# GitLab Process

Demo videos linked on course website.

## Topics

1) Some notes on Git
    a) .gitignore,
    b) Commit Messages
    c) Reverting
2) How can we organize development?
   Use branches and workflows

## .gitignore

- .gitignore File
  - Lists file types to exclude from Git:..
  - Example:
    Exclude .bak, build products, some IDE files
- Tag
  - “Tag” the project’s contents at a specific commit
  - Can later check-out that tag to return to the project state at that time
  - Example Uses
    - Track project code going into a release: “V1.51”

## Commit Messages

- A good commit message is required!
  - Line 1: .. (<50 characters)
    Capitalize your statement
    Use imperative: "Fix bug..." vs "fixed" or "fixes"
  - Line 2: ..
  - Line 3+: .. ; wrap your text ~70 characters

Example:

```
Correct text alignment on UI

Begin using form layout to ensure UI elements
line up correctly. Removed use of badtext.lib.
```

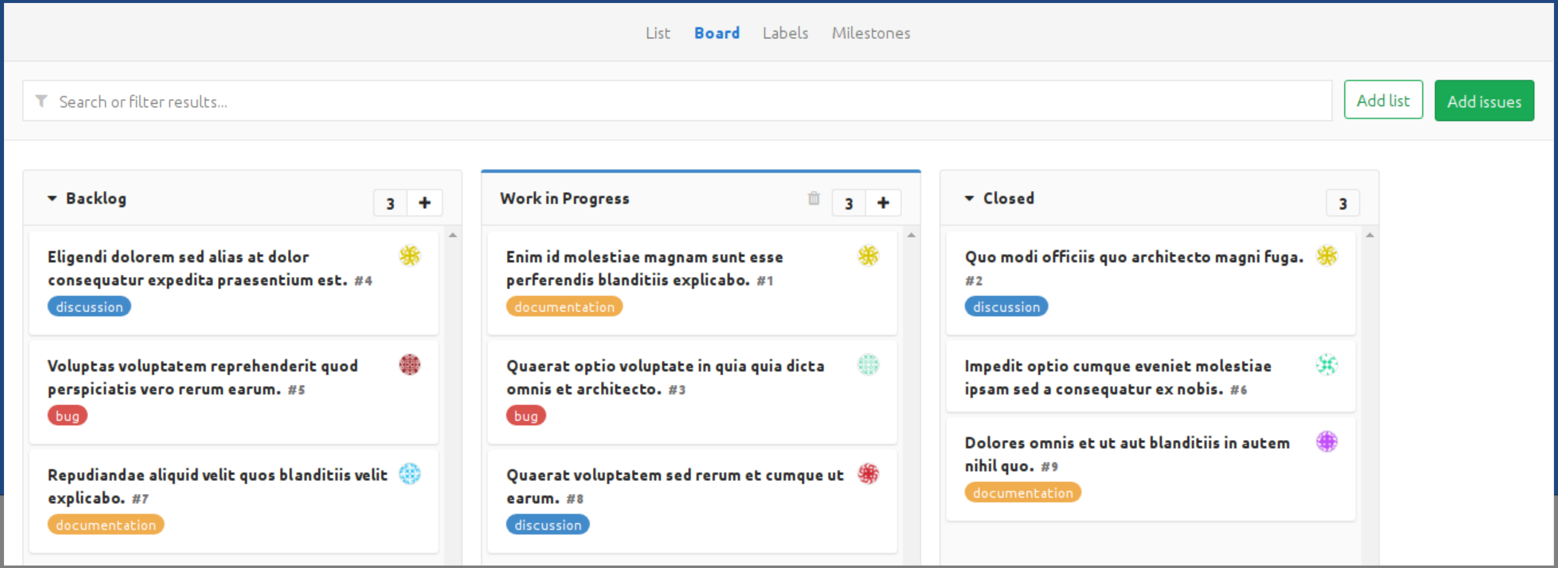
## Reverting Changes

- Git checkout = revert
  - ..
  - Overwrite file in working directory with one from local repo.
- Revert with Caution
  - Will lose all uncommitted changes in the file.
  - Normally Git does not let you lose changes.
  - If in doubt, grab a backup copy (ZIP your folder) then revert.
    - Just make sure you don’t commit the backup!

## Branches and Workflow

# Issues in GitLab

- GitLab tracks Issues:
  ..
- Value of Issues
  - Use as product’s backlog (GitLab’s “boards”)
  - Assign issue to a dev to show who's working on it
  - Update issue with extra info as needed



Backlog 3

Eligendi dolorem sed alias at dolor consequatur expedita praesentium est. #4
discussion

Voluptas voluptatem reprehenderit quod perspiciatis vero rerum earum. #5
bug

Repudiandae aliquid velit quos blanditiis velit explicabo. #7
documentation

Work in Progress 3

Enim id molestiae magnam sunt esse perferendis blanditiis explicabo. #1
documentation

Quaerat optio voluptate in quia quia dicta omnis et architecto. #3
bug

Quaerat voluptatem sed rerum et cumque ut earum. #8
discussion

Closed 3

Quo modi officiis quo architecto magni fuga. #2
discussion

Impedit optio cumque eveniet molestiae ipsam sed a consequatur ex nobis. #6

Dolores omnis et ut aut blanditiis in autem nihil quo. #9
documentation

## Branches

- .. Main source code *branch* in a Git repo.
- .. Latest code on master.
- Chaotic Commits
  - Too chaotic to have many teammates constantly committing code to master.
  - Solution:..
- Branch (Feature Branch)
  - Do work on separate track (the branch) from Master
  - Commit changes to your branch
  - When feature is ready,
    ..

# Branching and Merge Request (Overview)

- Process Overview
  GL = done in GitLab
  IJ = done in IntelliJ
  - GL: Pick an *issue* to implement & create feature branch.
  - IJ: Commit/push changes to the branch.

    When feature is ready:
  - IJ: Merge Master to Feature branch
    (resolving conflicts)
  - GL: Create merge request to merge branch to Master.
  - GL: Branch is deleted when merge request accepted.
    (manually remove merged local branch)

## Issues and Branching

1. Create issue for bug/feature
   - Implementing a feature or fixing a bug should start with a GitLab issue.
   - Ex: Issue 14: "Add help button to game activity"
2. Assign issue to yourself
3. Create feature branch in GitLab
   - It names the branch..
     - Ex: 14-game-help-button
   - In IntelliJ, checkout the branch:
     ..

## Issues and Branching (cont.)

4. Work on your branch
   - Do your work changing files
   - Check-in your changes via Git:
     - add: tell Git to commit changes in this file
     - commit: put changes into local repo on branch
     - push: push commit to remote repo on branch
5. Merge Master to Feature Branch
   - Get the latest code from master’s HEAD:
     - In IntelliJ: VCS --> Git --> Merge Changes
   - Resolve any merge conflicts
   - Test

## Issues and Branching (cont.)

6. Submit a.. via GitLab
   - Create request to merge your branch back to master
   - Since you already merged Master to Feature Branch, there *should* be no conflicts.
   - If branch name starts with a number, GitLab will pull info from the issue.
     Otherwise, have message include "Fix #14"

# Managing Merge Request

- Team members see merge requests and:
  - Code review:
    Comment on problems they see in the code
    (possibly leading to new commits to fix)
  - Thumbs-up/down for voting
- Repo Manager accepts merge request
  - Accepting merge requests will:
    - merge code to master (should be no conflicts)
    - ..
    - delete the source branch
      [optional; good practice to clean up]

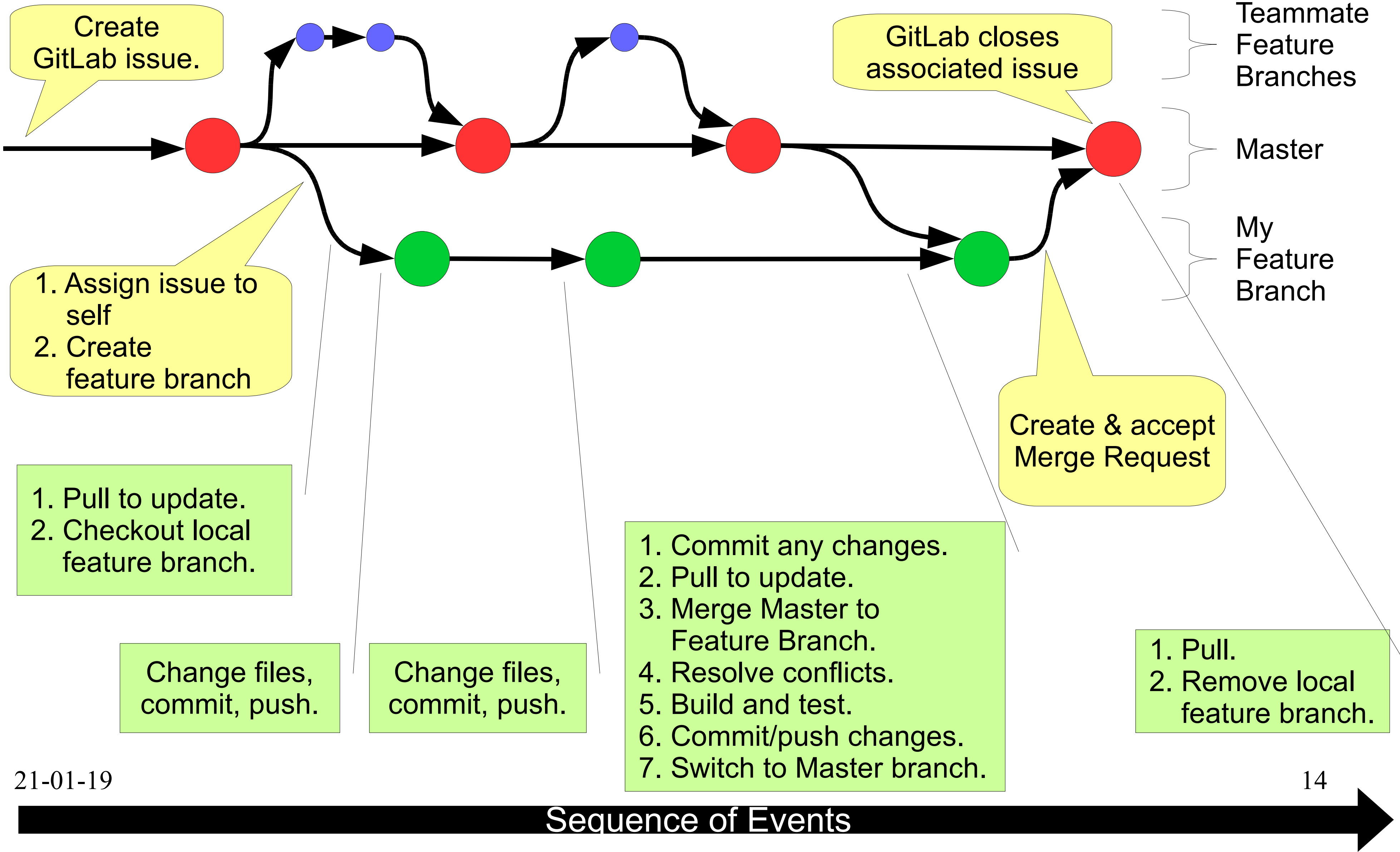
# GitLab Workflow

## Feature Branch, Merging Changes, Merge Request

**Legend**

🟨 In GitLab

🟩 In IntelliJ



Teammate Feature Branches
Master
My Feature Branch
Create GitLab issue.
GitLab closes associated issue
1. Assign issue to self
2. Create feature branch
Create & accept Merge Request
1. Pull to update.
2. Checkout local feature branch.
Change files, commit, push.
Change files, commit, push.
1. Commit any changes.
2. Pull to update.
3. Merge Master to Feature Branch.
4. Resolve conflicts.
5. Build and test.
6. Commit/push changes.
7. Switch to Master branch.
1. Pull.
2. Remove local feature branch.
Sequence of Events

# Summary

- Git Details
  - Merge conflicting changes as needed.
  - .gitignore: ignore files/folders
  - Descriptive commit messages
  - Revert discards changes (git checkout)
- Branches and Workflow
  - Create GitLab issues.
  - Do work on a feature branch.
  - GitLab merge request to merge branch to master.