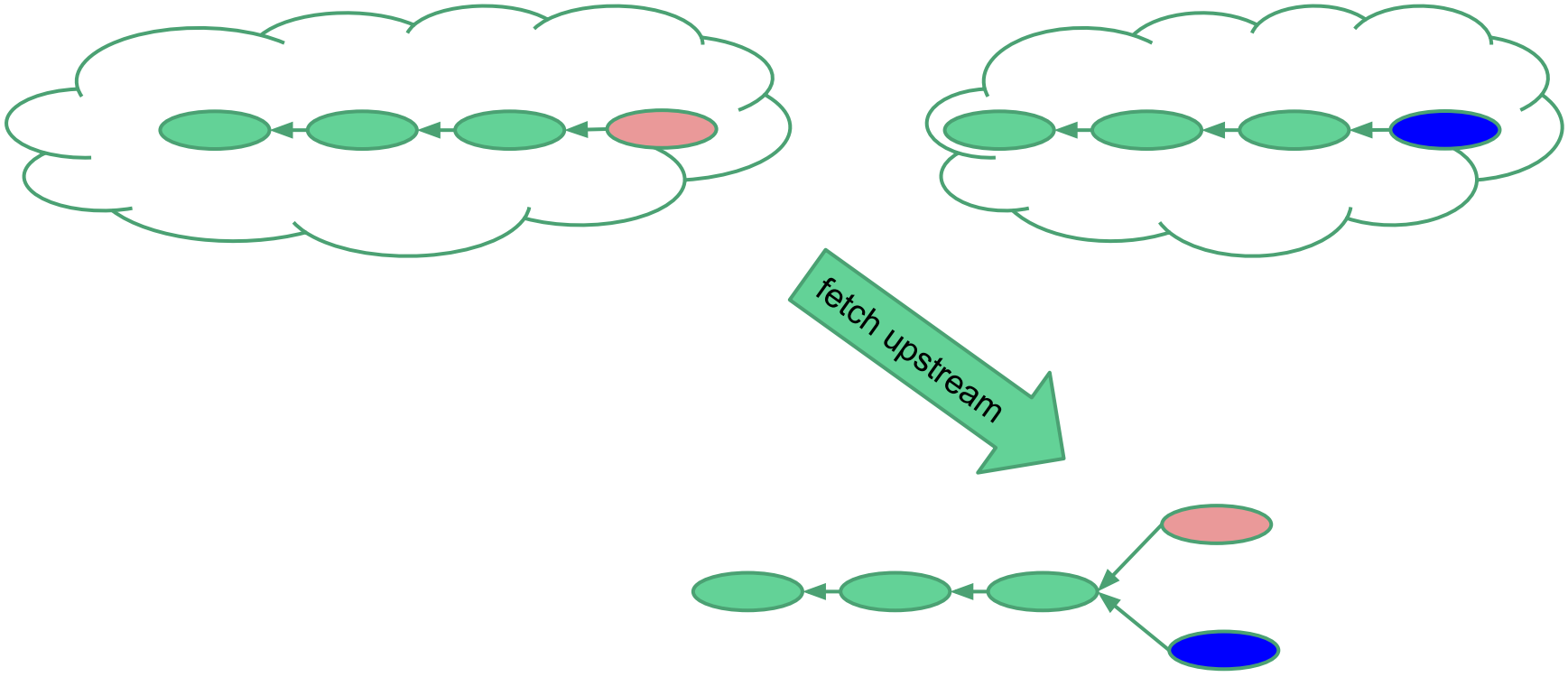


You can still create the pull-request, you just won't be able to merge it from the website.



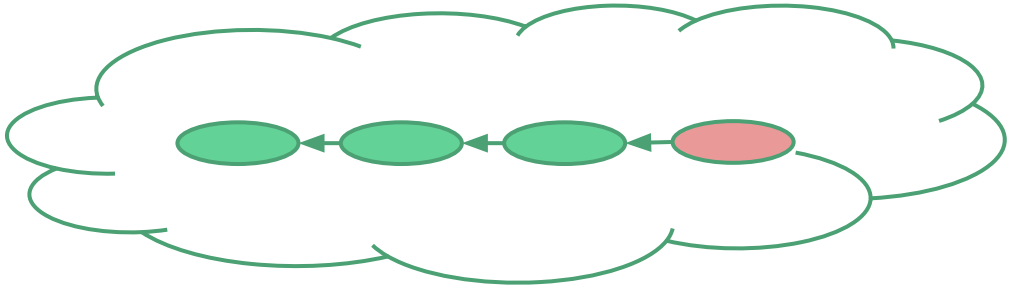
Resolve conflicts

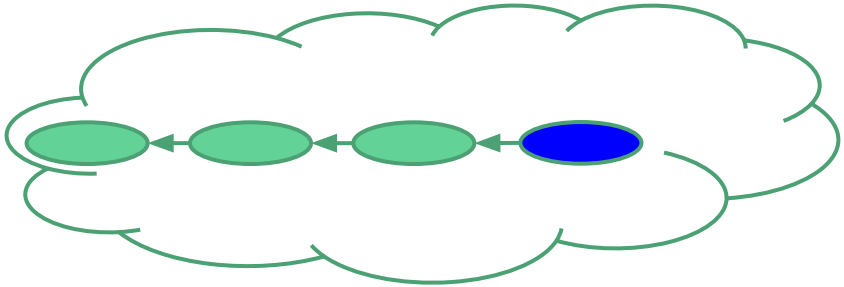
1. Fetch changes from the handout into the local clone.

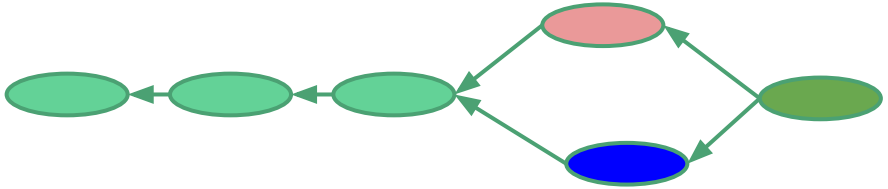

fetch upstream

Resolve conflicts

1. Fetch changes from the upstream into the local clone.
2. Merge changes locally

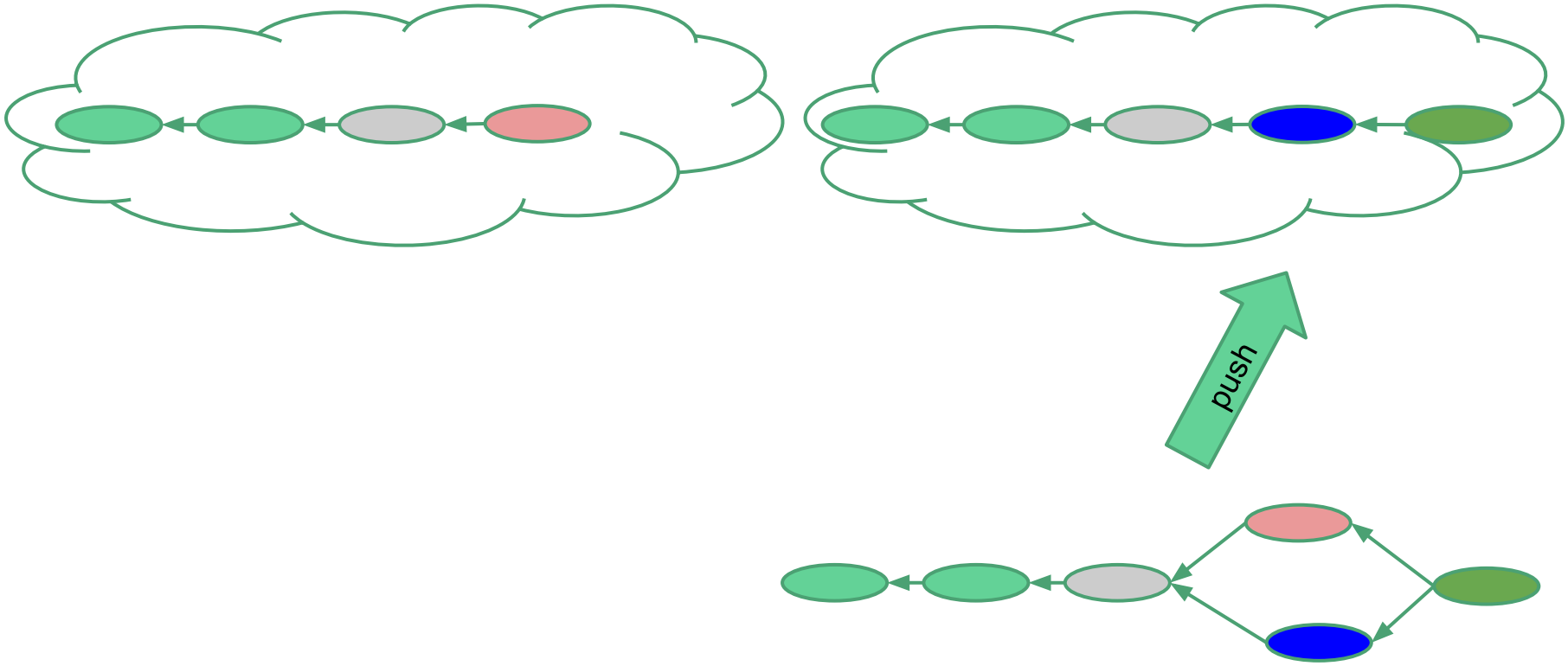






Resolve conflicts

1. Fetch changes from the upstream into the local clone.
2. Merge changes locally.
3. Push the final result to the fork


push

Resolve conflicts

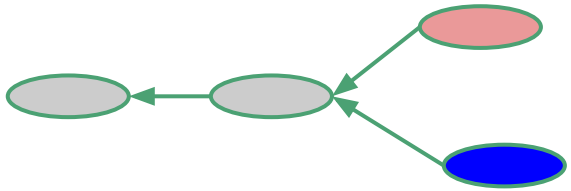
Github gives a nice, two part recipe

1. Configure the “source” (aka upstream) repo
   (This step only happens once per local clone).

2. Fetch (and merge) changes locally

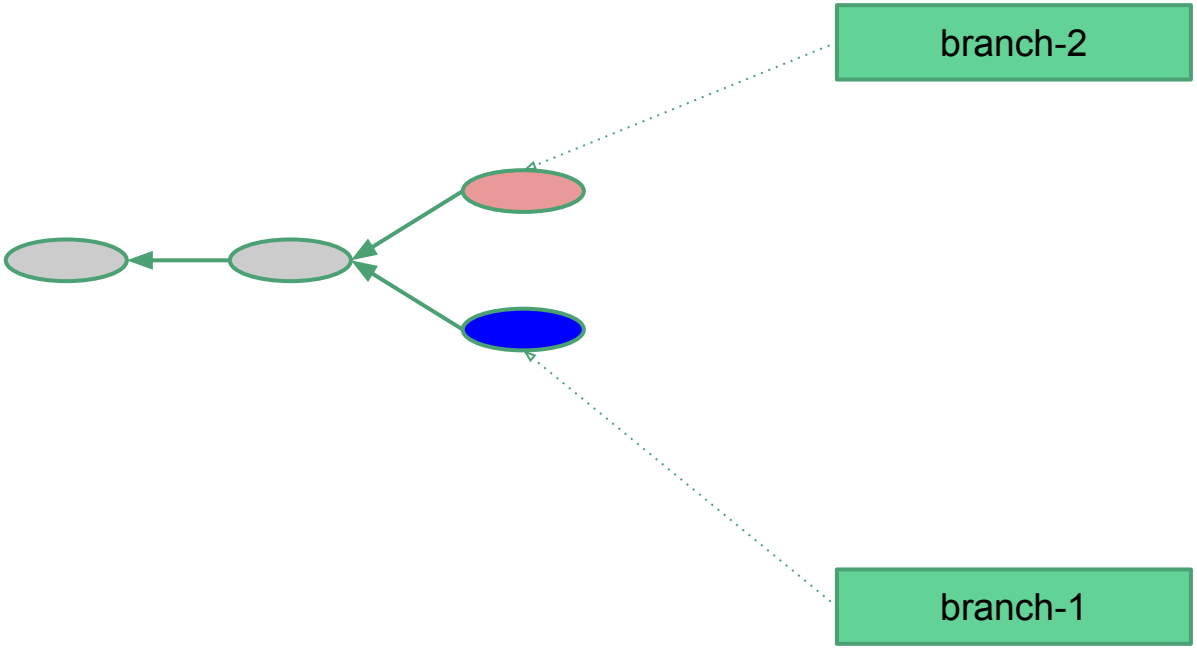
Branching

- In the previous slides, Git implicitly created a branch



- A repo is a graph of commits.
- Branches are just a (named) reference to a commit in the graph.

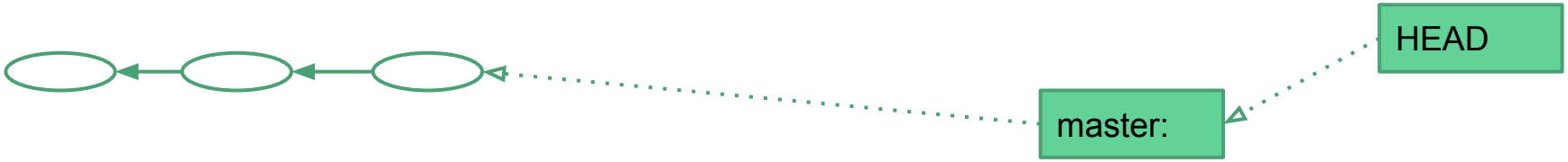
Branching



Branching is very handy for WIP

- Branching in Git is very often used as fancy undo
  - Checkout a (new) branch
  - Fool around a bit
  - Merge it or discard
- Jargon warning:
  - Git “checkout” switches branches.
  - Potentially confusing for svn users!

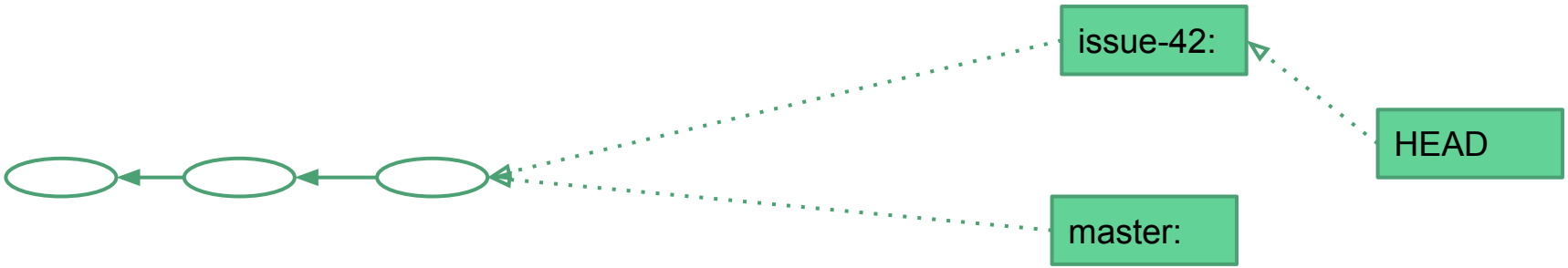
Branching



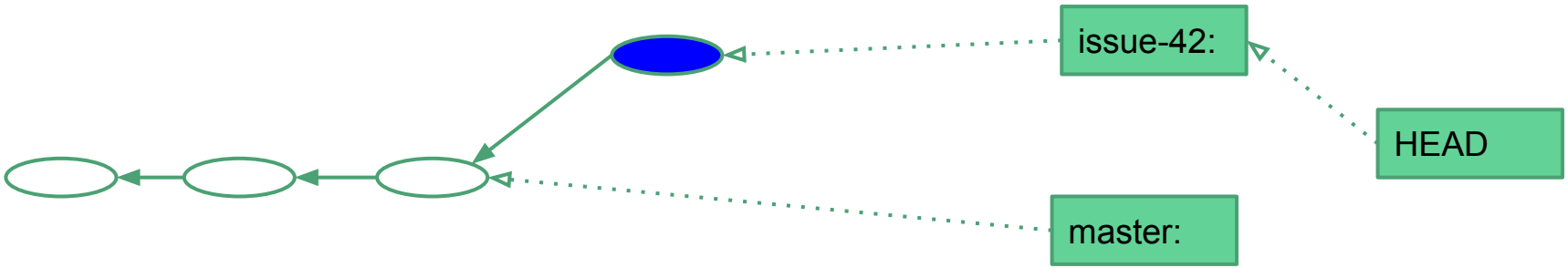
Branching - Creating A Branch

```
git checkout -b issue-42
```

- New branch points to the same commit
- `HEAD` to the tip of the current branch we're working on

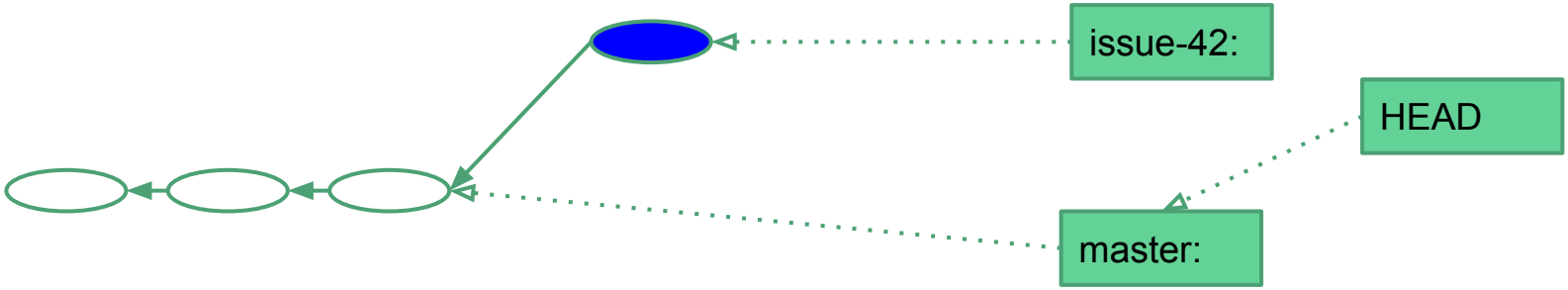


Branching - Commit Changes In A Branch



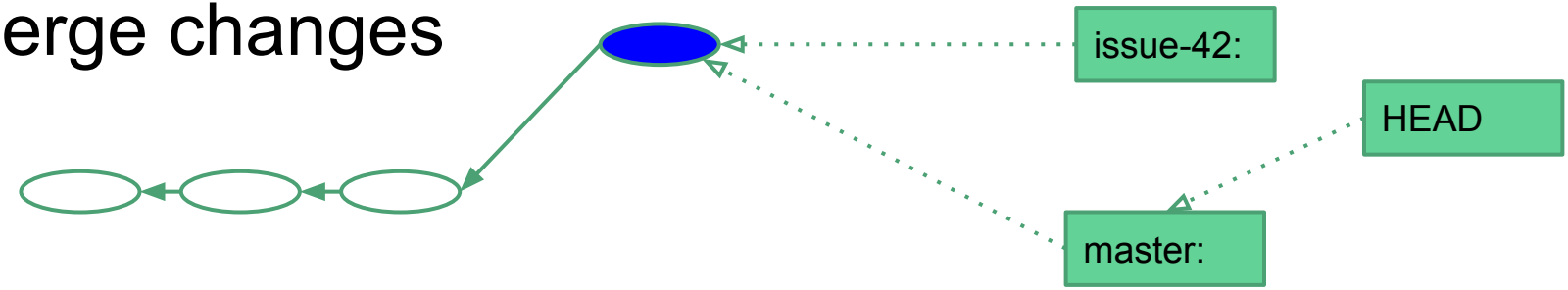
Branching workflow

- Create a new branch:
  - `git checkout -b issue-42`
- Make your changes in branch
- Checkout master



Branching workflow

- Create a new branch:
  - `git checkout -b issue-42`
- Make your changes in branch
- Checkout master
- Merge changes



Merging Branch locally is stealthy

- Issue: Branch was merged locally ⇒ No traces of that branch in your fork.
- What if you want the branch to appear in the fork?
  - More accurate history
  - Perhaps you want your teammate to review the branch, *before* you merge it.

Pull Request great for code review

1. Rather than simply merging your feature branch locally, push it to the fork.
2. Submit a pull request from the feature branch to the master branch,
    - Above, we saw a pull request across forks.
        - That is, master branch of one repo to master branch of another repo.
    - Equally effective between branches

Pull Request great for code review

- Common workflow:
  - You create a PR between two branches in your fork
  - Your partner reviews the PR, makes comments, etc.
  - You merge the PR after fixing the issues your partner has raised.
- Pull-requests are NOT a snapshot, they are “open requests” to pull changes from one branch to another.

Many Options

1. Push changes directly from (local) master to (fork's) master.
2. Use branches locally (as fancy undo), merge locally, and push from master to master.
3. Use branches remotely, with the full collaboration tools available to you (Pull-requests, issues)

Github Issue tracking

- A list of work
  - Bugs, features, design improvements, etc
  - An open issues represents a task to be completed
  - Can assigned to users, labeled, and much more ...
- GitHub has an excellent guide
- Improve traceability
  - Links code changes (commits) to a discussion that explains why the changes was made.

Choose commits strategically

- As you master Git, you will start thinking strategically about committing code so that the commit history will tell a coherent story.
- What work should be a reviewed PR?
    - Hint: significant feature or bug fix
- What work should be a simple commit to master?
    - Hint: Minor changes like comments
- Advanced technique: Squash commits
    - Combine a bunch of commits into one, before pushing.

Your Assignment

- In your first assignment, you will use GitHub as follows:
  - Each student will have a read-only, private repo.
    We call this repo the handout repo.
  - You will fork the handout, and work on the fork
    - Push commits (from your local machine to your fork on GitHub)
    - Optionally, open issues to keep track/document of your progress
  - Submit your solution as a PR
  - After the deadline, your PR will be merged