

Git & GitHub Workflow

Before we start ...

Join The Course GitHub Organization

- If you haven't registered to join our course GitHub organization, do it now!
 - Final deadline: **Wednesday, Jan 17, at 4 pm**
 - If you fail to do so, you will not be able to work on A1 and receive a **mark of 0**.
- Please note that these deadlines are strict!

Last week ...

Git - Review

- You start with one of the two options:
 - `git init` - Create a new repo
 - `git clone` - Create a clone of an existing (remote) 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:

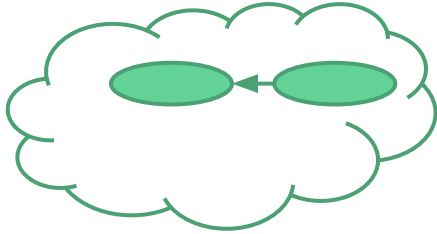
- In order to contribute to a public repo, you
 - *Fork* the public repo
 - *Clone* the fork to you local machine
 - *Commit* changes locally
 - *Push* them to your fork
 - Create a *pull-request*
 - Somebody (with write permission to the original repo) *merges* your pull-request.

This week ...

Git/GitHub Workflow

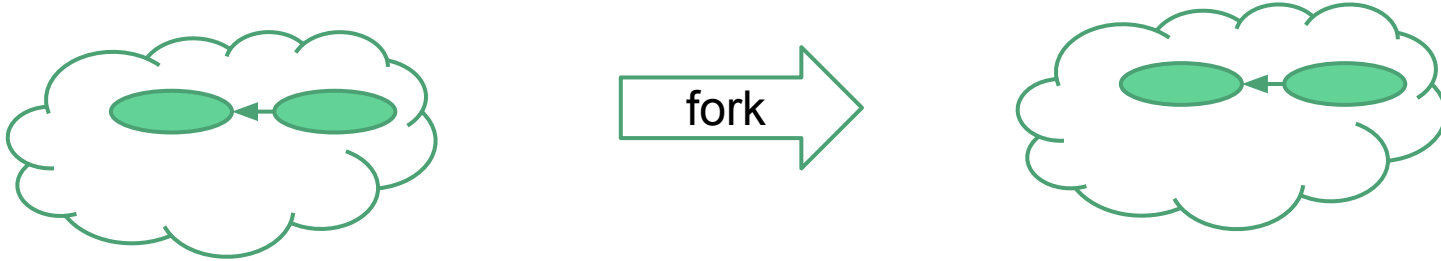
Let's see a few more common workflows ...

Another Git/GitHub Workflow



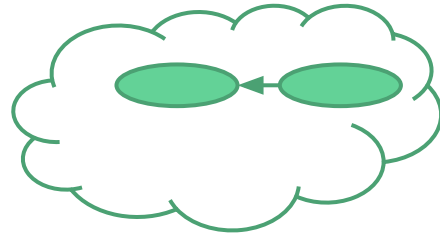
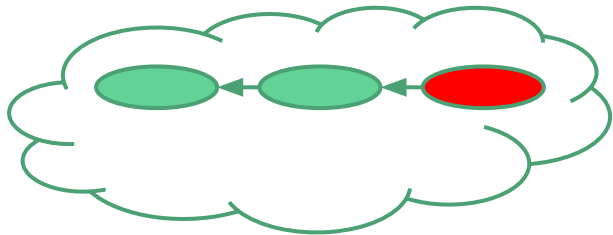
Project repo, publicly readable, only maintainers and core devs can write (in the case of your assignment, only the instructors can write)

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 pushes 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)

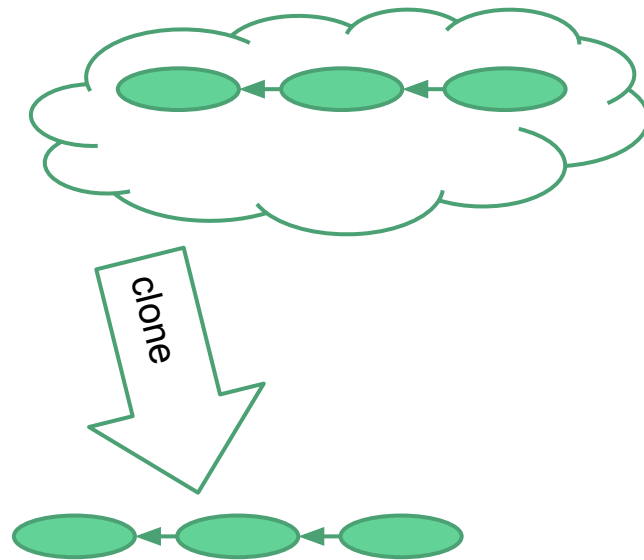
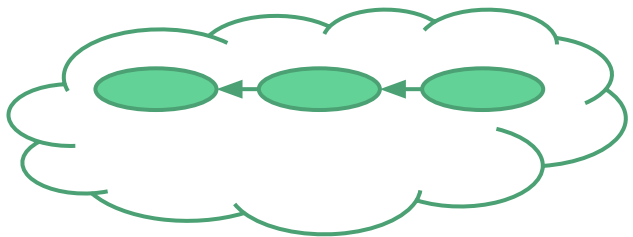
Another Git/GitHub Workflow



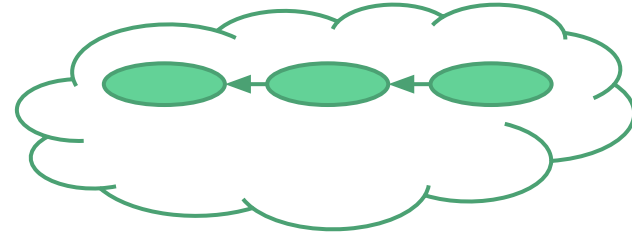
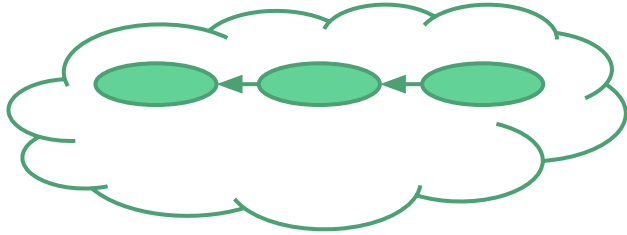
- You have write permission \Rightarrow you can merge the PR
- This case is easy:
 - No conflicts (you didn't change your fork)
 - Can be done entirely using GitHub's web UI.
- **Important:** This does NOT mean that your code won't break!
 - Example: method name change

Git/GitHub Workflow

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

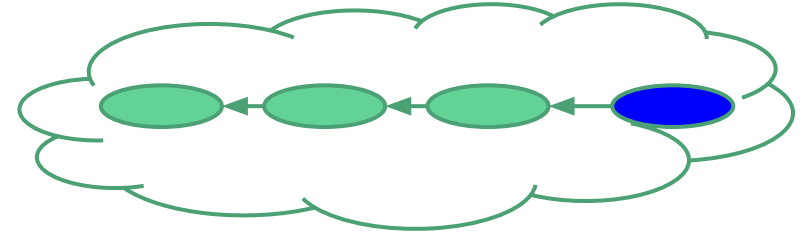
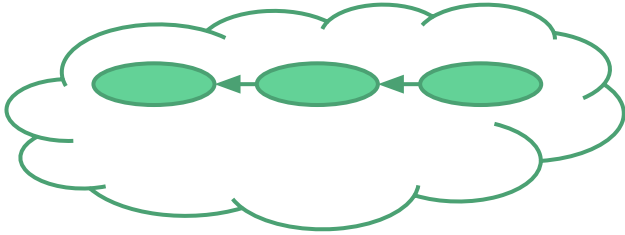


Local clone of our fork. (Now we can work on our code)



Great! Local commits.
Making progress!



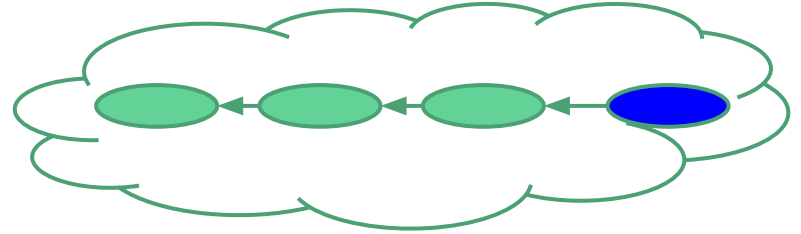
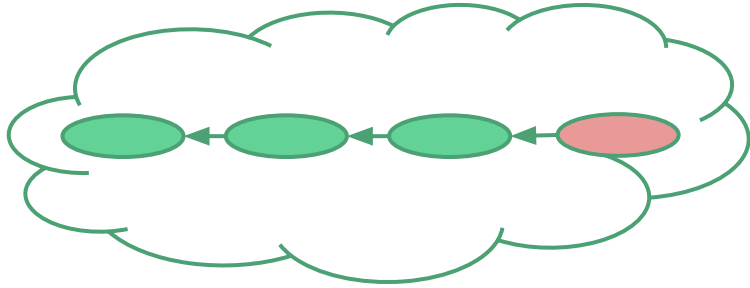


Push WIP (Work-In-Progress) to your fork as you commit - Use your fork as a backup.

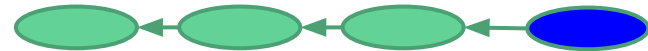
Handout changes again..

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

Handout changes again..

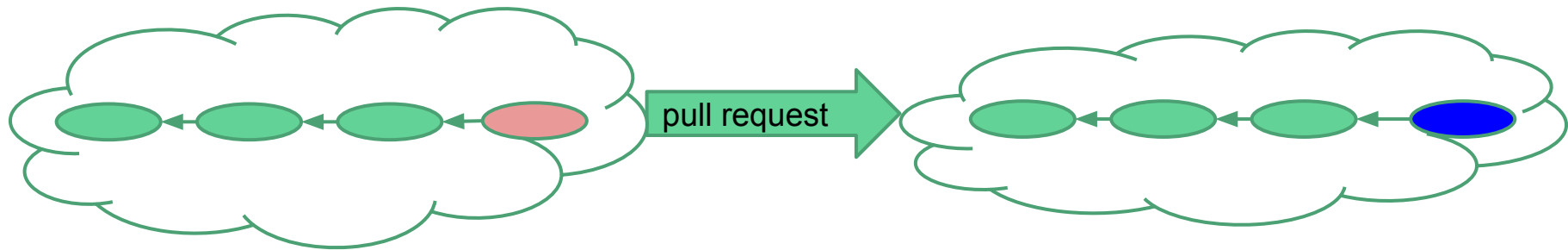


Can we submit a pull-request?



Case 1 - No Conflicts

- Suppose the red and pink commits (from the previous slide) change different files.
- Totally separate changes \Rightarrow 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:

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#).

base fork: forcedotcom/force-dot-com-es...
base: master

...

head fork: davidatkooltra/force-dot-com-...
compare: master

✓ Able to merge. These branches can be automatically merged.

Create pull request

Discuss and review the changes in this comparison with others.

?

↶ 1 commit

📄 1 file changed

💬 0 commit comments

👤 1 contributor

Commits on Sep 13, 2017

davidatkooltra

Trivial change

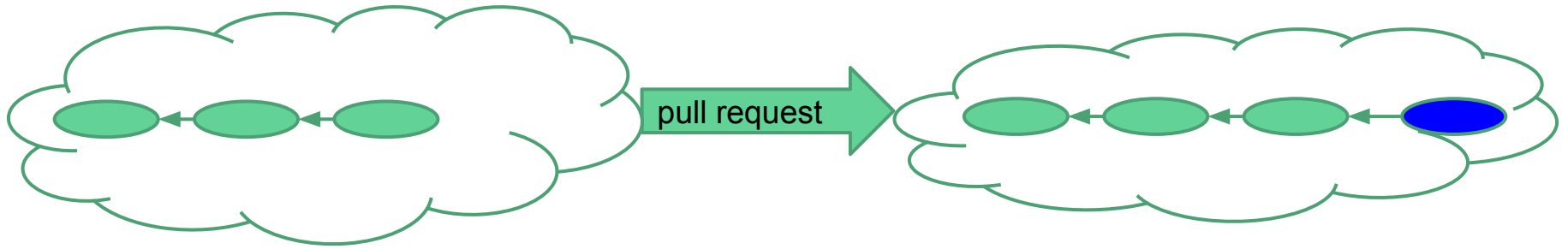
63f4bcd

Showing 1 changed file with 1 addition and 1 deletion.

Unified

Split

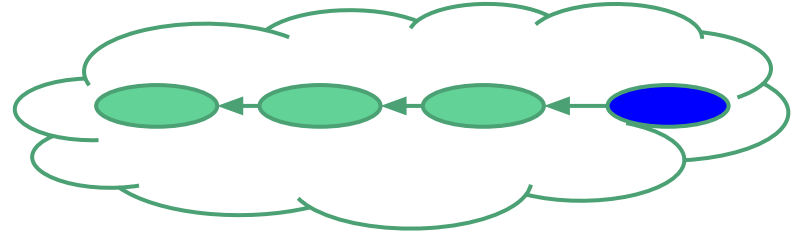
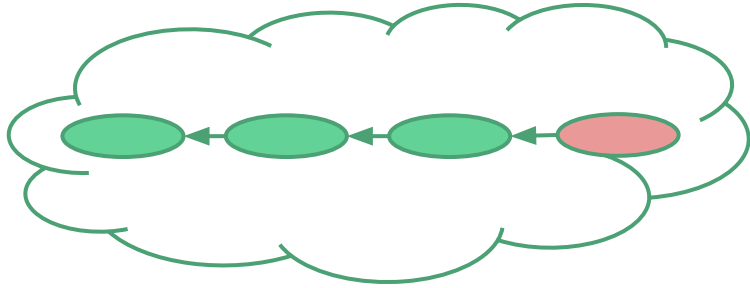
NOTICE: On GitHub, the “destination repo” is on the left, and the “source repo” is on the right.



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

The screenshot shows the GitHub web interface for merging a pull request. On the left is a green icon with a white branching diagram. The main content area has a light green header with a checkmark icon and the text: **This branch is up-to-date with the base branch** and *Merging can be performed automatically.* Below this is a green button with a white branching icon and the text **Merge pull request**. To the right of the button is the text: *You can also [open this in GitHub Desktop](#) or view [command line instructions](#).* The phrase [command line instructions](#) is circled in red.

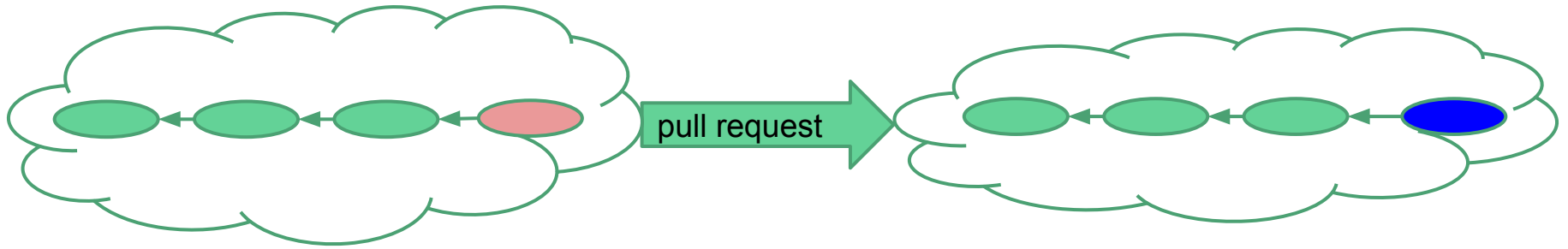
Case 2 - Conflicts



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 need to resolve the conflict before merging.

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#).

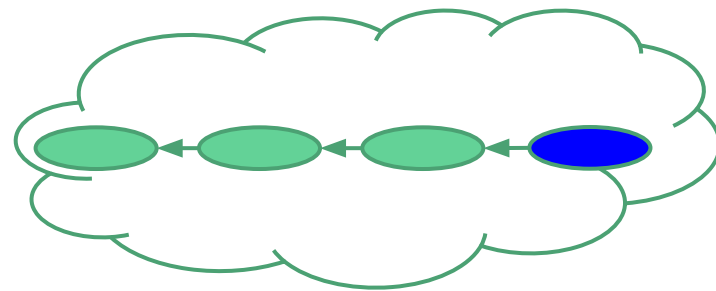
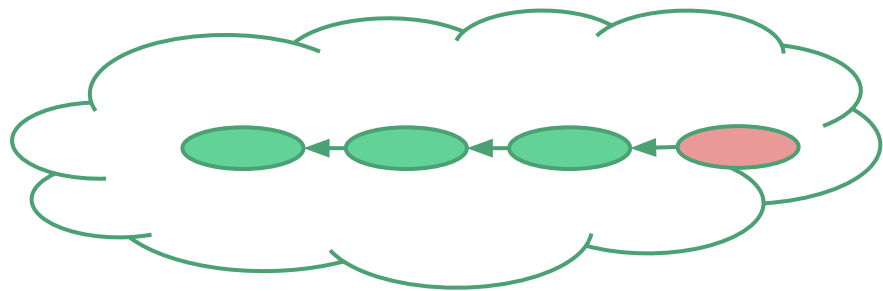
 base fork: **jmzaleski/matz-test-repo-c...** ▼ base: **master** ▼ ... head fork: **csc301-fall-2015/matz-test-...** ▼ compare: **master** ▼

✗ Can't automatically merge. Don't worry, you can still create the pull request.

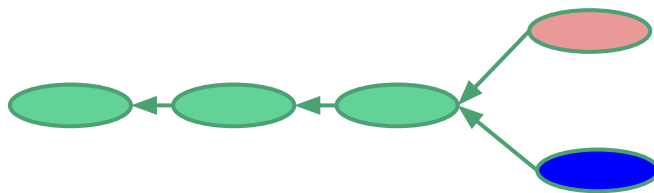
 **Create pull request** Discuss and review the changes in this comparison with others. 

Resolve conflicts

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

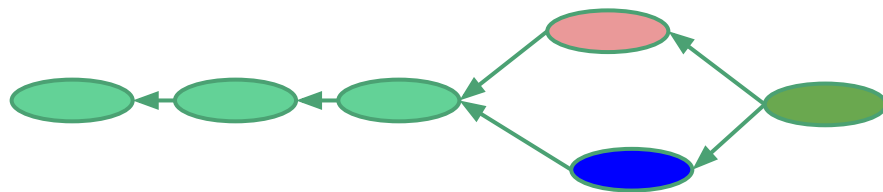
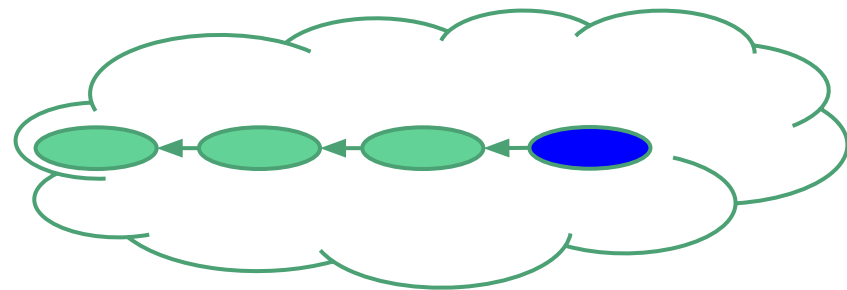
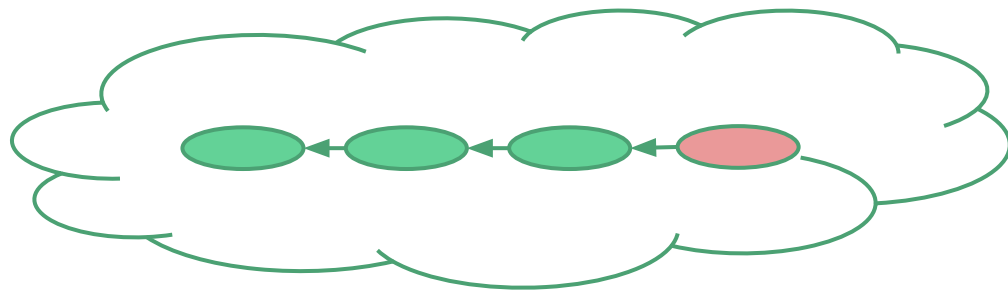


fetch upstream



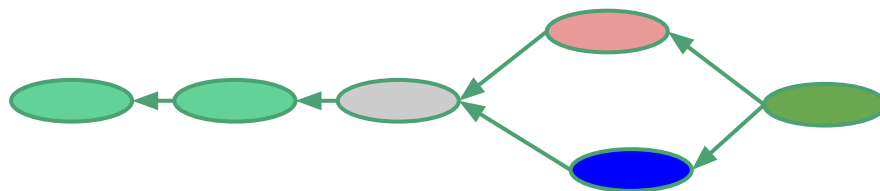
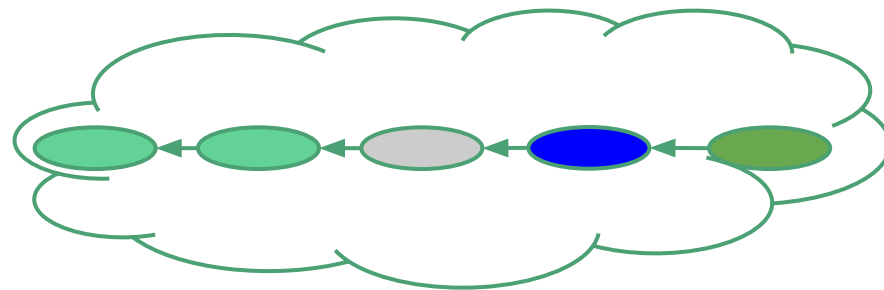
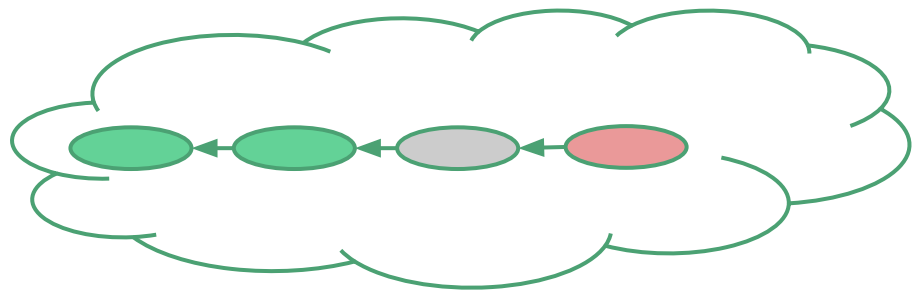
Resolve conflicts

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



Resolve conflicts

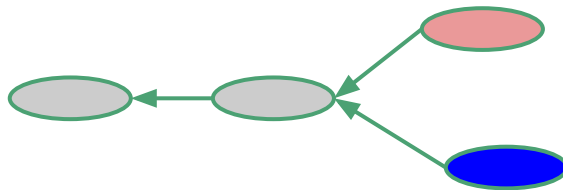
Github gives a nice, two part recipe

1. Configure the “source” (aka upstream) repo
(This step only happens once per local clone).
2. Fetch (and merge) changes locally

Suggestion: Practice this recipe with a classmate.

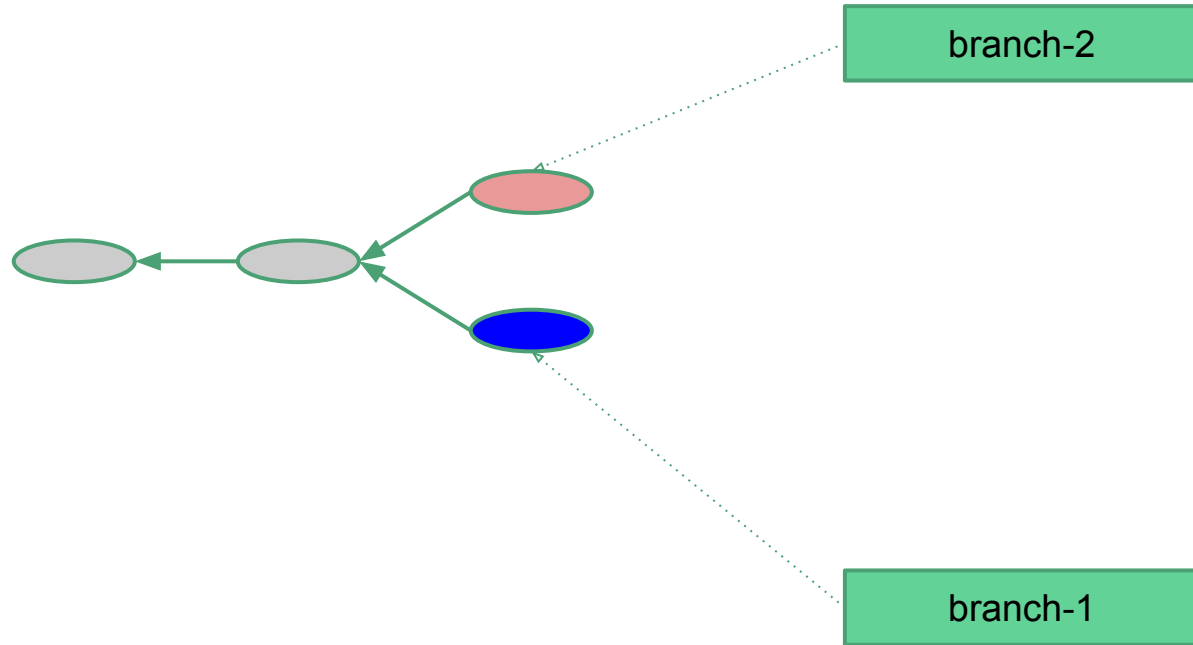
Branching

- 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!

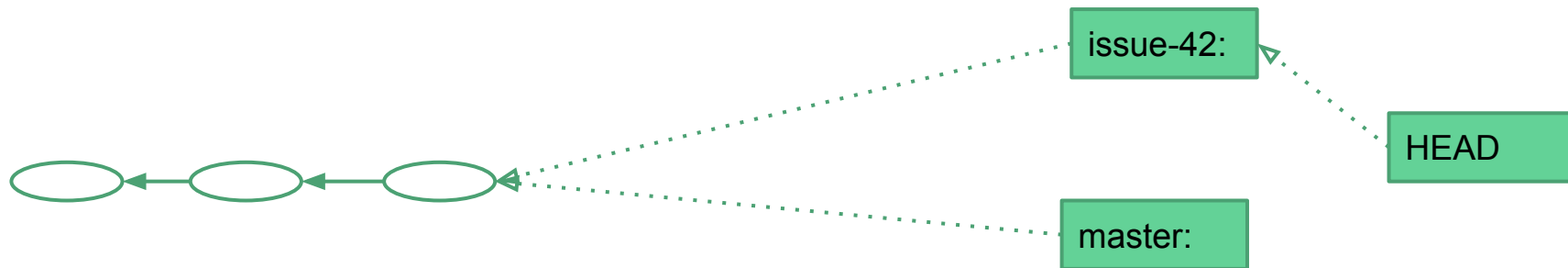
Branching



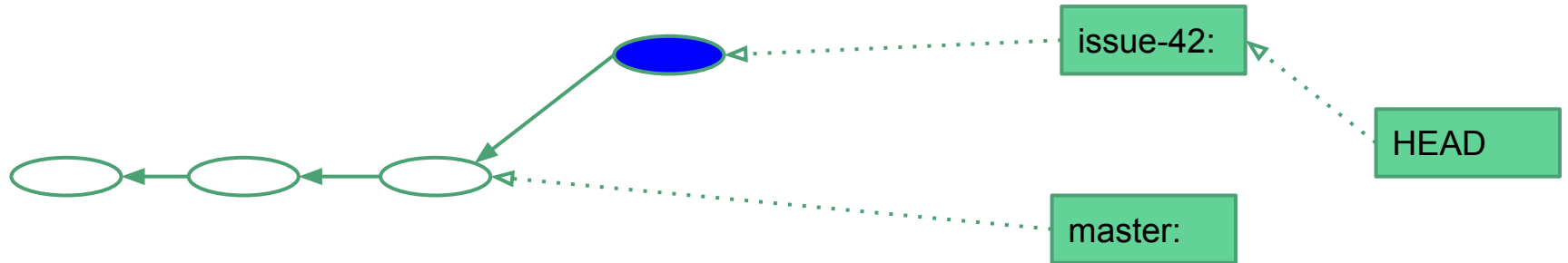
Branching - Creating A Branch

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

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

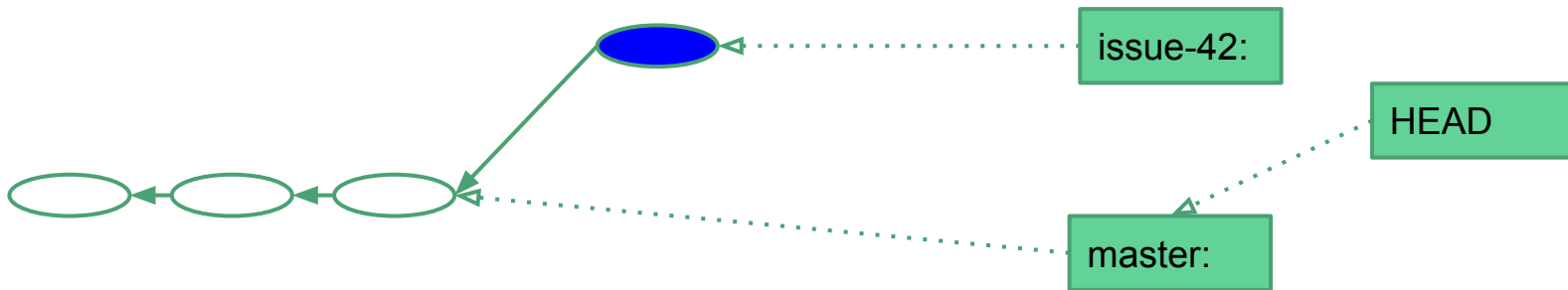


Branching - Commit Changes In A Branch



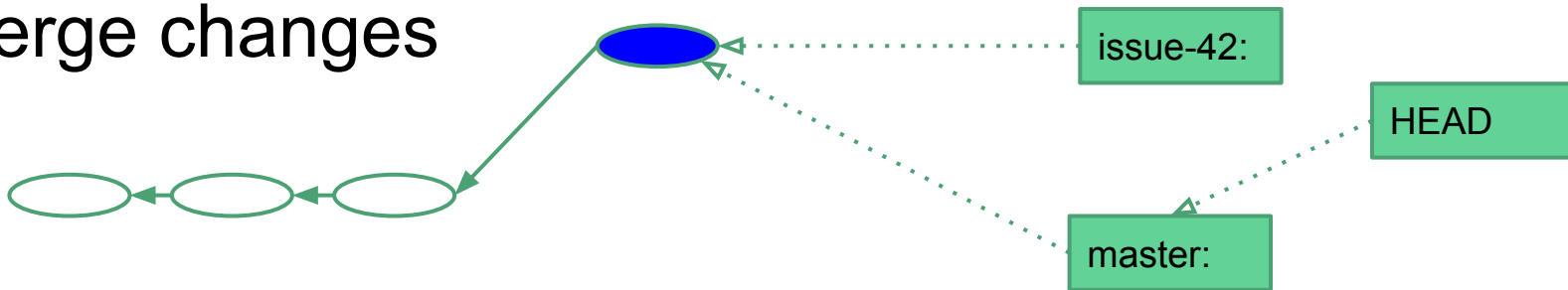
Branching workflow

- 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 Branches

- When merging (one branch into another), there are (usually) 3 cases:
 - Fast forward
 - Changes only happened in one branch (since the branching point)
 - Git performs the merge by moving a pointer
 - Recursive
 - The changes in the two branches are NOT conflicting
 - Git performs the merge by creating a new commit (with two parents, and all changes from both branches)
 - Fix conflicts manually
 - Changes in the branches are conflicting
 - Git adds markers to source files to indicate where the conflicts are
 - The developer fixes the conflicts by hand and commits the changes

Merging Branch locally is stealthy

- Issue: Branch was merged locally \Rightarrow 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 GitHub.
2. Submit a pull request from the feature branch to the master branch,
 - Previously, we saw a pull request across forks.
 - That is, master branch of one repo to master branch of another repo.
 - Works equally effective between branches in the same repo.

Pull Request great for code review

- Common workflow:
 - You create a PR between two branches in your fork
 - Your teammate(s) review the PR, makes comments, etc.
 - You fix whatever needs to be fixed, based on your teammates' review
 - Once everybody is happy, someone on the team merges the PR
- 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 (remote) master.
 - The remote repo may or may not be a fork.
As long as you have write permission to a repo, you can push changes to it.
2. Use branches locally, merge locally, and push from master to master.
3. Use branches remotely, with the full collaboration tools available to you (Pull-requests, issues)

General advice: keep master as a stable source of truth.

Github Issue tracking

- A list of work
 - Bugs, features, design improvements, etc
 - An open issues represents a task to be completed
 - Can be 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?
 - Ex: significant feature or bug fix
- What work should be a simple commit to master?
 - Minor changes such as comments, typos in documentation, blank lines, etc.
- Advanced techniques (Optional):
 - Fix the latest commit with `git commit --amend`
 - Rewrite history using `git rebase`.

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

Team Project - Today

- 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 (or you know other users will).
 - Try to be realistic (i.e. you need to be able to build an MVP).
 - 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.
- We will publish a sign-up sheet later this week.
(After we finish the individual registration to our GitHub organization)