COSC 499: Capstone Software Engineering Project

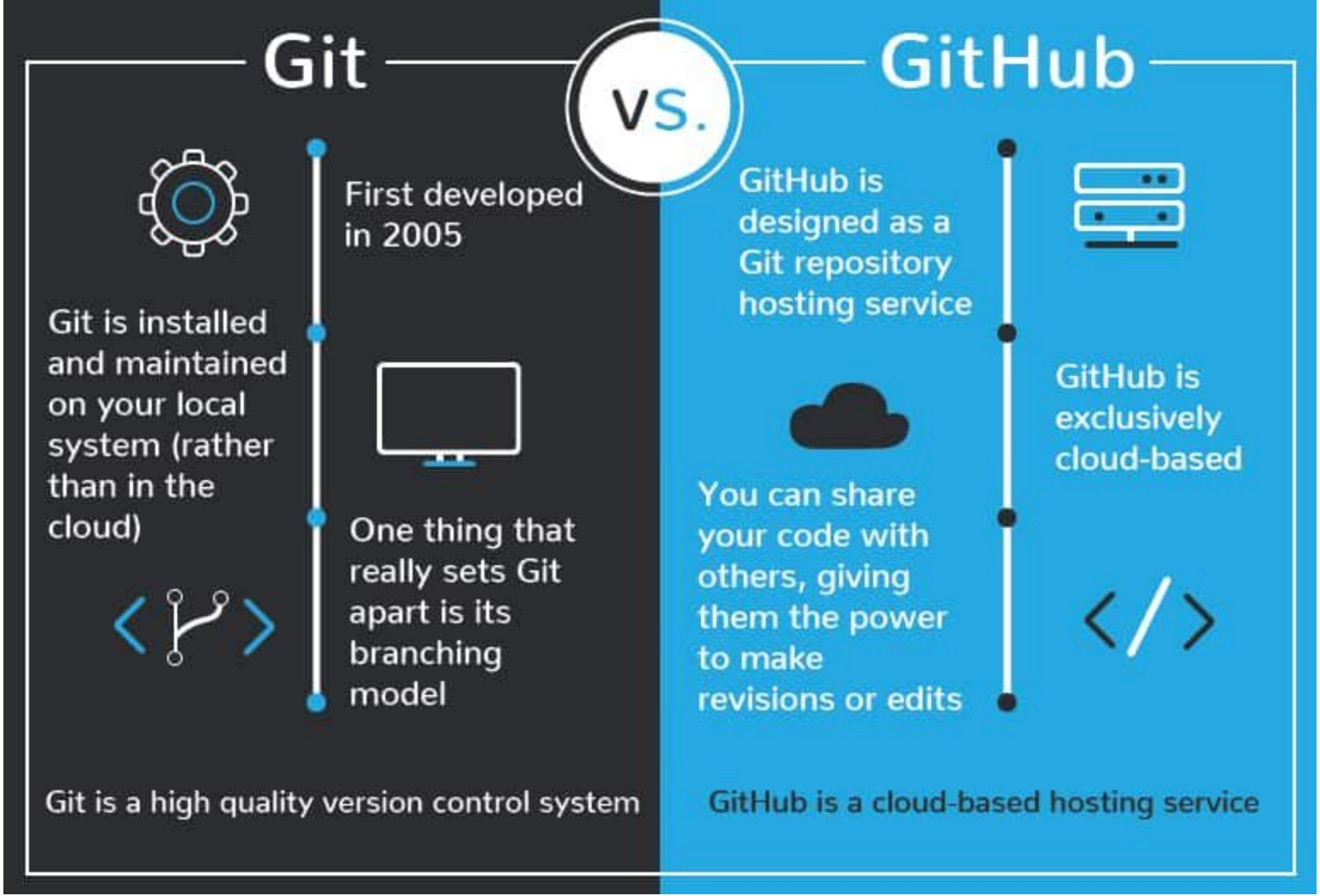


Image taken from https://devmountain.com/

Git and GitHub

- Git is a modern version control system for software developers
- GitHub is a public platform that supports Git in the cloud
  - Do not push files with secrets (use `.gitignore`)
- Using GitHub
  - GitHub CLI https://cli.github.com/
  - GitHub Desktop https://docs.github.com/en/desktop
    some IDEs also have Git integration
  - This course: GitHub classroom

Examples?

Development Process in Git Commands

- Common steps:

```
git clone
git pull origin master
git checkout -b new_branch_name
```

[create files, write tests, write code]

repeat

```
git add file_name
git status
git commit -m "string message describing change"
git push new_branch_name
```

[create pull request, wait for review, make fixes as needed, merge to master]

Development Process in Git Commands

- Common steps:

```
git clone
git pull origin master
git checkout -b new_branch_name
[create files, write tests, write code]
git add file_name
git status
git commit -m "string message describing change"
git push new_branch_name
[create pull request, wait for review, make fixes as needed, merge to master]
```

repeat

So important!

Why?

Basic Development Process: Alone

-

- Write new code
- Attempt to synch new code
- Merges branch to master

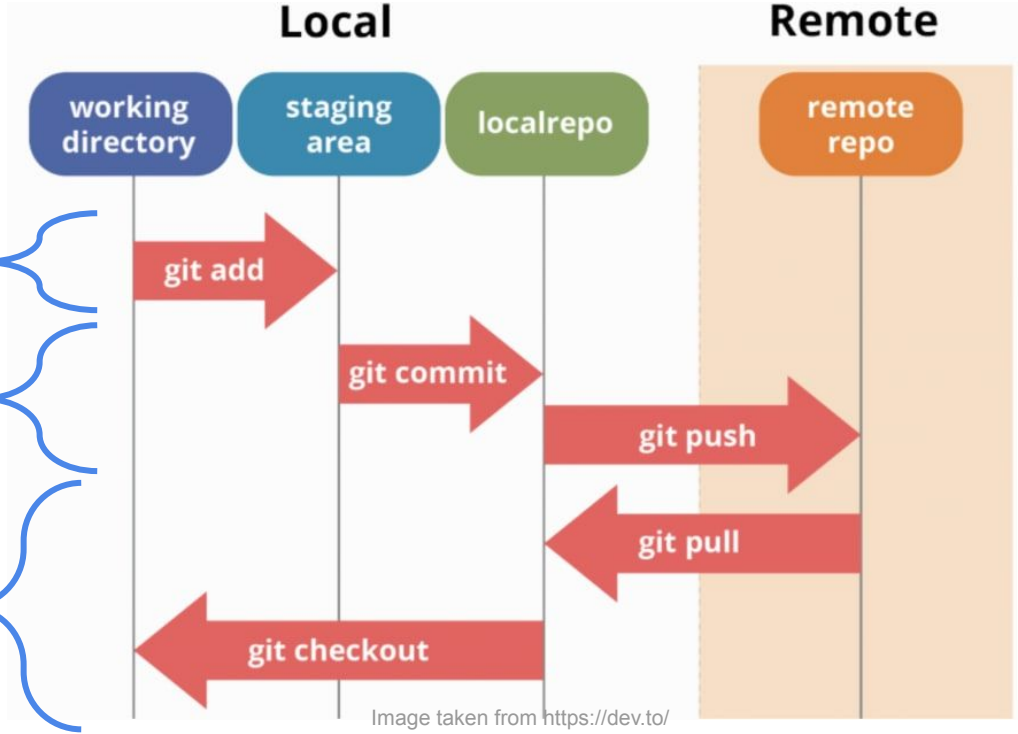


Image taken from https://dev.to/

Basic Development Process: In a Team

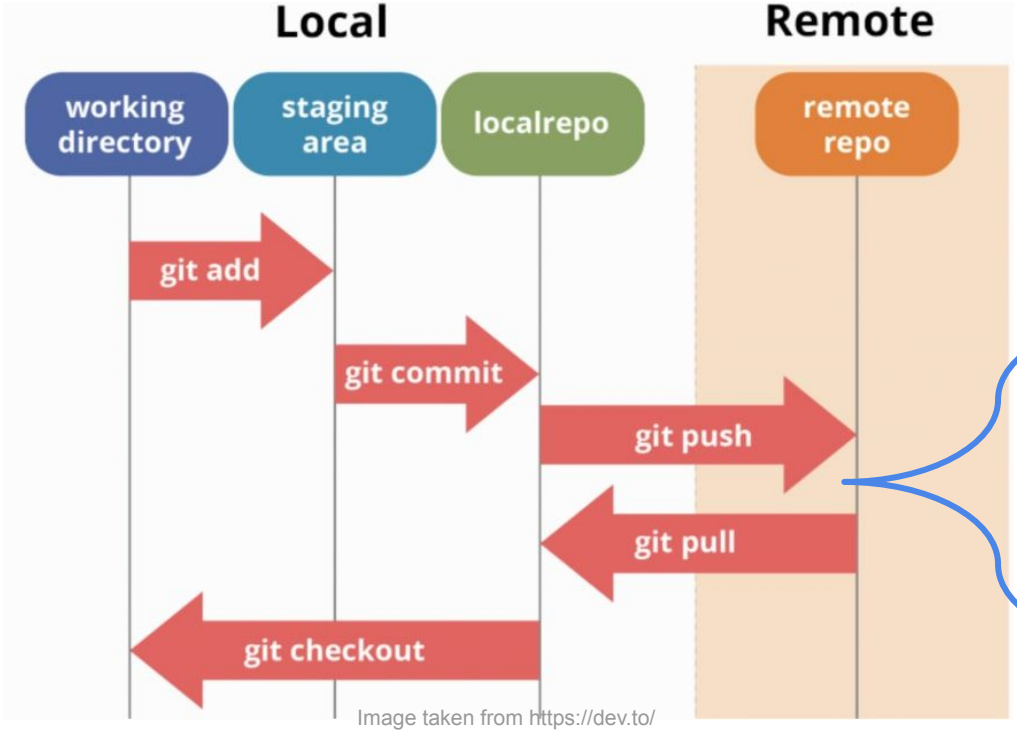


Image taken from https://dev.to/

- Create a pull request (PR)
- Assign reviewers (usually 2)
- Fix changes as needed
- Repo policy can auto merge if both reviewers approve

Branching and Merging

- Everyone should follow the same branching strategy
  - Helps teams move faster
  - Teams can work on parallel builds

Branching and Merging

- Everyone should follow the same branching strategy
  - Helps teams move faster
  - Teams can work on parallel builds
- Why use branches?
  - Your edits don't immediately effect the original source code
- Branch when you have a new feature

Branching and Merging

- Everyone should follow the same branching strategy
  - Helps teams move faster
  - Teams can work on parallel builds
- Why use branches?
  - Your edits don't immediately effect the original source code
- Branch when you have a new feature
- Naming convention
  - Use a short representative description of the feature
  - Do NOT name it after yourself

Branching and Merging

- Everyone should follow the same branching strategy
  - Helps teams move faster
  - Teams can work on parallel builds
- Why use branches?
  - Your edits don't immediately effect the original source code
- Branch when you have a new feature
- Naming convention
  - Use a short representative description of the feature
  - Do NOT name it after yourself
- Merging your branch
  - Process of joining your branch with original source code after fully testing

Visualizing Git Flow

- Master branch - stable code, ready to stage/deploy
- Develop branch - where features merge to

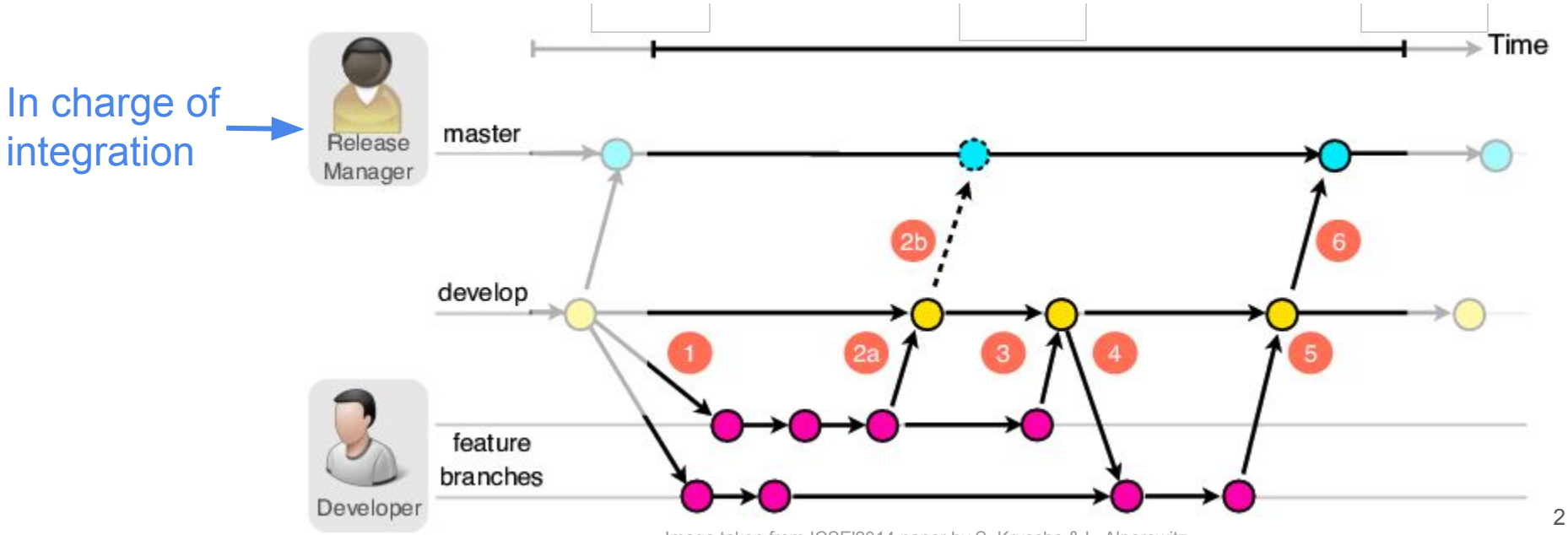
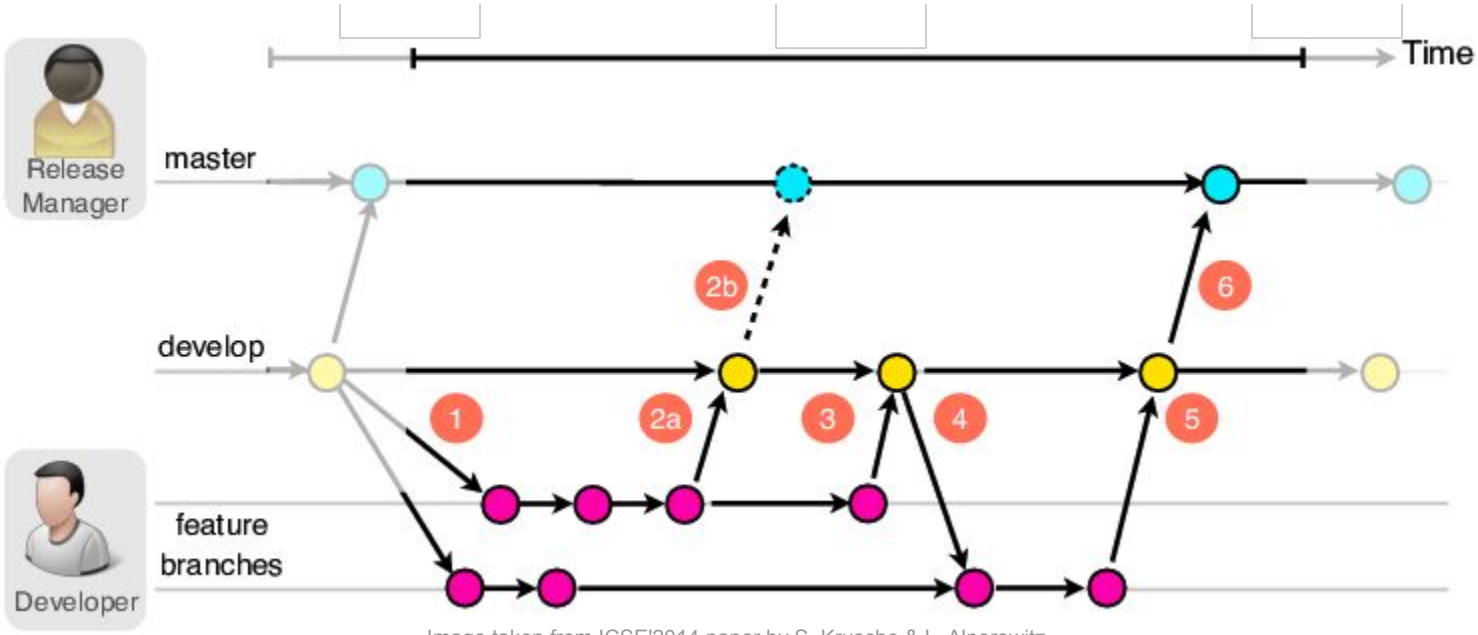


Image taken from ICSE'2014 paper by S. Krusche & L. Alperowitz

Visualizing Git Flow

- Master branch - stable code, ready to stage/deploy
- Develop branch - where features merge to



Features branch off of and merge into develop

Image taken from ICSE'2014 paper by S. Krusche & L. Alperowitz

Visualizing Git Flow

- Master branch - stable code, ready to stage/deploy
- Develop branch - where features merge to

How many features were merged to develop?

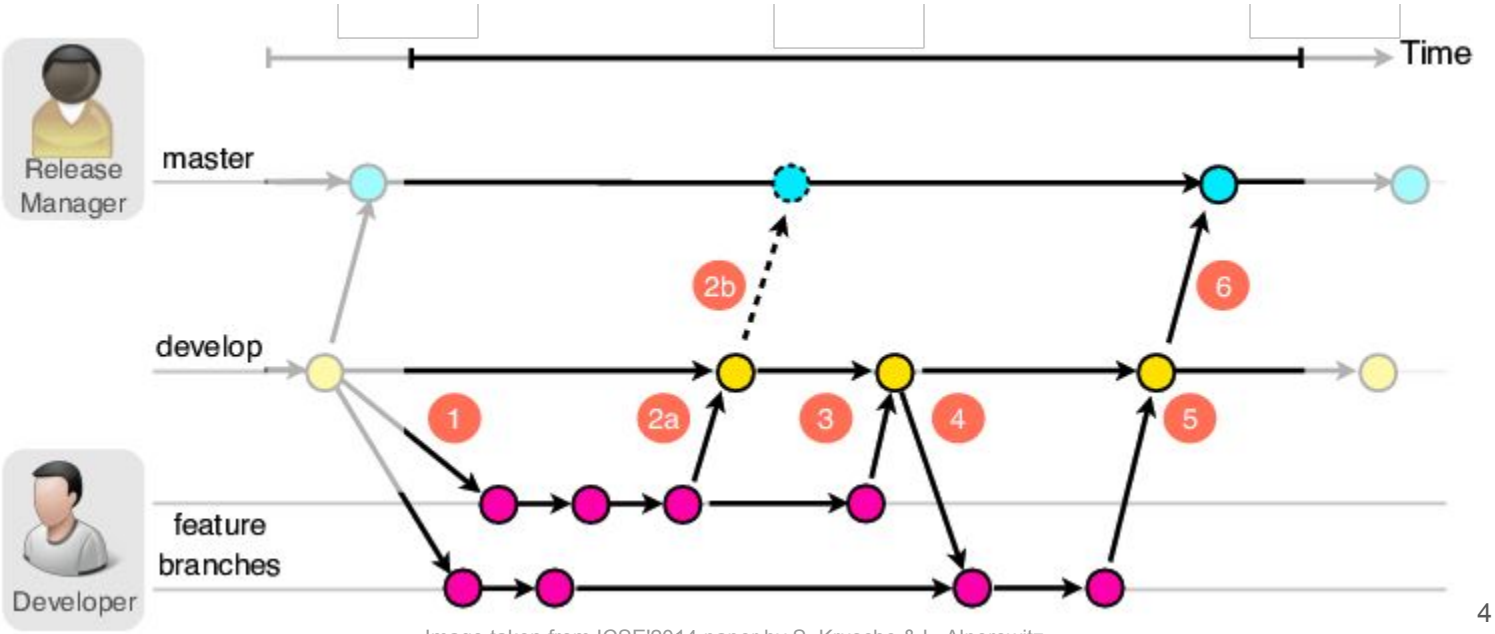


Image taken from ICSE'2014 paper by S. Krusche & L. Alperowitz

Visualizing Git Flow

- Master branch - stable code, ready to stage/deploy
- Develop branch - where features merge to

After the merge in 2a, why was there continued development that led to 3?

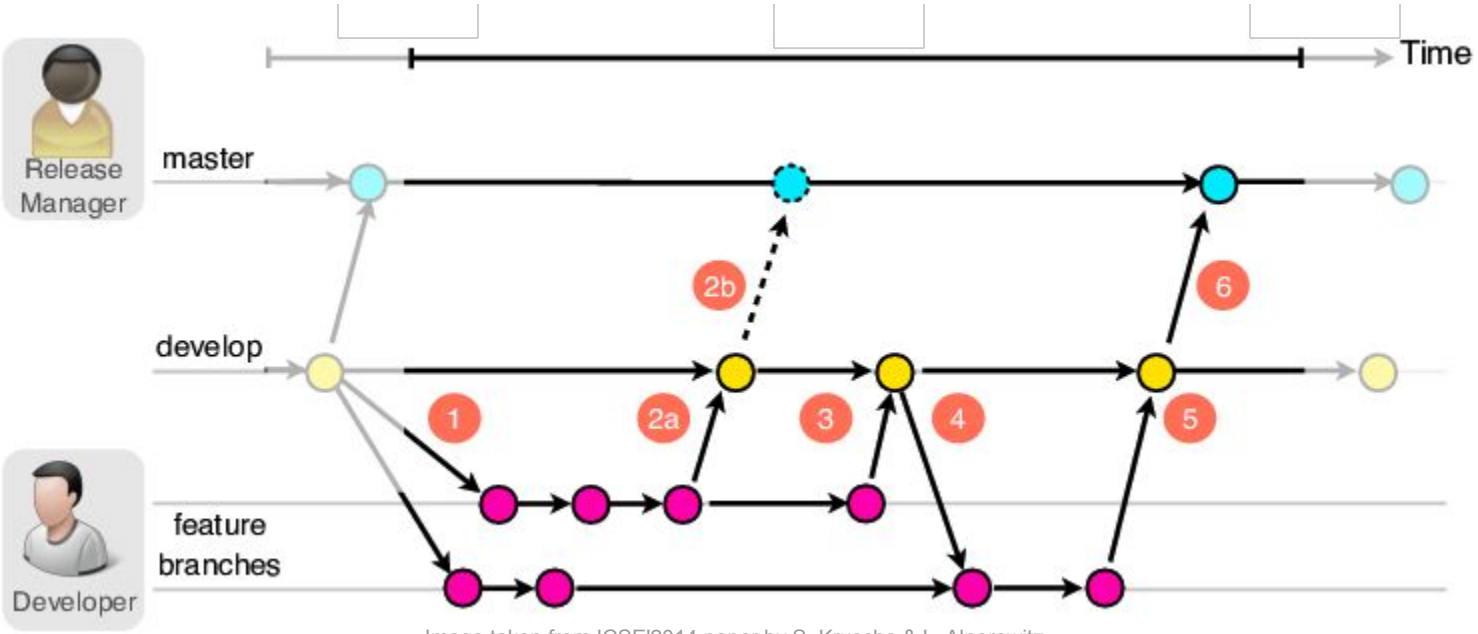


Image taken from ICSE'2014 paper by S. Krusche & L. Alperowitz

Visualizing Git Flow

- Master branch - stable code, ready to stage/deploy
- Develop branch - where features merge to

Why is there another pull from develop at 4?

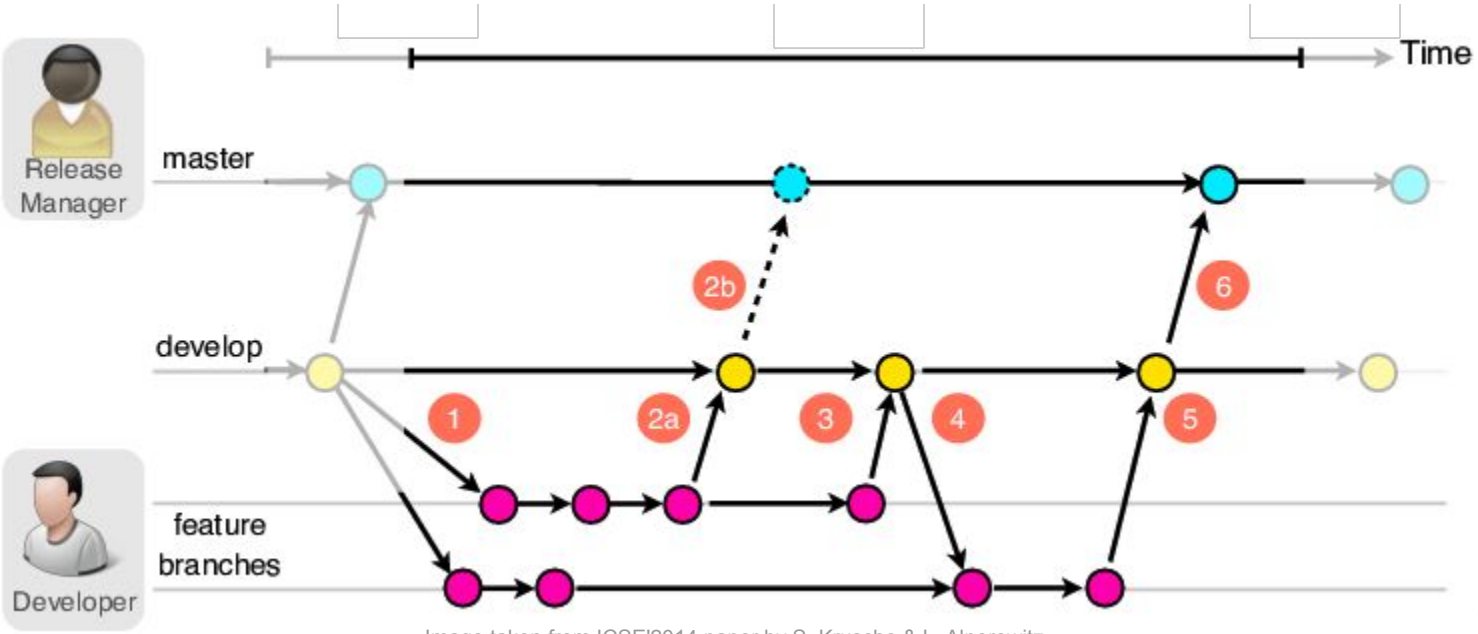


Image taken from ICSE'2014 paper by S. Krusche & L. Alperowitz

Branching Conventions for this Course

- Branches for your repo
  - master — for deployment / deployment-ready
  - develop — for active development
  - doc — for repo documentation only
  - log — for class logs only; irrelevant to client
  - one per feature
- In log branch:
  - student A indiv logs
  - student B indiv logs
  - …
  - team logs

Branching Conventions for this Course

- Do NOT delete your feature branches after they have been merged
  - We want to see commit patterns in GitHub insights
- The integration lead is the only one who can merge develop to master
  - You may want 1-2 people assigned to integration
- Required reviewers:
  - Features and documentation require 2 reviewers
  - Logs require 1 reviewer
  - After reviewers approve a PR, the repo can auto-merge the feature to develop or the last reviewer can do it manually

Another Example

- More complex branching strategy

Organization dependent

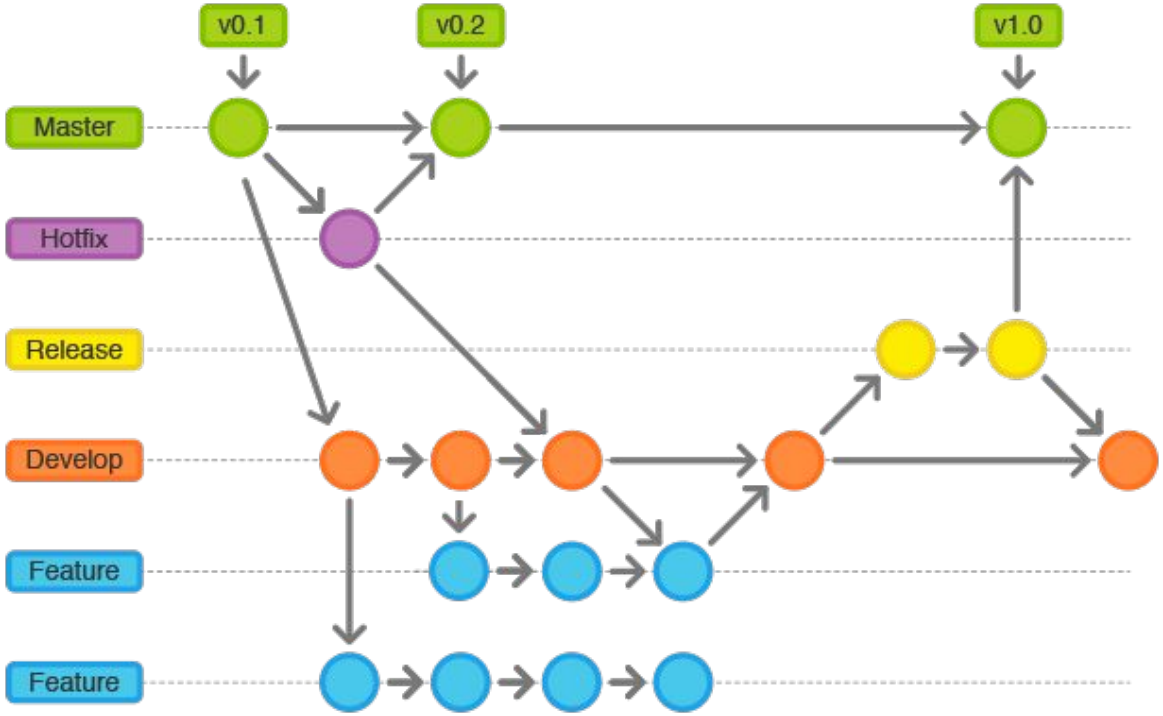


Image taken from https://lrtitecnologia.wordpress.com/

Another Example

- More complex branching strategy

Hotfix requires a synch with both master and develop

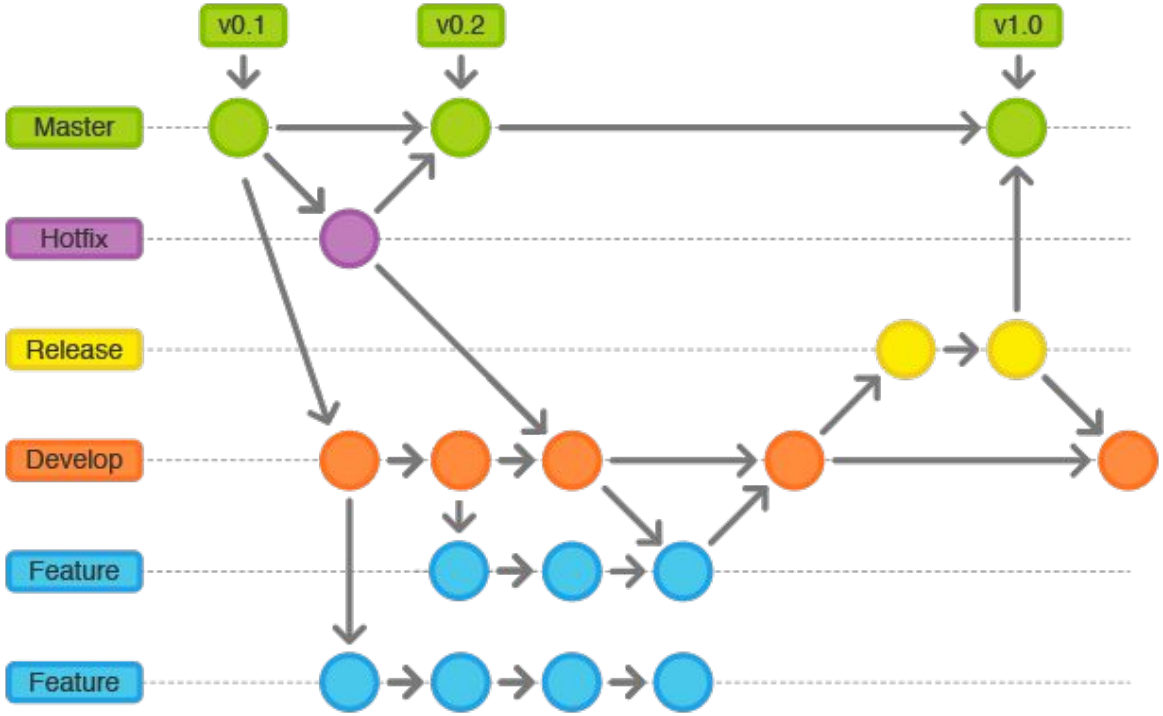
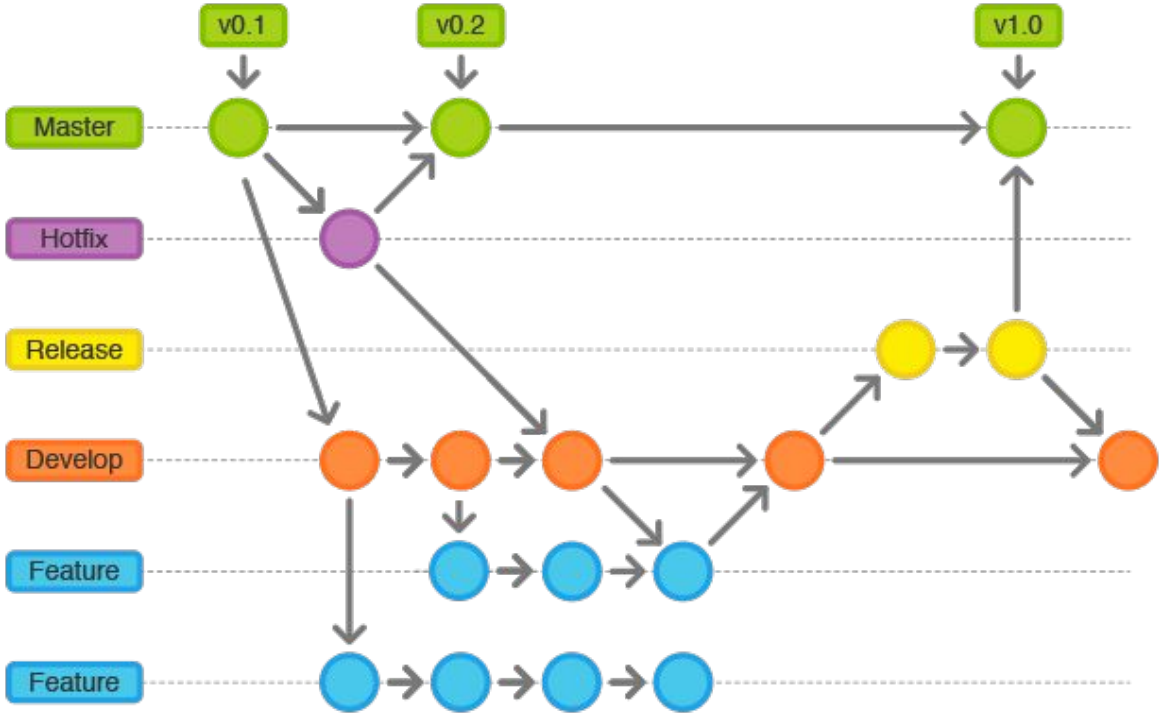


Image taken from https://lrtitecnologia.wordpress.com/

Another Example

- More complex branching strategy



What's happening with the bottom feature?

Image taken from https://lrtitecnologia.wordpress.com/

Developer Maxim: Merge Early and Often

- Merging is analogous to a "synch" action
- What happens if you develop your feature for a long time while everyone else is developing and merging their work?

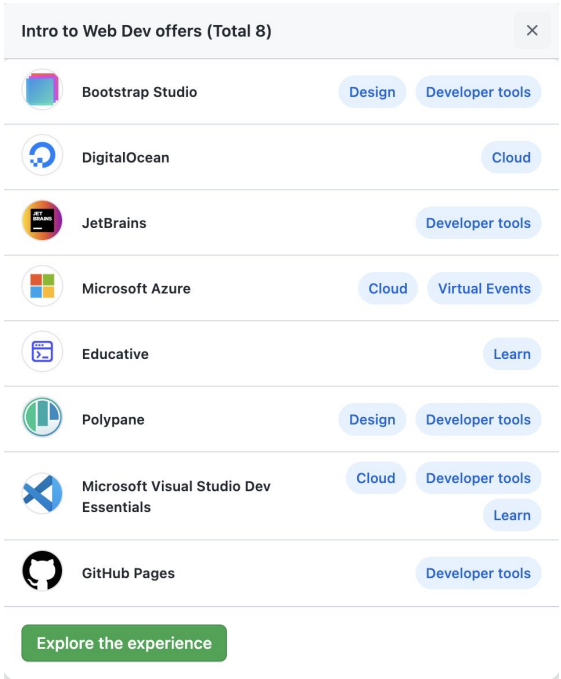
Developer Maxim: Merge Early and Often

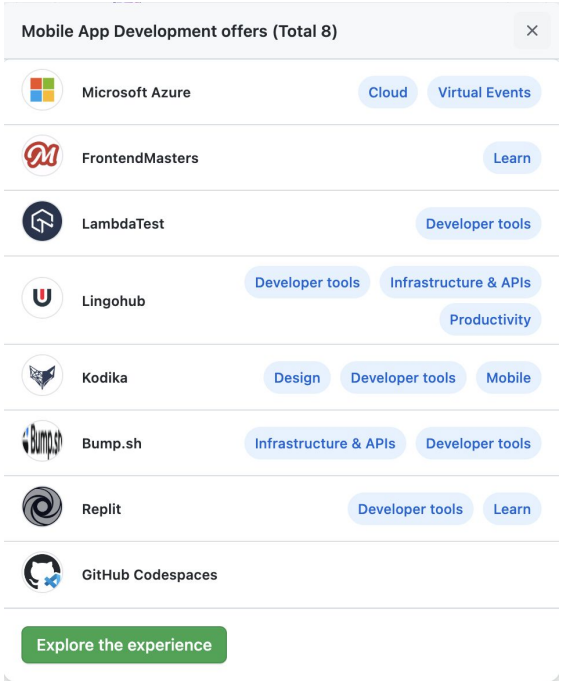
- Merging is analogous to a "synch" action
- What happens if you develop your feature for a long time while everyone else is developing and merging their work?
  - You get stuck handling all the merge conflicts

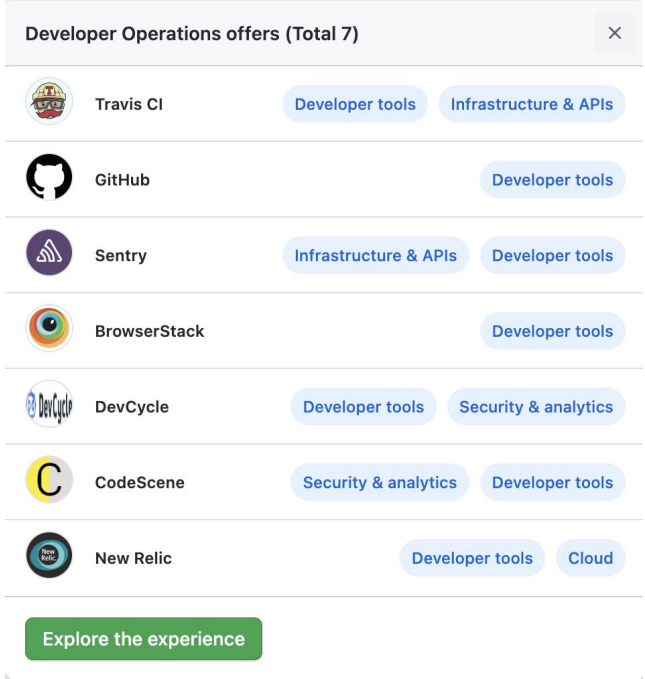
- Therefore: Pull changes and integrate frequently
  - Keep your PRs small
  - Avoid massive merge conflicts

Other Goodies: GitHub Student Developer Pack

- Available to students and offers a range of free services
  https://education.github.com/pack







Next Steps

- Watch the IP guest lecture
- Review sample IP agreement
  - Client's ownership over project
  - Students and instructor maintain educational rights
- Review project plan template

- Next week:
  - Finalize project option
  - Draft out project requirements, print 6 copies for class
  - Meet with teams for each project option ~20 min
  - Submit client questions
  - Complete Team exercise
  - Submit logs at the end of Week 3 (your first "dry-run")