GitHub for Developers

Squares Conference 2016

Joel Glovier

Lead Product Designer, GitHub

@jglovier

Workshop agenda

- An overview of GitHub
- GitHub Flow
- Collaborating with other developers
- A quick look at Git commands
- Q&A

An overview of GitHub

The difference between Git and GitHub

GIT

An open source, distributed version control system

GITHUB

A private company that provides a Git hosting platform

The difference between Git and GitHub

Git

- git-scm.com
- version control system
- sandbox changes via branches
- distributed VCS

GitHub

- github.com
- provides Git repository hosting
- most popular place to share open source projects
- back up your repositories
- makes collaborating with other developers easier

Repositories

- Where work on a specific project takes place
- Your project root
- You can have public and private repositories
  - Public means anyone can see your project, and fork it
  - Private means only you and collaborators you invite can see your project
- Can belong to a personal account, or an organization account
- Supports a GitHub Pages site

Cloning

- Copying a repository from GitHub (the "remote server") to your local machine
- You can clone from the command line, or with GitHub Desktop

Branching

- A branch is an environment to try out new ideas
- Branching is a core concept of Git and GitHub Flow
- The purpose of a branch is to provide a safe place to make changes to code without breaking your stable code
- Work happens on a branch, and gets merged into another branch (e.g. master) via a pull request
- Only 1 Rule: Master should always be deployable
- GitHub Pages uses a special branch called `gh-pages` to host your code
- Learn more at: https://guides.github.com/introduction/flow/

Commits

- Commits are logged changes to the code
- The items that make up your project history
- Whenever you add, edit, or delete a file, you're making a commit, and adding it to your branch.
- You can commit from the command line, from GitHub Desktop, or from GitHub.com
- Commit messages are a great way to make your project history easy to follow for other developers
- Learn more at: https://guides.github.com/introduction/flow/

Issues

- Issues are a feature of GitHub.com
- They are independent from your stored Git repository information
- For keeping track of tasks, planning features, and logging bugs
- Provides a persistent and reference-able place for conversations
- Can be opened by anyone with access to your repository
- Can reference other issues
- Can be closed from commits or pull requests
  - “Closes #23” or “Addresses #18”
- Supports Markdown, and file/image attachements
- Learn more at: https://guides.github.com/features/issues/

Pull Requests

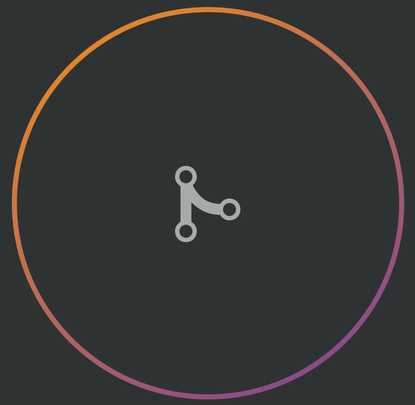
- Used to request changes from one branch be merged into another (e.g. the master branch, or any other branch)
- Pull Requests initiate discussion about your commits
  - A feature of GitHub.com, not Git
  - You can @mention other people or teams
- Useful for contributing to open source projects and for managing changes to shared repositories
- Fundamental to GitHub Flow
- Learn more at: https://guides.github.com/introduction/flow/

Forking

- Creating an independent copy of another project
  - Useful for contributing to open source projects
  - Can be updated from “upstream”
- Useful if you want to customize a project beyond the scope of the original project (e.g. Electron is a fork of Chromium, a customized Node package, etc)
- Used just like any other repository, except it has an upstream link to another repository
- Learn more at: https://guides.github.com/activities/forking/

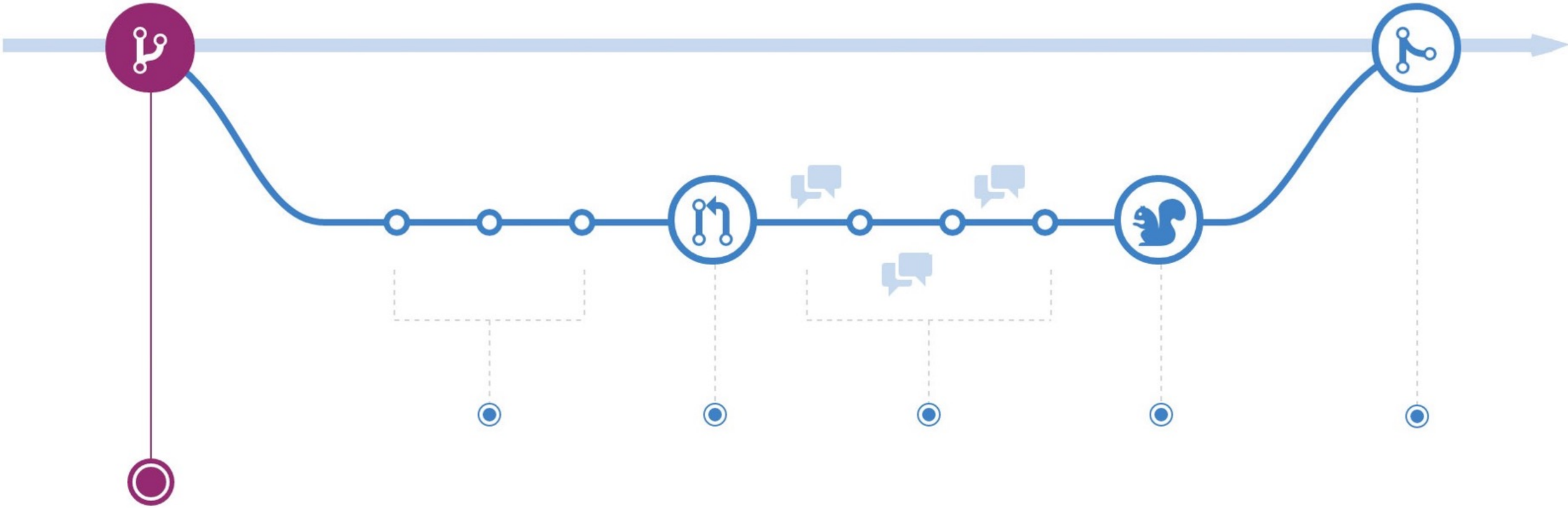
Git Operations: Terminal vs GitHub Desktop

- The terminal is the default place where you operate Git
- GitHub Desktop is a GUI that abstracts some of the complexities of Git and allows you to use GitHub flow without the Terminal
- GitHub Desktop is not a full fledged Git client, so there are many Git operations which you cannot perform with GH Desktop



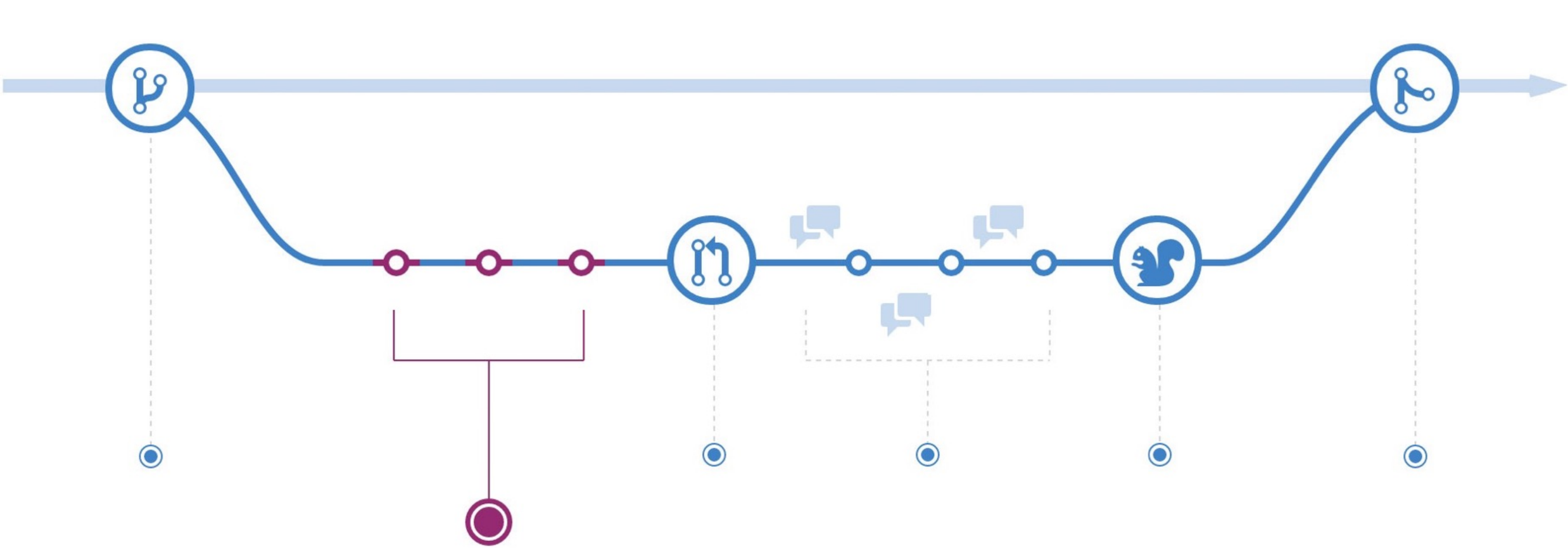
GitHub Flow

GitHub Flow



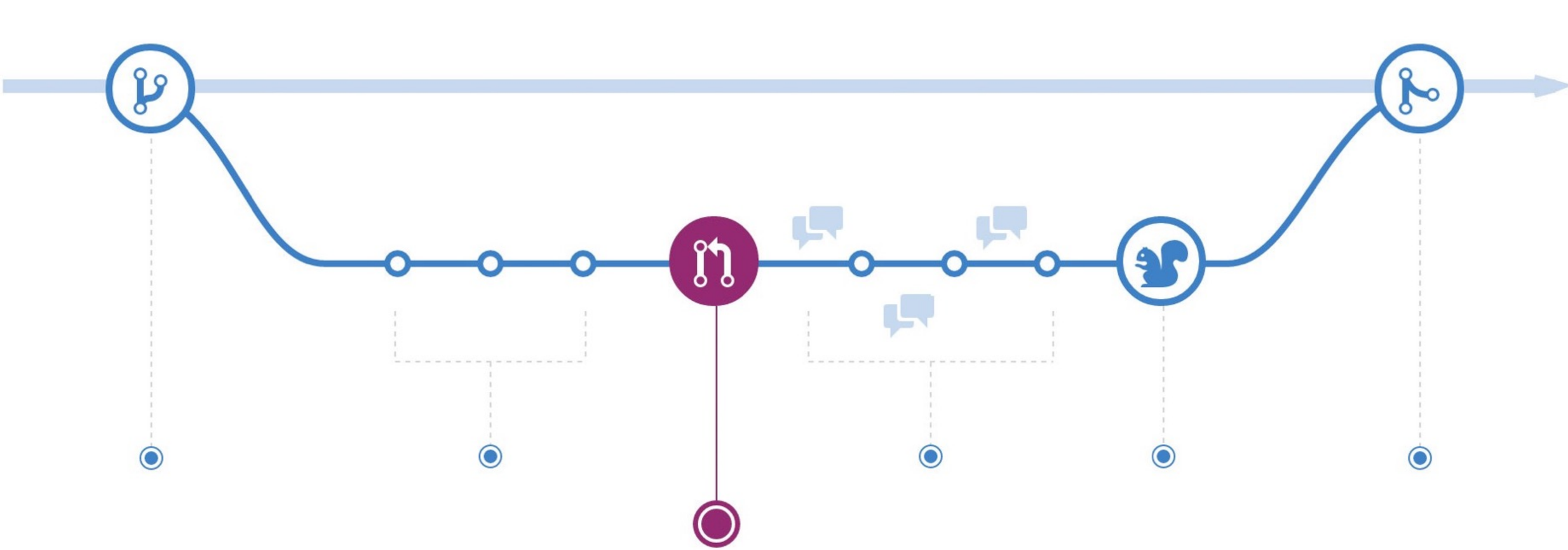
Step 1 Branch from master

GitHub Flow



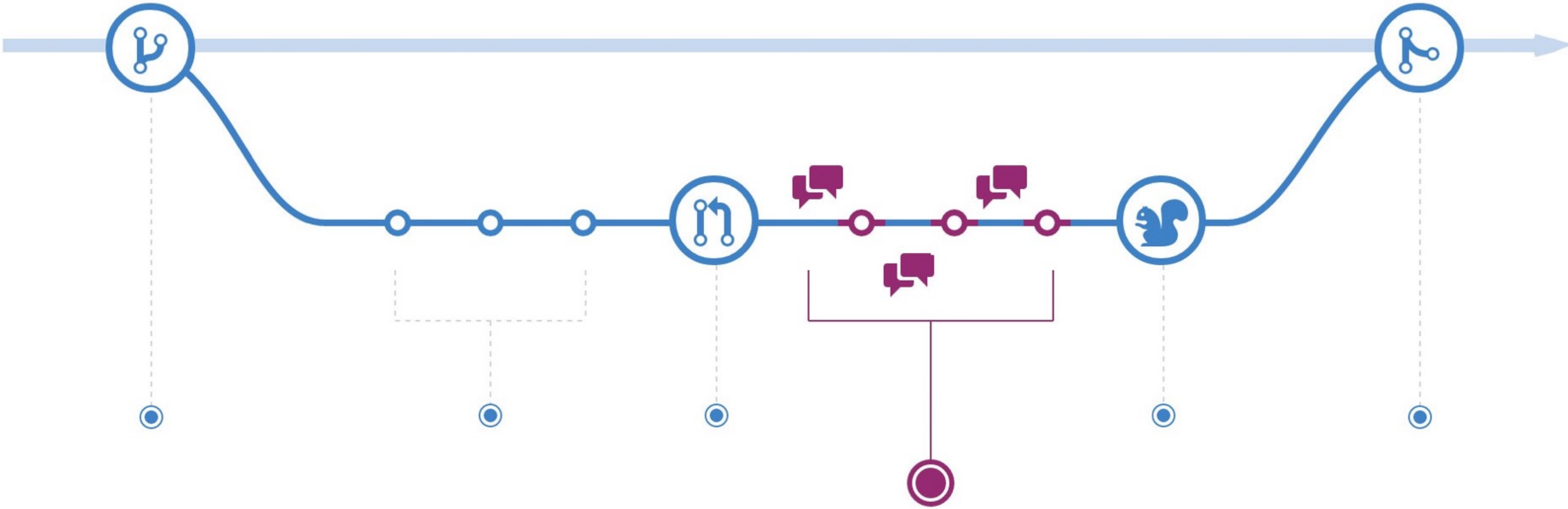
Step 2 Add commits

GitHub Flow



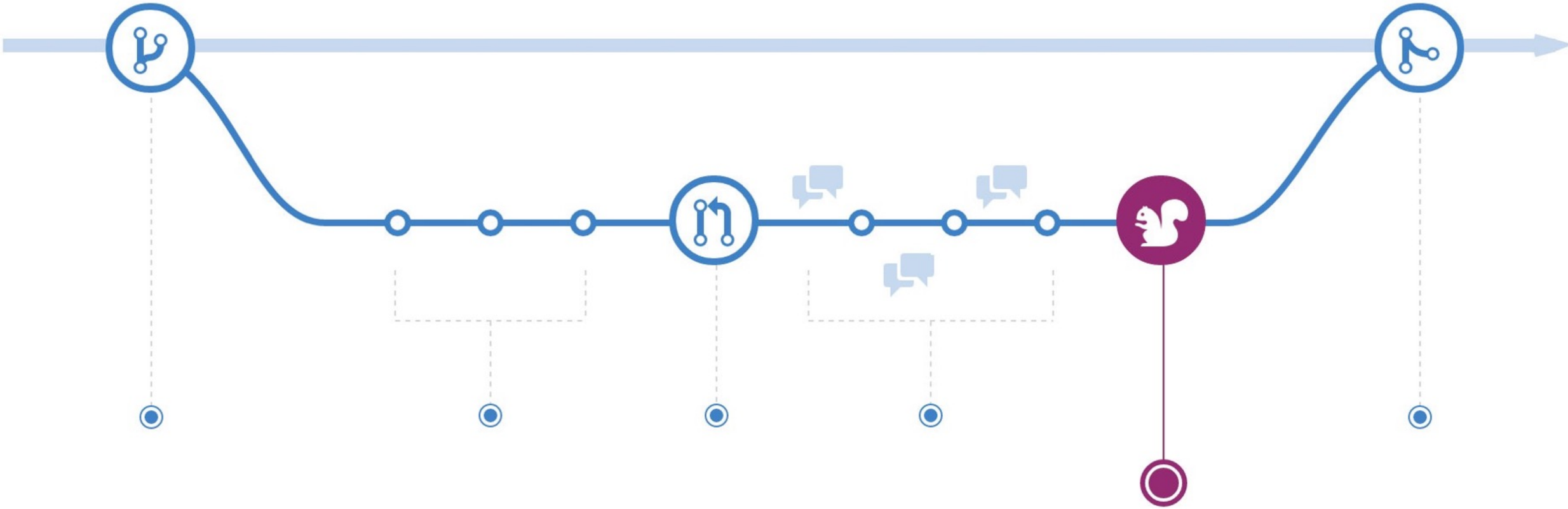
Step 3 Open a Pull Request

GitHub Flow



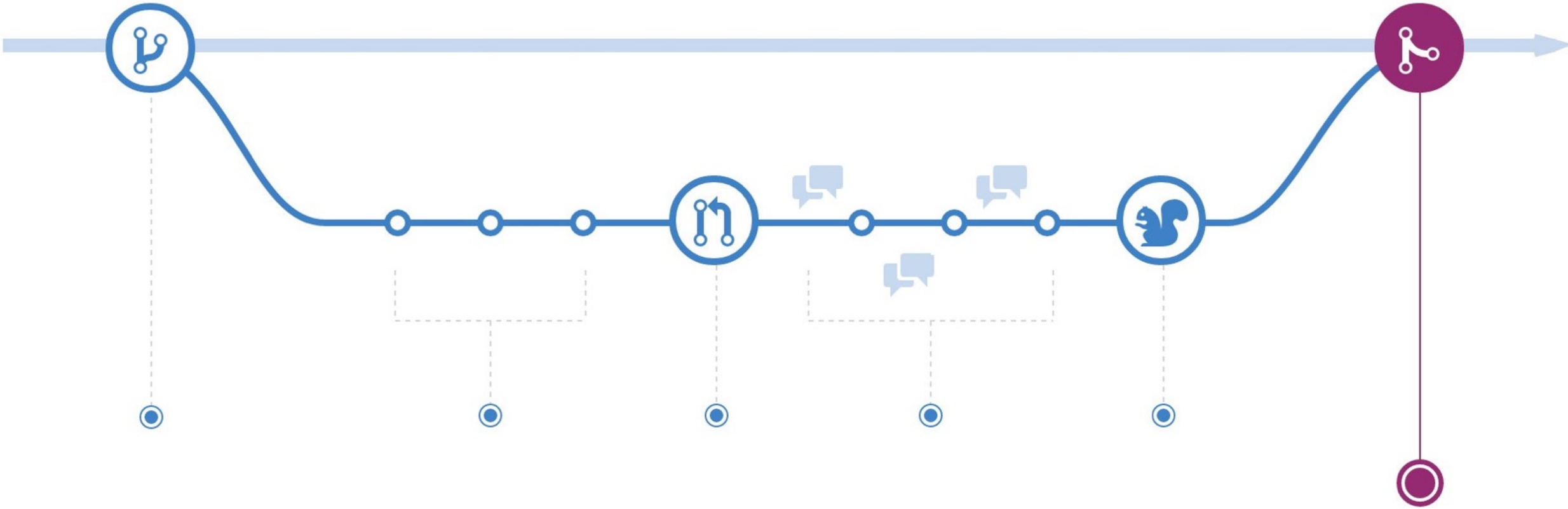
Step 4 Discuss and review your code

GitHub Flow



Step 5 Deploy your branch

GitHub Flow



Step 6 Merge your branch to master

Collaborating with other developers

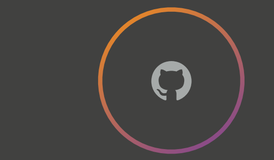
Adding collaborators to your repository

- You can add collaborators to any repository that you own
- Collaborators have push access to your project
- If you want people to collaborate with you but not have push access, you can simply make a public repository, and invite people to fork and open a pull request

Creating an Organization and Teams

- If you have a company, or want to collaborate on several projects with the same group of people, you may want to create an organization
- Teams are a way within an organization to make subsets of people with various areas of responsibility
- Teams allow you to @mention groups of people in Issues and Pull Requests
- Learn more at: https://help.github.com/categories/user-accounts/ and https://github.com/business

Handling merge conflicts



- Occurs when more than once person or commit addresses the same line of code
- A merge conflict means Git does not know which version is the correct version, so you have to manually resolve it
- They can be frustrating, but solvable with the right tools
  - Look for the arrows
  - There are Atom packages available to help resolve them
- You can avoid merge conflicts with process, but they are not completely avoidable
- Lets try an example…

A quick look at Git commands

Basic Git commands

- `git init` • create a new local repository
- `git clone [path]` • copy a remote repository to your local machine
- `git add` • adds a file
- `git commit -a` • commit all changed files
- `git push` • pushes your local changes to the remote repository
- `git pull` • pulls changes down from remote to your local copy
- `git status` • shows the status of your local repository
- `git checkout [branchname]` • checkout an existing branch
- `git checkout -b [branchname]` • creates a new branch and makes it the current branch

Some advanced Git commands

- `git stash` • stores your changes away from your branch so you can commit and push a subset, or reset your branch, etc
- `git stash pop` • re-introduces your stored changes
- `git bisect` • allows you to find where your code broke by splitting it in half
- `git revert` • allows you to undo a specific commit
- `git cherry-pick` • allows you to select commits from a branch that may have been discarded and remake the commits
- `git rebase` • allows you to replay your commits over the existing commits
- `git merge --squash` • allows you to squash all commits into one for benefit of creating a clean history
- Learn more at: http://git-scm.com & StackOverflow

Q & A time

Thanks for coming!

Hit me up @jglovier on Twitter with questions anytime.