UNIVERSITY OF TORONTO MISSISSAUGA

CSC207

GitFlow, Merge Requests, Paired Programming

Learning Objectives for Today

- GitFlow overview
- Paired Programming Overview
- Do your first code review!

Git Flow

Step 1: Clone the Group Repository

All partners will need to clone the repository on their own machines. The example below is on github, but the gitlab process is the same.

```
Ayeshas-MacBook-Pro:GitExample macbookpro15$ git clone https://github.com/Daniel-Laufer/csc207-git-in-teams-example.git
Cloning into 'csc207-git-in-teams-example'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
```

Git Flow

Step 2: Make branches

Make branches to isolate the changes each team member makes to the repository. This is how partners Dan and Ayesha make their own branches. You will make a develop branch, as well as feature branches. Each teammate will work on a feature branch.

Dan's Terminal:

```
➡  csc207-git-in-teams-example $ git branch
* main
➡  csc207-git-in-teams-example $ git checkout -b danBranch
Switched to a new branch 'danBranch'
➡  csc207-git-in-teams-example $ git branch
* danBranch
  main
```

Ayesha's Terminal:

```
[Ayeshas-MacBook-Pro:csc207-git-in-teams-example macbookpro15$ git checkout -b ayeshaBranch  ]
Switched to a new branch 'ayeshaBranch'
[Ayeshas-MacBook-Pro:csc207-git-in-teams-example macbookpro15$ git branch                    ]
* ayeshaBranch
  main
```

Git Flow

Step 2: Make branches

This is what the history of commits on the various branches will look like in git, once created.

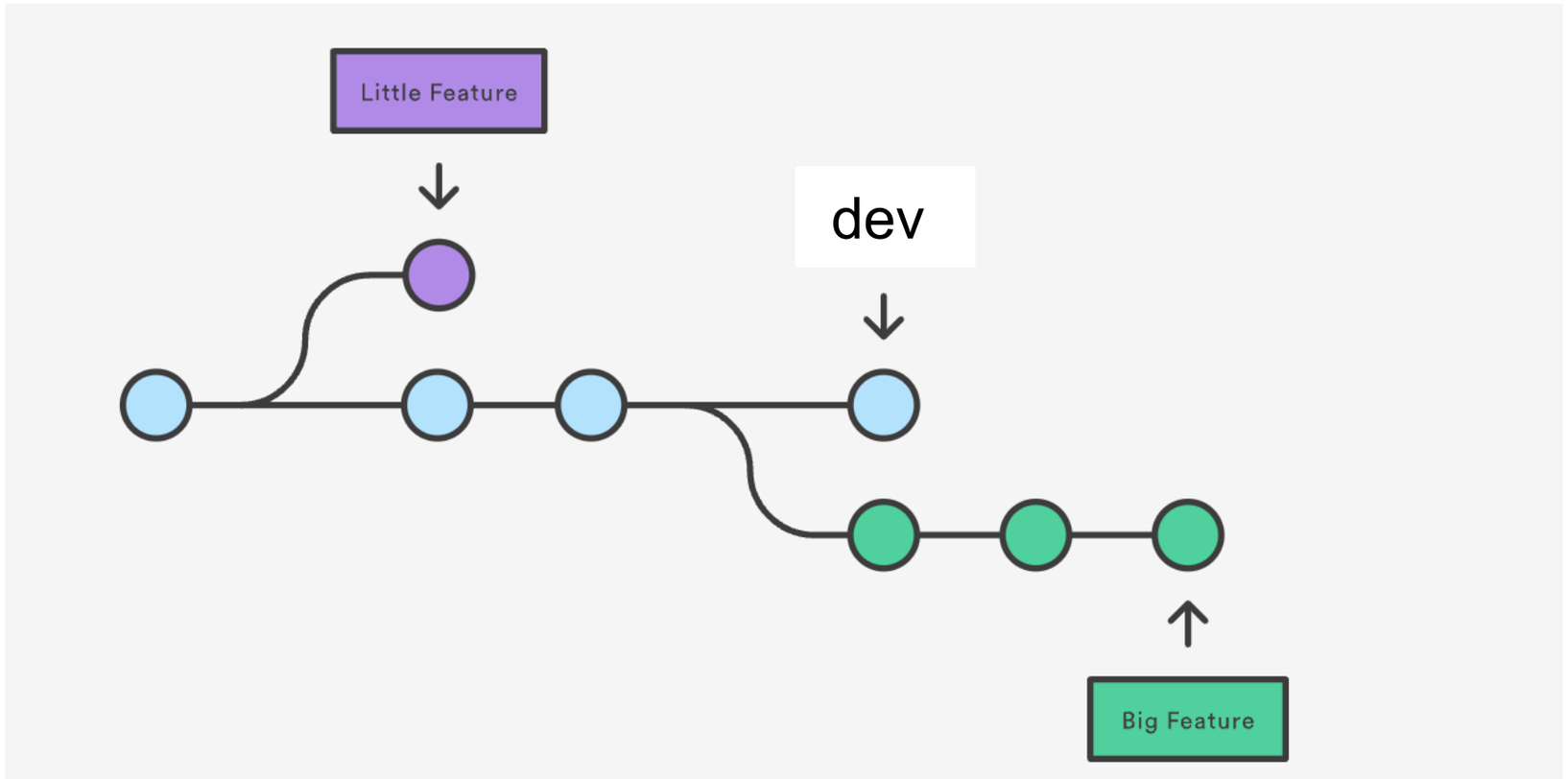


Image from https://www.atlassian.com/git/tutorials/using-branches

Git Flow

Step 3: Add code to branches

Each teammate can track files locally (with git add <filenames>), make changes to the files and then commit changes. These commits will take place **on local branches** and therefore be isolated from other branches.

One teammate can start by writing a test that fails to the existing file of tests. This teammate should not change any other files at this moment.

The other teammate can start by trying to pass a given test in the existing file of tests. This teammate should not change any other files at this moment.

This is kind of like ping-pong paired programming.

Git Flow

Step 3: Add code to branches

Once either teammate has made some changes, these can be committed to the local branch. One teammate can then push the change, and the entire branch, to the remote repository as follows:

```
[Ayeshas-MacBook-Pro:csc207-git-in-teams-example macbookpro15$ git push                                  ]
 fatal: The current branch ayeshaBranch has no upstream branch.
 To push the current branch and set the remote as upstream, use

     git push --set-upstream origin ayeshaBranch
[Ayeshas-MacBook-Pro:csc207-git-in-teams-example macbookpro15$ git push --set-upstream origin ayeshaBranch ]
```

Git Flow

Step 4: Merging at the command line

Teammates can now request merge their changes to the develop branch. To do this, we'll switch to the branch we want to merge onto, as illustrated below. This merge will also need to be communicate to the remote side, as follows:

```
Ayeshas-MacBook-Pro:csc207-git-in-teams-example macbookpro15$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
Ayeshas-MacBook-Pro:csc207-git-in-teams-example macbookpro15$ git branch
  ayeshaBranch
* main
```

```
Ayeshas-MacBook-Pro:csc207-git-in-teams-example macbookpro15$ git merge ayeshaBranch
Updating 2c49b1b..7bcf700
Fast-forward
 SimpleObject.java | 13 ++++++++++++-
 1 file changed, 12 insertions(+), 1 deletion(-)
Ayeshas-MacBook-Pro:csc207-git-in-teams-example macbookpro15$ git branch
  ayeshaBranch
* main
Ayeshas-MacBook-Pro:csc207-git-in-teams-example macbookpro15$ git push
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/Daniel-Laufer/csc207-git-in-teams-example.git
   2c49b1b..7bcf700  main -> main
```

Git Flow

Step 5: Retrieving the changes

Another teammate might now try to do the same thing on his or her machine. But when this person pushes, they may encounter an error:

```
➡  csc207-git-in-teams-example $ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

```
➡  csc207-git-in-teams-example $ git merge danBranch
Updating 2c49b1b..140ba40
Fast-forward
 SimpleObject.java | 5 +++++
 1 file changed, 5 insertions(+)
```

```
➡  csc207-git-in-teams-example $ git push
To https://github.com/Daniel-Laufer/csc207-git-in-teams-example.git
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/Daniel-Laufer/csc207-git-in-teams-example.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Git Flow

Step 5: Retrieving the changes

This teammate might then try to do a git pull, and encounter this error:

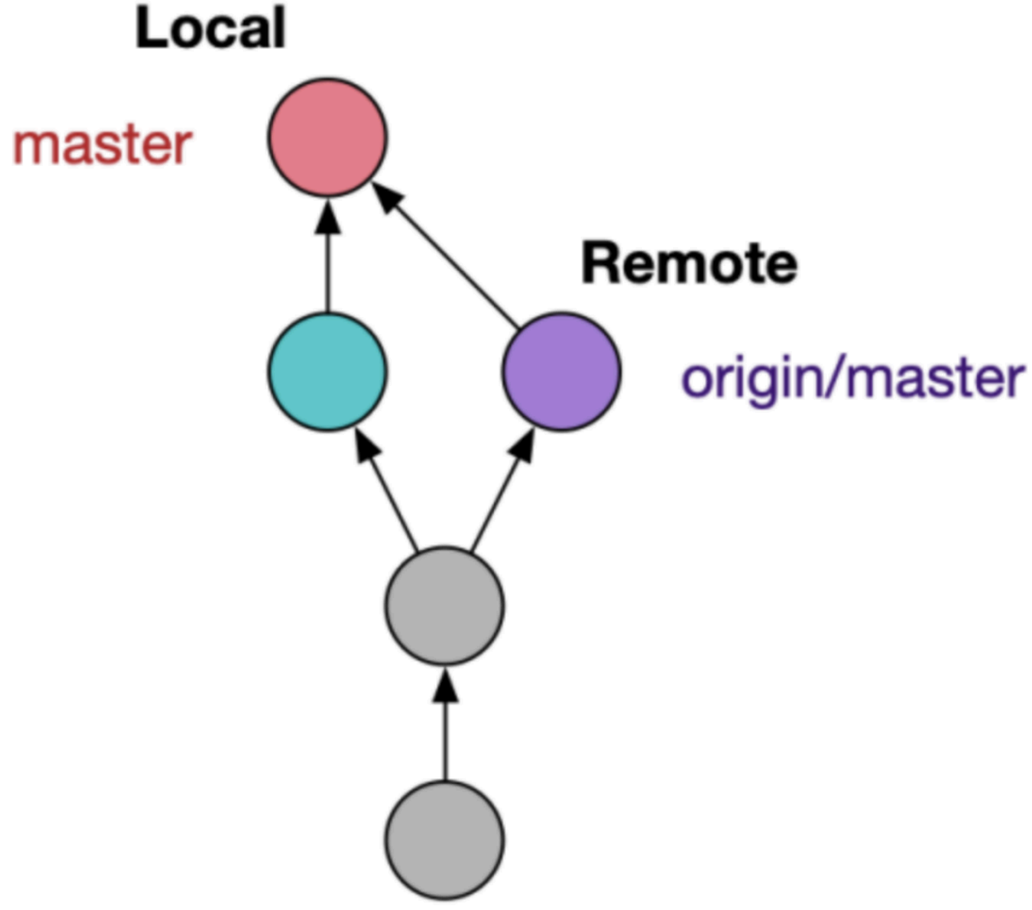
```
➜ 1 csc207-git-in-teams-example $ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), 440 bytes | 146.00 KiB/s, done.
From https://github.com/Daniel-Laufer/csc207-git-in-teams-example
   2c49b1b..7bcf700  main          -> origin/main
 * [new branch]      ayeshaBranch -> origin/ayeshaBranch
hint: You have divergent branches and need to specify how to reconcile them.
hint: You can do so by running one of the following commands sometime before
hint: your next pull:
hint:
hint:   git config pull.rebase false  # merge
hint:   git config pull.rebase true   # rebase
hint:   git config pull.ff only       # fast-forward only
hint:
hint: You can replace "git config" with "git config --global" to set a default
hint: preference for all repositories. You can also pass --rebase, --no-rebase,
hint: or --ff-only on the command line to override the configured default per
hint: invocation.
fatal: Need to specify how to reconcile divergent branches.
```

What to do?

Git Flow

Step 5: Your options

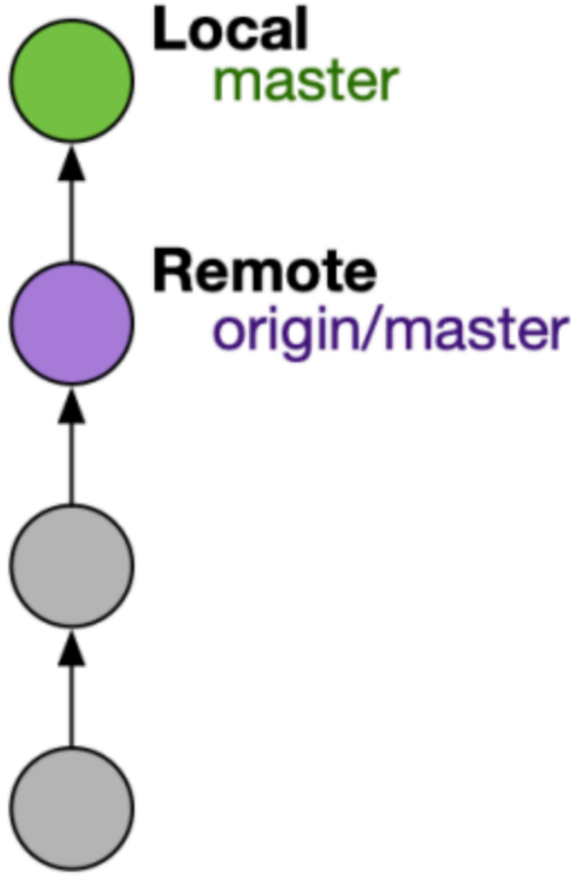
If the teammate sets **git config pull.rebase false** and then pulls, they'll end up merging their local version of the master branch with the one on the remote side (assuming this is possible). The resulting chain of commits will then look like this:



Git Flow

Step 5: Your options

If instead they set **git config pull.rebase true** and then pull, the local changes will be chained atop the master branch in the history of commits, as follows:



Git Flow

Step 5: Your options

Regardless of what option is selected, once changes are pulled from the remote and reconciled with the local code, the local commit history still needs to be **pushed** in order to see them reflected on the remote side.

Of course, more complex merge conflicts can still happen!!

Issuing a pull or merge request is one way to manage merges that may be complicated in a way that is that is transparent.

Merge Requests

Merge Requests

csc207_20239 › group_test › Merge requests › **New**

New merge request

From `master` into `develop` Change branches

Title (required)

My Merge Request

Mark as draft
Drafts cannot be merged until marked ready.

Description

Default

Choose a template

Filter

Project Templates

Default

No template

Reset template

```
describe the problem or user story being addressed.

ts as needed.

s or feature requests.

### Additional Notes
Include any extra information or considerations for reviewers, such as impacted areas of the codebase.

### Merge Request Checklists
- [ ] Code follows project coding guidelines.
- [ ] Documentation reflects the changes made.
- [ ] I have already covered the unit testing.
```

Merge Request

When you are ready to have your code reviewed, you will issue a Merge Request on GitLab.

Common issues:

It’s hard to review the code because you don’t know what the revisions are about.

Sometimes you may have to run a local server in order to even see changes, if you are working on a web application.

Merge Request

A template can help structure your comments.

```
### Description
This merge request addresses, and describe the problem or user story being addressed.

### Changes Made
Provide code snippets to illustrate

### Screenshots
Screenshots to confirm any UI changes

### Related Merge Requests
Provide links to any related code reviews, if any.

### Additional Notes
Include any extra information or considerations for reviewers, such as impacted areas of the codebase.

### Merge Request Checklist
- [ ] Code follows project coding guidelines.
- [ ] Documentation reflects the changes made.
- [ ] I have covered the unit testing.
```

You can put templates in your repository! Path to each template should be:

group_ID/.gitlab/merge_request_templates/template_name.md

Reviewing Merge Requests

Issue Number 1:

There’s a ton of new code to review and it’s overwhelming.

Solution:

Split pull request into small pieces.

Reviewing Merge Requests

Issue Number 2:

You’re not really ready to merge, you just want a code review.

Solution:

Issue a “work in progress” Merge Request. Note requests do not need to be “approved”.

Reviewing Merge Requests

Issue Number 3:

Code review comments are of the form `It seems better to xxx here.’ You wonder if that means the change is optional.

Solution:

Consider tagging comments so as to communicate if suggestions are optional or not by issuing a prefix like “[must]” or “[IMO]”. You can also tag questions with a prefix like [ask], if you like.

You might then decide to fix only the [must] parts if your time is tight, and address [IMO] comments later.

Reviewing Merge Requests

Issue Number 4:

Comments go on and on and on.

Solution:

It’s often easier and faster to discuss lengthy items in person on on Zoom.

If there is ambiguity or diversity in perspective, you might want to do ‘difficult’ code reviews in groups.

Group reviews yield diverse opinions as to potential solutions.

Reviewing Merge Requests

Issue Number 5:

Comments provoke “feels”. For example, a comment like “plz fix” might be interpreted as being “curt”.

Solution:

Try to be playful and respectful in comments; feel free to use cute emojis or whatever you think sets a nice tone :).

Notes on Style

- TAs will be assessing the JavaDoc in final submissions, but TAs will not be marking any other aspects of coding style.

- You will give feedback to your teammates about style during code reviews, however.

- To avoid having to cater to everyone's individual tastes for code formatting, consider conforming to a style guide, like Google's: https://google.github.io/styleguide/javaguide.html

- Note that there are also automated tools to help check for Java code style that you are welcome to explore, e.g. https://checkstyle.sourceforge.io/

Other team programming paradigms

Driver-Navigator Paired Programming is common:

- The **Driver** is the person at the wheel, i.e. the keyboard. A driver should always talk through what they are doing while doing it.

- The **Navigator** is in the observer position. They review code on-the-go, give directions, look out for larger issues, bugs, and note obstacles.

- The pair starts will a well defined (small) task

- They switch roles frequently (every 15 min)

UNIVERSITY OF TORONTO MISSISSAUGA

Other team programming paradigms

Ping Pong Paired Programming is a form of **Test Driven Design:**

- Start with a incomplete programming project
- Person A writes a test that fails given the current lack of code
- Person B then writes code to pass the test, and adds another test that fails given the lack of remaining code
- This continues until there are no more tests to be written and the code passes all tests.
- You are welcome to experiment with these paradigms in your groups!