Git workflows and when you might encounter them

Childhood Cancer Data Lab x Alex's Lemonade Stand

Objectives

- Describe different approaches to git workflows we use at the Data Lab
- Provide context for why you might pick one approach over another
- Review concrete examples (i.e., public Data Lab repositories)

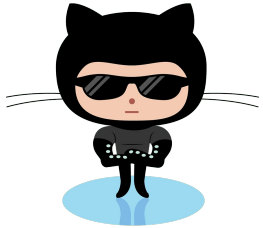
We're not going to get into every kind of workflow, of which there are many, or every bit of terminology.

Background

Basic principles that transcend an individual project

- Having remote backup of a code base that's consistently updated is a good thing.
- There should be some agreed upon (and documented) set of standards for what's allowed into whatever branch we're treating as the main or default branch.
- All code benefits from another set of 👀 (for a number of reasons we'll get into later in the workshop…).
- Usually whatever we're contributing to is a living document, scientific project or product where iterative development can be advantageous.

A bit of technical table-setting

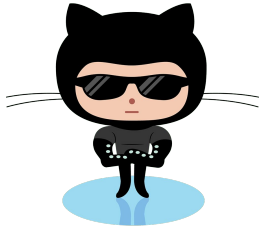


GitHub repositories have **different permission levels** (directly quoting these docs):

- ***Read:*** *Recommended for non-code contributors who want to view or discuss your project*
- ***Triage:*** *Recommended for contributors who need to proactively manage issues, discussions, and pull requests without write access*
- ***Write:*** *Recommended for contributors who actively push to your project*
- ***Maintain:*** *Recommended for project managers who need to manage the repository without access to sensitive or destructive actions*
- ***Admin:*** *Recommended for people who need full access to the project, including sensitive and destructive actions like managing security or deleting a repository*

Image from: https://octodex.github.com/bouncer/

A bit of technical table-setting



- **GitHub organization base permissions.** Every GitHub organization has the concept of a base role with some base level of permissions. This impacts the git workflows you can use. If the organization you belong to sets the base permissions to write, you'll have write access to every repository by default and be able to make branches in your organization's repositories.
- **When forks are necessary.** If you have read-only access to a repository, e.g., because it's someone else's public project, that's generally when you are going to use a forks of a repository.

Image from: https://octodex.github.com/bouncer/

Project archetypes

Project archetype: **Analysis Project**

What is it? A repository where we're performing research or writing material that may or may not be publicly available, but we don't expect people other than our coworkers to "consume" it

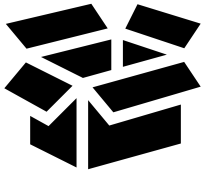
Needs:

- Analyses are the correct to the best of our knowledge
- Code can run without error

Don't need to be overly concerned about releases except when it's preprint, revision, etc., time

Don't want people to go a long time without getting feedback

Project archetype: **Packages and Workflows**



What is it? A repository under consistent development that we expect other people to use regularly, like an R package or a Nextflow workflow

Same **needs** as analysis projects, plus an ability to develop new functionality without releasing something “half-baked” for general usage

Still **don’t want** people to go a long time without getting feedback

Project archetype: **Documentation**

What is it? A repository that exclusively contains user-facing documentation about a product with a code base that's managed separately

Want to be able develop documentation roughly at the same time that features get implemented, i.e., because they are fresh in our minds

Need to wait until features are released to publish documentation about them

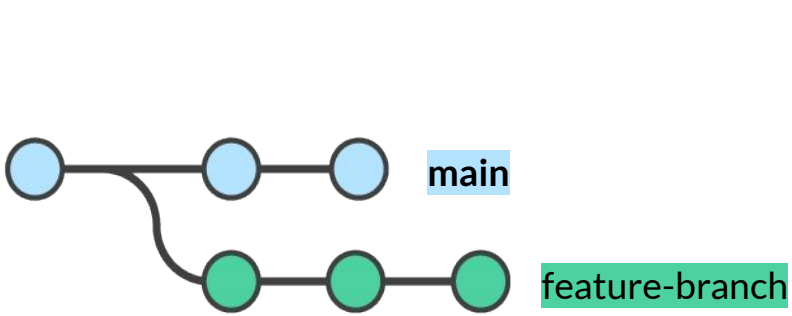
Again, **don't want** people to go a long time without feedback!

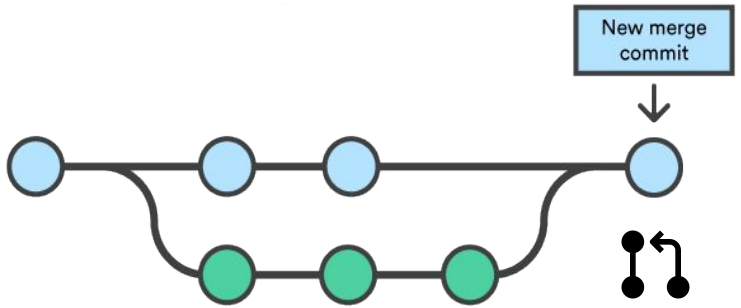
Git workflows

Feature branch workflow

Overview: Start with the `main` branch which is the “official branch” and develop on a named branch every time you want to add something new.

File a pull request to merge when it’s ready to be added to the official branch.

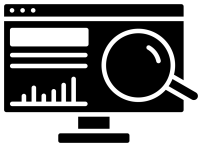




Images from https://www.atlassian.com/git/tutorials/using-branches

Feature branch workflow

We’ve found this to be most appropriate for



Analysis Projects

Where it’s most important to keep an up-to-date, official record of the project with code and documentation reviewed for correctness, etc.

Feature branch example: `sc-data-integration`

What is it? A repository that contains analyses that support some of the decisions we make around cell typing and/or integration underlying our Single-cell Pediatric Cancer Atlas Portal. This code isn't used in production environments; it's more for us to keep track of and surface our work to others.

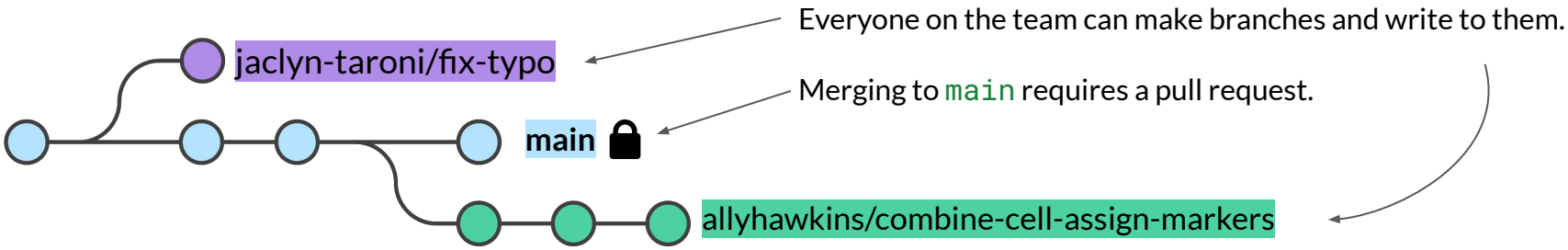


Image adapted from https://www.atlassian.com/git/tutorials/using-branches

Development and main workflow

Overview: Any new work starts in named branches off of a `development` branch. When that work is finished, file a pull request to `development`. The `main` branch is the official or publicly facing release (whatever that means for your project).

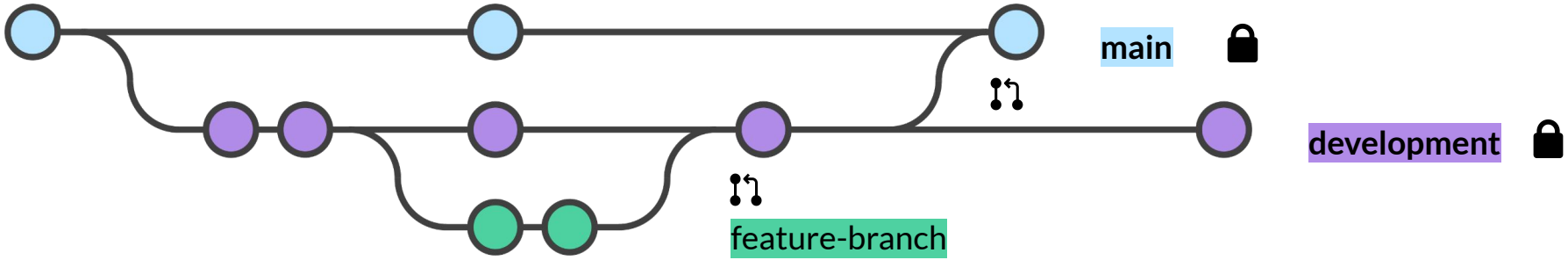
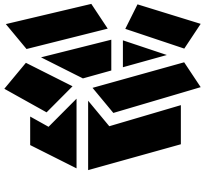


Image adapted from https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Development and main workflow

We've found this to be most appropriate for



Packages and Workflows

Documentation

Where you can continue to add to `development` internally and merge `development` into `main` when it's ready for other people

Development and main example: `scpca-nf`

What is it? A repository that contains the Nextflow pipelines that we use to process data for the ScPCA Portal.

- `main` is the default branch that contains the current release version.
- `development` contains new updates ready in advance of release.

When changes in `development` merit a new release, a pull request is filed to merge `development` into `main`. We tag a release on `main`.

Development and main example: `refinebio-docs`

What is it? A repository that contains user-facing documentation for our product refine.bio.

- `development` is the default branch that serves as the “working copy” of the docs.
- `main` contains the current user-facing version of the docs (this is what Read The Docs uses).

When documentation in `development` are relevant to users, a pull request is filed from `development` to `main`.

Forking workflow

Overview: Contributors work on their own fork, or copy of, the official project repository. Typically, contributors don't have write access to the official project repo. This can often be used in conjunction with any other workflow; it just depends on what branch the pull requests target.

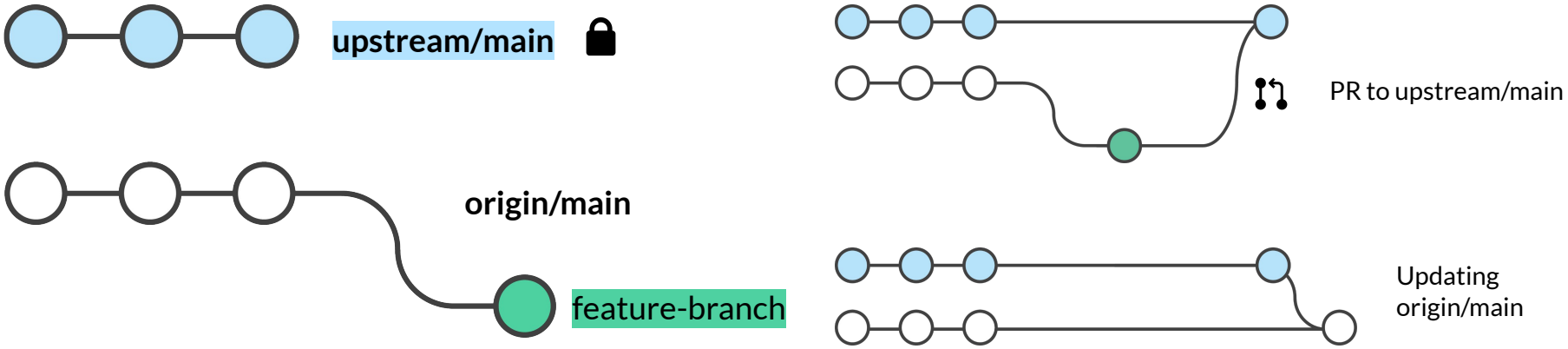
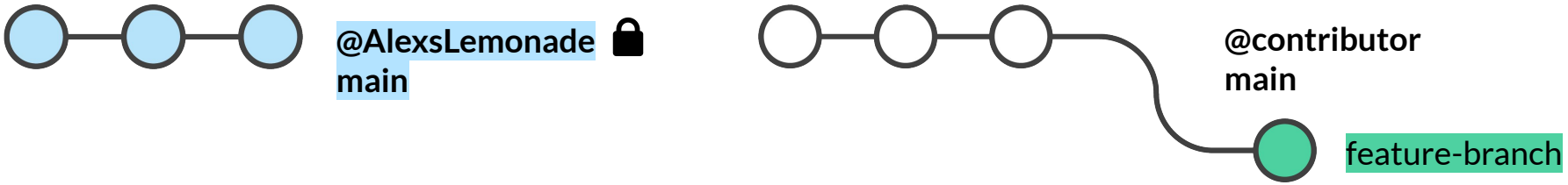


Image adapted from https://www.atlassian.com/git/tutorials/using-branches

Forking example: `OpenPBTA-analysis`

What is it? An analysis project where not all contributors have write access to the `AlexsLemonade` project repository.



A note on pull requests when **forking**

Allowing edits from maintainers lets people with push access to the upstream repository commit to the branch in your personal fork

https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/working-with-forks/allowing-changes-to-a-pull-request-branch-created-from-a-fork

Git workflows summary

- The Git workflow strategy for a given repository should be dictated by the permissions levels for contributors and the specific needs of the project.
- A **feature branch** workflow is useful when your principal aim is to keep an official record of an ongoing project.
- A **development and main** workflow can be used when you need to distinguish what is actively being worked on from what should be publicly facing.
- A **forking** workflow is typically used when contributors don't have write access to the official repository but can be used in conjunction with other models.