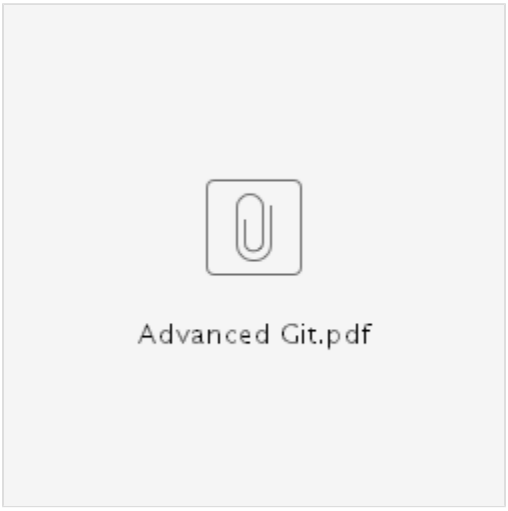
Intro to Git

Basics

- The convention edX uses for naming branches is <username/nameofbranch>. For example, `johndoe/shinynewbutton` or `johndoe/roundingerror`.
- To make your first branch:
    - **git checkout -b "johndoe/myfirstfeature"**.
    - "checkout" lets you switch to whatever branch you'd like to work on; running "checkout" with the "-b" flag indicates you want to *cr eate* this new branch and then switch to it.
- To display all of the local branches you have, with a star next to the branch that's currently active / checked out:
    - **git branch**
- Suppose you edit the file foo.bar. To add the file to the list of files to be committed:
    - **git add foo.bar**
- To display all files that have been modified and/or added to the current commit:
    - **git status**
- To commit any added files to your local git repository:
    - **git commit -m "My first feature!"**
- To show diffs from a commit:
    - **git show <commit_hash>**

Advanced

David Baumgold gave this presentation on Advanced Git at PyCon 2015



Branching

- Every time you want to make a new branch, follow this command sequence:
    - **git checkout master** (so that you're branching off master)
    - **git pull** (so that master is up to date)
    - **git checkout -b newbranch**
- If you'd like to make a *newbranch* based on *oldbranch*, don't branch off of *oldbranch*—branch off master, then **git cherry-pick** each of the commits you want from *oldbranch*.
- To push your branch to origin for other developers to view it:
    - If the branch doesn't exist yet on origin:
        - **git push -u origin newbranch** (short for **git push --set-upstream origin newbranch**)
        - A new local branch has no upstream until it's pushed the first time.
        - This command set the upstream and specifies the name of the remote branch (which is almost always the same name as the local branch).
    - After the branch exists:
        - **git checkout newbranch**
        - **git push**
- To view another developer's branch in your local git repository:

    - **git checkout -t origin/<branchtoview>**
- Delete old branches that are no longer in use—since Jenkins automatically builds all branches periodically, we don't want Jenkins to waste time building, say, a branch that's already been merged into master. To do this deletion:
    - **git push origin :branchtobedeleted** to delete the remote branch
    - **git branch -d branchtobedeleted** to delete the local branch.
- Before submitting a pull request, read How to Rebase a Pull Request and set the default push configuration to "simple" by running:
    - **git config --global push.default simple**
- To submit a pull request: Using Pull Requests at Github
- Code review guidelines. Make sure at least two people review your PR before you merge!
- todo: include bit about pulling remote branches

Git Rebase

Problem: You'd like to squash a bunch of your commits together, you have twenty commits and most of them are things like "fixing," "added more stuff", etc. Ideally you'd like to have a small number of commits with reasonable messages, like, "Added feature X," "Added unit tests for feature X," "Wrote scss for feature X."

Solution: **git rebase -i**. This should take you to a text editor with a list of your commits. Sometimes you'll do this and no commits will appear (probably when git thinks you've already rebased those last few commits); if that's the case, try **git rebase -i HEAD~X**, only replace "X" with however many commits you want to go back—i.e. **git rebase -i HEAD~20** to go back 20 commits.

So now your commit list should appear! You can now decide what you want to have happen to each of your commits—git shows you some commands at the bottom of the window. Probably you'll want to put an "r" next to commits you want to keep *but* whose commit messages you want to reword, an "f" for any commit whose message you want to discard and just squish into the previous one, and "p" for anything you want to keep. For example, say you make the following changes for your rebase:

```
p 61c25fa Added new section
f 1038vf oops syntax error
f 502fff fixed some quality errors
f 5605a refactored ugly code
r 80912 unit tests part one
f 83428b unit tests part two
```

The rebase will prompt you to change the commit message for "unit tests part one"; let's say we change that to "Added unit tests". Then our final result will look like this:

```
p 61c25fa Added new section
r 80912 Added unit tests
```

Note that you **can** rebase an open pull request, so feel free to combine smaller commits together even if it's already in review or something.

Git Bisect

Problem: You just ran git fetch, and now your shiny new feature no longer works. That git fetch added twenty commits; how are you supposed to find which commit broke things?

Solution:

- **git bisect start**: Tells git to begin bisecting.
- **git bisect good 908b51**: the hash here being whatever the hash of the last known good commit is
- **git bisect bad**: indicates that the commit we're currently on is bad
- git bisect will then do a binary search of all the commits between the "good" and "bad" ones. Each time, it gets you a new commit that you can investigate—run the LMS/CMS, run tests, do whatever to figure out whether the commit is good or bad. Then enter either "git bisect good" or "git bisect bad"
- Once you're done, git bisect will say something like "First bad commit is 720aef41", and you can see what in that commit introduced the problem.
- At any point you can run **git bisect reset** to stop bisecting and go back to wherever you were when you started the bisect.

Text Editor

By default Git seems to use nano for text editing (when you're prompted to change commit messages, resolve merge conflicts, do an interactive rebase, etc). You can change the default by running **git config --global core.editor vim** , or whatever your text editor of choice is.