**Scenario #1**

Let's walk through the Git commands that go along with each step in the scenario you just observed in the video.

**Step 1: You have a local version of this repository on your laptop, and to get the latest stable version, you pull from the develop branch.**

***Switch to the develop branch***

*git checkout develop*

***Pull the latest changes in the develop branch***

*git pull*

**Step 2: When you start working on this demographic feature, you create a new branch called demographic, and start working on your code in this branch.**

***Create and switch to a new branch called demographic from the develop branch***

*git checkout -b demographic*

***Work on this new feature and commit as you go***

*git commit -m 'added gender recommendations'  
git commit -m 'added location specific recommendations'  
...*

**Step 3: However, in the middle of your work, you need to work on another feature. So you commit your changes on this demographic branch, and switch back to the develop branch.**

***Commit your changes before switching***

*git commit -m 'refactored demographic gender and location recommendations '*

***Switch to the develop branch***

*git checkout develop*

**Step 4: From this stable develop branch, you create another branch for a new feature called friend\_groups.**

***Create and switch to a new branch called friend\_groups from the develop branch***

*git checkout -b friend\_groups*

**Step 5: After you finish your work on the friend\_groups branch, you commit your changes, switch back to the development branch, merge it back to the develop branch, and push this to the remote repository’s develop branch.**

***Commit your changes before switching***

*git commit -m 'finalized friend\_groups recommendations '*

***Switch to the develop branch***

*git checkout develop*

***Merge the friend\_groups branch into the develop branch***

*git merge --no-ff friends\_groups*

***Push to the remote repository***

*git push origin develop*

**Step 6: Now, you can switch back to the demographic branch to continue your progress on that feature.**

***Switch to the demographic branch***

*git checkout demographic*

# Scenario #2

Let's walk through the Git commands that go along with each step in the scenario you just observed in the video.

#### Step 1: You check your commit history, seeing messages about the changes you made and how well the code performed.

##### *View the log history*

*git log*

#### Step 2: The model at this commit seemed to score the highest, so you decide to take a look.

##### *Check out a commit*

*git checkout bc90f2cbc9dc4e802b46e7a153aa106dc9a88560*

After inspecting your code, you realize what modifications made it perform well, and use those for your model.

#### Step 3: Now, you're confident merging your changes back into the development branch and pushing the updated recommendation engine.

##### *Switch to the develop branch*

*git checkout develop*

##### *Merge the friend\_groups branch into the develop branch*

*git merge --no-ff friend\_groups*

##### *Push your changes to the remote repository*

*git push origin develop*

# Scenario #3

Let's walk through the Git commands that go along with each step in the scenario you just observed in the video.

#### Step 1: Andrew commits his changes to the documentation branch, switches to the development branch, and pulls down the latest changes from the cloud on this development branch, including the change I merged previously for the friends group feature.

##### *Commit the changes on the documentation branch*

*git commit -m "standardized all docstrings in process.py"*

##### *Switch to the develop branch*

*git checkout develop*

##### *Pull the latest changes on the develop branch down*

*git pull*

#### Step 2: Andrew merges his documentation branch into the develop branch on his local repository, and then pushes his changes up to update the develop branch on the remote repository.

##### *Merge the documentation branch into the develop branch*

*git merge --no-ff documentation*

##### *Push the changes up to the remote repository*

*git push origin develop*

#### Step 3: After the team reviews your work and Andrew's work, they merge the updates from the development branch into the master branch. Then, they push the changes to the master branch on the remote repository. These changes are now in production.

##### *Merge the develop branch into the master branch*

*git merge --no-ff develop*

##### *Push the changes up to the remote repository*

*git push origin master*

### Resources

Read [this great article](http://nvie.com/posts/a-successful-git-branching-model/) on a successful Git branching strategy.

### Note on merge conflicts

For the most part, Git makes merging changes between branches really simple. However, there are some cases where Git can become confused about how to combine two changes, and asks you for help. This is called a merge conflict.

Mostly commonly, this happens when two branches modify the same file.

For example, in this situation, let’s say you deleted a line that Andrew modified on his branch. Git wouldn’t know whether to delete the line or modify it. You need to tell Git which change to take, and some tools even allow you to edit the change manually. If it isn’t straightforward, you may have to consult with the developer of the other branch to handle a merge conflict.