

# Lab 2.3: Implementing a complete Git Flow using Git and GitHub

---

## Background

The print magazine *Flavor* has asked your firm to build and maintain an app so they can share recipes with their hungry audience.

The typical development cycle for the project is as follows:

- On the 1<sup>st</sup> of every month, your team receives a list of each writer's pick for recipe of the month.
- On the 15<sup>th</sup> of every month, the new version of the app is pushed to a staging server where the magazine's editors can do a final review.
- On the last day of the month, the new version of the app is pushed to production.

Development for this project began in January. It is now the beginning of February and the development cycle for v1.1 of the app has begun.

## Getting Started

Your team will be split between Developers and one or more Maintainers. Maintainers will be responsible for cutting new releases and accepting Pull Requests from other team members. Maintainers should have write access to this repository and Developers should have read access.

## Project Structure

- Application code can be found in the `/app/` folder.
- The application contains a file named `VERSION` that contains the major, minor, and patch numbers for the project.
- Source files are written in markdown and can be identified by the `.md` file extension.

## Next

Next we will walk through the process of creating a GitHub Fork and local clone of this repository.

### 1. Setup

**1- Fork this project.** ([https://github.com/emileastih1/EfreiLesson2\\_GitFlow.git](https://github.com/emileastih1/EfreiLesson2_GitFlow.git))

1. Open the link
2. Click the "fork" button, and choose your personal GitHub account if prompted

---

  - Please wait until everyone has caught up.



## Activities

Follow along with the activities below to get yourself ready for the rest of the walkthrough.

## 2 - Clone your Fork

### All Team Members

Clone this project to your local machine:

```
$ cd ~/my/parent/directory
$ git clone https://github.com/source-username/repository-name.git
# clone the fork repository from GitHub

$ cd repository-name
# change working directory to the cloned repository
```

## Verification :

View existing remotes:

```
$ git remote  
origin
```

```
$ git remote -v  
origin https://github.com/source-username/repository-name.git (fetch)  
origin https://github.com/source-username/repository-name.git (push)
```


You should see an origin remote that points to the source GitHub project:

```
$ git remote show origin  
* remote origin  
Fetch URL: https://github.com/source-username/repository-name.git  
Push URL: https://github.com/source-username/repository-name.git  
HEAD branch: master  
Remote branches:  
    develop tracked  
  
    master trackedgit remote  
Local branch configured for 'git pull':  
    master merges with remote master  
Local ref configured for 'git push':  
    master pushes to master (up to date)
```

View existing branches:

```
$ git branch  
# show local branches  
* master  
Git  
$ git branch -r  
# show remote branches  
origin/HEAD -> origin/master  
origin/develop  
origin/master  
  
$ git branch -a  
# show all branches  
* master  
remotes/origin/HEAD -> origin/master  
remotes/origin/develop  
remotes/origin/master
```

---

  - Please wait until everyone has caught up.



---

## 3 - Add Remote for your GitHub Fork

## All Team Members

Add a me remote:

```
$ git remote add me https://github.com/your-username/repository-name.git
# add the me remote
```

```
$ git remote -v
origin https://github.com/source-username/repository-name.git (fetch)
origin https://github.com/source-username/repository-name.git (push)
me https://github.com/your-username/repository-name.git (fetch)
me https://github.com/your-username/repository-name.git (push)
```

You should see a me remote that points to your GitHub Fork repository:

```
$ git remote show me
* remote me
Fetch URL: https://github.com/your-username/repository-name.git
Push URL: https://github.com/your-username/repository-name.git
HEAD branch: master
Remote branches:
  develop new (next fetch will store in remotes/me)
  master new (next fetch will store in remotes/me)
Local ref configured for 'git push':
  master pushes to master (up to date)
```

Maintainers will need to create branches and push directly to the source repository.

All team members will need to pull changes from the source repository in order to branch from for feature branches.

Fetch branch data from the origin remote:

```
$ git fetch origin (git fetch me) or git remote update
From https://github.com/source-username/repository-name
* [new branch]      develop    -> origin/develop
* [new branch]      master     -> origin/master

$ git branch -a
* master
remotes/origin/HEAD -> origin/master
remotes/origin/develop
remotes/origin/master
remotes/me/develop
remotes/me/master
```

---

  - Please wait until everyone has caught up.



---

## 4 - Configure Local Branches

### All Team Members

By now you should have noticed that you do not have a local develop branch

```
$ git branch
* master
```

Create a develop branch that tracks from origin's develop branch:

```
$ git checkout -b develop --track origin/develop
Branch develop set up to track remote branch develop from origin
```


Notice that viewing the details for the origin remote indicates that the local develop and master branches are configured to push to and pull from the source

GitHub repository's branches:

```
$ git remote show origin
...
  Local branches configured for 'git pull':
    develop merges with remote develop
    master merges with remote master
  Local branches configured for 'git push':
    develop pushes to develop (up to date)
    master pushes to master (up to date)
```

You should now be ready to move on to the rest of the walkthrough. If you'd like to see the repository you've created on your local machine in GitHub desktop, you can add a repository by choosing a local path.

---

  - Please wait until everyone has caught up.



---

## Next

Next we will walk through the process of creating feature branches, publishing changes to GitHub, and making a request to merge changes into the source repository using a Pull Request.

## 2. Feature Branches

Date	Phase
February 1 <sup>st</sup>	Development
v1.0 was pushed to production yesterday without issue. All code from the release has been merged into the master and develop branches of the project. The new recipes for February have been sent to the team:	

Writer	Recipe
Cuba Pudding Jr.	<a href="http://allrecipes.com/recipe/228654/quick-oatmeal-pancakes/">http://allrecipes.com/recipe/228654/quick-oatmeal-pancakes/</a>
Eggs Benny	<a href="http://allrecipes.com/recipe/174361/asparagus-with-cranberries-and-pine-nuts/">http://allrecipes.com/recipe/174361/asparagus-with-cranberries-and-pine-nuts/</a>
John Lemon	<a href="http://allrecipes.com/recipe/18241/candied-carrots/">http://allrecipes.com/recipe/18241/candied-carrots/</a>
Madame Croque	<a href="http://www.realsimple.com/food-recipes/browse-all-recipes/roast-pork-sandwich/">http://www.realsimple.com/food-recipes/browse-all-recipes/roast-pork-sandwich/</a>

## Activities

Follow along with the activities below to walk through the process of creating a feature branch and creating a Pull Request to merge your changes into the source repository.

### 1 - Create a Feature Branch

#### All Team Members

Choose a writer — you will add their recipe to the project within a new feature branch. If there are more writers than people, that is fine, only one feature branch will ultimately be merged.

Create a feature branch off of the develop branch that contains the writer's name and the month of the pick:

```
$ git checkout develop
```

```
$ git checkout -b cuba-pudding-jr-feb
```

```
$ git branch
```

```
* cuba-pudding-jr-feb  
  develop  
  master
```

💡 Feature branches will be named such that someone else can look at what branches are in progress and get a rough idea of what work is being done on each branch.

💡 You can create a branch without immediately checking out the branch via the `git branch` command:

```
$ git branch cuba-pudding-jr-feb
```

---

👤 🖐️ - Please wait until everyone has caught up.



## 2 - Make Changes to the Project

### All Team Members

In your text editor, make the following changes:

1. Add the new recipe under the `/app/recipe/feb/` directory.
2. Update the writer's page in the `/app/writer/` directory.
3. Update the main page `/app/index.md`.

Since other people are going to be making changes at the same time, be careful not to make changes to lines of code that are not relevant to your change.

---

👤 🖐️ - Please wait until everyone has caught up.



## 3 - Review Changes

## All Team Members

If you view the current git status, you will see 2 files with unstaged changes and a new folder that has not been tracked by git:

```
$ git status
On branch cuba-pudding-jr-feb
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
```

```
    modified:   app/index.md
    modified:   app/writer/cuba-pudding-jr.md
```

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
```

```
    app/recipe/feb/
```

no changes added to commit (use "git add" and/or "git commit -a")

---

  - Please wait until everyone has caught up.

---

## 4 - Stage and Commit Changes

Stage all of the changes that you've made thus far:

```
$ git add app/*
# Careful, using a wildcard adds any file that has changed that matches the
pattern.
```

```
$ git status
On branch cuba-pudding-jr-feb
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
```

```
    modified:   app/index.md
    new file:   app/recipe/feb/quick-oatmeal-pancakes.md
    modified:   app/writer/cuba-pudding-jr.md
```



Now that all of your changes have been staged, commit them with an appropriate message:

```
$ git commit -m "Adding Cuba Pudding Jr.'s Feb Recipe"
```

---

  - Please wait until everyone has caught up.



## 5 - Push Changes

Now that your changes have been committed, let's get them published to your Fork on GitHub:

\$ git push will not work directly so you need to:

```
git push --set-upstream me branchName
```

---

  - Please wait until everyone has caught up.



## 6 - Open a Pull Request

On your GitHub fork, you should see a block at the top indicating that you've recently pushed a branch. If this is visible, click the "Compare & Pull Request" button to the right of your feature branch. If it is not available, choose your branch in the dropdown just above the code directory listing and then click the "New Pull Request" button to the right of the dropdown.

On the Pull Request interface, make sure that the base fork is `source-username/repository-name` and the base branch is `develop`. This means that you are requesting to merge your changes into the `develop` branch of the source repository. At this time also make sure that the head fork and compare branches match your GitHub Fork and feature branch, respectively.

Please draft a message documenting the changes that you've made and then click the green "Create pull request" button.

---

  - Please wait until everyone has caught up.



---

## Next

Next we will walk through the process of reviewing a Pull Request, viewing someone else's code locally, and merging a Pull Request.

### 3.Code Review

**Date**

**Phase**

February 2<sup>nd</sup>

Development

Now that all of the mods for the next release have been made, it is time to review and get code merged.

## Activities

Follow along with the activities below to walk through the process of reviewing a Pull Request, viewing someone else's code locally, and merging a Pull Request.

### 1 - Review a Pull Request

#### All Team Members

Navigate to the source repository on GitHub, click the "Pull Requests" tab, and click on the Pull Request opened by the team member sitting to your right.


Review the following things:

1. Verify that the user is requesting to merge their feature branch into the `develop` branch.
2. Check to see if there is any important information on the Conversation tab.
3. Check the Commits tab to get a high-level view of the individual changes that were made.
4. Verify that the changes on the Files Changed tab look correct to you.

If anything looks wrong, please make a quick comment outlining the issue. If there is a specific line of code, you can comment directly on the line from the Files Changed tab.

If everything looks okay, please make a quick comment indicating that you believe the code is ready to be merged.

---

  - Please wait until everyone has caught up.

---

## 2 - View Contributor Code Locally

### All Team Members

If you are curious to run someone else's code locally, you will need to set up a remote to their GitHub fork and grab their latest commits.

Create a remote pointing to the person to your right's Fork:

```
$ git remote add teammate https://github.com/teammate-username/repository-name.git
# add the teammate remote
```

```
$ git remote -v
me https://github.com/your-username/repository-name.git (fetch)
me https://github.com/your-username/repository-name.git (push)
origin https://github.com/source-username/repository-name.git (fetch)
origin https://github.com/source-username/repository-name.git (push)
teammate https://github.com/teammate-username/repository-name.git (fetch)
teammate https://github.com/teammate-username/repository-name.git (push)
```

Fetch their commits and view their branches:

```
$ git fetch teammate
# fetch the latest from teammate

$ git branch -r --list teammate/*
# show all remote branches for the teammate
teammate/develop
teammate/eggs-benny-feb
teammate/master
```

Checkout a tracking branch:

```
$ git checkout eggs-benny-feb
Branch teammate-feature-branch set up to track remote branch eggs-benny-feb from
teammate.
Switched to a new branch 'teammate-feature-branch'
```

💡 As long as only one of your remotes has a branch called eggs-benny-feb, it knows to create a local tracking branch.

💡 Now you've switched the code in your directory to a branch containing all of the changes committed to your teammate's branch. If you have your project open in sublime or another text editor you will notice that the source files contain your teammate's changes.

If you run the `git log` command you can see a log of the latest commits made to the current branch:

```
$ git log
commit 6b7a7c2bc46753d0d394758b8fd04bcd3b4cf896
Author: My Teammate <teammate@gr8people.com>
Date: Mon Feb 8 12:58:20 2016 -0500
```

Adding Eggs Benny February Recipe

...

💡 Press enter to scroll through lines & `ctrl+c` to exit the log.

Once you are done, you can delete this branch:

```
$ git checkout develop
# switch to develop

$ git branch -d teammate-feature-branch
# delete tracking branch
```

||

👮 🖐️ - Please wait until everyone has caught up.

🚧 🚧 🚧 🚧 🚧 🚧 🚧 🚧 🚧 🚧 🚧 🚧 🚧 🚧 🚧 🚧

---

## 3 - Accept Pull Requests


### Maintainers

Open the list of open Pull Requests on GitHub and split them between all maintainers.

If there were more team members than writers, close Pull Requests for duplicate writers using the "Close pull request" button at the very bottom of the page.

For all remaining Pull Requests and click the green "Merge pull request" button. Assuming there are no merge conflicts, GitHub will automatically merge the code into the `develop` branch. If there are merge conflicts, for this exercise, close the pull request using the "Close pull request" button at the bottom. In practice, merge conflicts should be corrected by the team member requesting to merge changes. Sometimes this is not feasible and a maintainer may have to merge the changes manually.

---

  - Please wait until everyone has caught up.



---

## Next

Next we will walk through the process of fetching the latest changes to the source repository.

## 4. Fetching Latest

Date	Phase
February 3 <sup>rd</sup>	Development

Now that mods have been merged into the development branch, it is prudent to get the latest version of the code locally.

## Activities

Follow along with the activities below to walk through the process of pulling the latest commits to the `origin/develop` branch into your local tracking branch.

### 1 - Pull Latest from `develop`

#### All Team Members

Pull the latest commits to the `develop` branch from the source repository:

```
$ git checkout develop
```

```
$ git pull
```

## Next

Next we will walk through the process of applying a hotfix to the code in production.

## 5.Hotfix

Date	Phase
February 5 <sup>th</sup>	Hotfix

This morning the project manager received a frantic call from the EIC of *Flavor* magazine. The app was so successful that the writers have been inundated with emails from readers submitting their own recipes for consideration. The number of emails has been so great that their inboxes are completely useless. The magazine wants your team to remove the writers' email addresses from the app ASAP.

### Activities

Follow along with the activities below to walk through the process of creating a hotfix branch, creating a feature branch for the changes, getting changes merged into the hotfix, then merging the hotfix into both the `master` and `develop` branches.

## 1 - Create the Hotfix Branch

### Maintainers

Create a branch off of `master` named `hotfix-1.0.1`:

```
$ git checkout master
# switch to master branch
```

```
$ git checkout -b hotfix-1.0.1
# create & switch to hotfix branch
```

Bump the patch number contained in the [VERSION](#) file:

```
major=1
minor=0
patch=1
```

Stage and commit your change:

```
$ git add app/VERSION
# stage changes to the version file
```

```
$ git commit -m "Bump version to 1.0.1"
```

---

  - Please wait until everyone has caught up.

---

## 2 - Publish the Hotfix Branch

### Maintainers

Choose a maintainer to publish the hotfix branch. This maintainer should push the branch to origin:

```
$ git push -u origin HEAD
```

---

  - Please wait until everyone has caught up.

---


## 3 - Make Changes in Feature Branch

### Developers

Fetch the latest from origin and create a local tracking branch for the hotfix:

```
$ git fetch origin
# fetch latest from origin

$ git checkout hotfix-1.0.1
# checkout the hotfix branch
Branch hotfix-1.0.1 set up to track remote branch hotfix-1.0.1 from origin.
Switched to a new branch 'hotfix-1.0.1'
```

 As long as only one of your remotes has a branch called hotfix-1.0.1, it knows to create a local tracking branch.

Create a feature branch named `remove-emails` off of the hotfix branch.

```
$ git checkout -b remove-emails
```

After removing the email addresses from the main page (</app/index.md>), stage and commit the change:

```
$ git add app/index.md  
$ git commit -m "Remove email addresses from app"
```

---

  - Please wait until everyone has caught up.



---

## 4 - Publish and Request to Merge Feature Branch

### Developers

Choose a developer to publish the fixup branch and create a Pull Request against the hotfix branch.

Publish the branch:

```
$ git push -u me HEAD
```

Navigate to your GitHub fork and open the pull request, making sure to request to merge changes into the hotfix-1.0.1 branch.

---

  - Please wait until everyone has caught up.



---

## 4 - Finish Hotfix Branch

### Maintainers

Choose a maintainer to accept the new pull request to merge the hotfix branch into master,.

---

  - Please wait until everyone has caught up.





---

## 5 - Merge Back into `develop`

Next we will work to get the hotfix merged back down into `develop`, so switch to the hotfix branch and pull down all the latest changes:

```
$ git checkout hotfix-1.0.1
# switch to hotfix branch
```

```
$ git pull
# pull latest changes
```

💡 Running `git pull` on a tracking branch will automatically fetch and merge changes.

Merge hotfix into `develop`, creating a new merge commit:

```
$ git checkout develop
```

```
$ git pull
```

```
$ git merge hotfix-1.0.1
```

💡 Always make sure that `develop` is up to date before merging. There may be some merge conflicts that will need to be addressed at this point.

Push the commit up to the origin repository:

```
$ git push
```

## Next

Next we will walk through the process of cutting a new release branch.

## 6. Release Branch (release-1.1)

Date	Phase
------	-------

February 15 <sup>th</sup>	Release
---------------------------	---------

It is now time to begin testing v1.1 of the app and submit it for the magazine's review

## Activities

### 1 - Create a Release Branch

#### Maintainers

Bring your local develop branch up to date with the source repository:

```
$ git checkout develop
```

```
$ git pull
```

Create a new branch off of develop named release-1.1:

```
$ git checkout -b release-1.1
```

Bump the version number to 1.1:

```
major=1
```

```
minor=1
```

```
patch=0
```

Finally, stage and commit the change:

```
$ git add app/VERSION
```

```
$ git commit -m "Bump version to 1.1"
```

## 2 - Publish the Release Branch

### Maintainers

Choose a maintainer to publish the release branch to the source repository:

```
$ git push -u origin HEAD
```

## Next

Next we will walk through the process of fixing bugs in the release branch.

## 7. Release Bugs

Date	Phase
February 24 <sup>th</sup>	Release

The magazine is happy with the work that has been done, but has asked that the recipes picked for January appear on the home screen under the heading "Last Month's Favorites". Oh and by the way, John Lemon has been fired so please would you remove him from the application.

## Activities

Follow along with the activities below to walk through the process of fixing issues in the release branch.

## 1 - Fix Issues in Feature Branches

### Developers

Fetch the latest from origin and create a local tracking branch for the release:

```
$ git fetch origin
# fetch latest from origin



$ git checkout release-1.1
# checkout the release branch
```

Choose two developers to address each issue.

- Developer 1:
  - i. Create a feature branch off of release-1.1 named last-months-favorites.
  - ii. Commit the following changes to the feature branch:
    - a. Create a section in [/app/index.md](#) titled "Last Month's Favorites".
    - b. Copy the text that was published last release and paste under "Last Month's Favorites". Be sure not to include John Lemon.
- Developer 2:
  - i. Create a feature branch off of release-1.1 named remove-john-lemon.
  - ii. Commit the following changes to the feature branch:
    - a. Delete the file `/app/writer/john-lemon.md`.
    - b. Remove references to John Lemon from [/app/index.md](#)

Other team members may also choose an issue and make the changes themselves locally to gain more practice working in feature branches.

---

  - Please wait until everyone has caught up.



---

## 2 - Publish & Request to Merge Feature Branches

## Developers

The two developers from the last step should now publish their feature branches to GitHub and open up Pull Requests against the `release-1.1` branch in the source repository:

```
$ git push -u me HEAD
```

---

  - Please wait until everyone has caught up.



---

## 3 - Merge Feature Requests into Release Branch

### Maintainers

Review the developer Pull Requests and merge them into `release-1.1`.

## Next

Finally, we will walk through the process of completing a release: merging into `master`, back down into `develop`, and creating a GitHub release tag.

## 8 Completed Release

Date	Phase
------	-------

February 28 <sup>th</sup>	Release
---------------------------	---------

It is the last day of the month and no more requests have come in from the, time to push the code to production and merge changes to the release back into the `develop` branch.

## Activities

### 1 - Merge Release Branch into `master`

#### Maintainers

Choose a maintainer to merge release-1.1 into master.

```
$ git checkout release-1.1
```

```
$ git pull
```



```
$ git checkout master
```

```
$ git pull
```

```
$ git merge release-1.1
```

```
$ git push
```

---

  - Please wait until everyone has caught up.



---

## 2 - Merge Release Branch into `develop` Maintainers


Next we will work to get changes made to the release merged back down into `develop`, so switch to the release branch and pull down all the latest changes:

```
$ git checkout release-1.1
```

```
$ git checkout develop
```

```
$ git merge release-1.1
```

```
$ git push
```

 Always make sure that `develop` is up to date before merging. There may be some merge conflicts that will need to be addressed at this point.

Choose a maintainer to push the the commit up to the source repository:

```
$ git push source develop
```

---

  - Please wait until everyone has caught up.



---

## 3 - Create a Release tag in GitHub

Choose a maintainer to create a Release in GitHub:

1. On the source repository page, click the "# releases" link in the stats bar at the top (or append /releases to the repo URL).
2. Click new release.
3. Name the release tag v1.1, choose the master branch, and document the changes comprising the release.

The release feature in GitHub is an easy way to keep track of changes associated with each release. Files may also be uploaded and attached to the release, so they can also act as a place to store any additional files related to the release that are not tracked in version control.

 **We're done!**

You did it! Feel free at this time to navigate around the source repository, yours and others' forks and reflect back on all the hard work that was done to get v1.1 out the door 😊.