**Intro to Git in the Pienaar Lab**

Last updated: 03/10/2021 by Lexi Petrucciani apetruc@purdue.edu

Welcome to the wonderful world of git! We are gonna start from the basics, introduce you to our work flow, and leave you with some useful commands. Hopefully, from there most things can be googled.

**Basics**

*Cloning a repository*

Let’s start by creating a folder and cloning our repository into it. First you are going to open the terminal and use ls and cd to navigate to where you want to store your directory. You can make a new folder in command line with mkdir folderName. I recommend creating a folder to store all the folders that house your repositories. For example, I have a folder named git that contains 3 folders each with its own repository. After that go to the online repository, click the code button, and copy the URL. Go back to the command line and run git clone URL. You should now have a local repository of the remote repository!

If you are having trouble cloning in the normal way (using that https address), you can also try cloning using ssh. We followed this tutorial on a Windows computer using git bash, except instead of

ssh -T [git@bitbucket.org](mailto:git@bitbucket.org) we used git@github.itap.purdue.edu

Now you can use git clone 'ssh-address', and it should work.

<https://confluence.atlassian.com/bitbucket/set-up-ssh-for-git-728138079.html>

*Setting up to use our git flow*

We are going to edit the git config file to add some aliases. These are basically shortcuts, so you don’t have to write as much in the command line each time. They will make the overall flow easier. So, to edit this file you should type in the following while in any git repository. The code command works if you are using VScode, or vi will function in the terminal.

code ~/.gitconfig or vi ~/.gitconfig

When you open this file it should look like this with your name and email.

# This is Git's per-user configuration file.  
[user]  
# Please adapt and uncomment the following lines:  
    name = apetruc  
    email = [apetruc@purdue.edu](mailto:apetruc@purdue.edu)

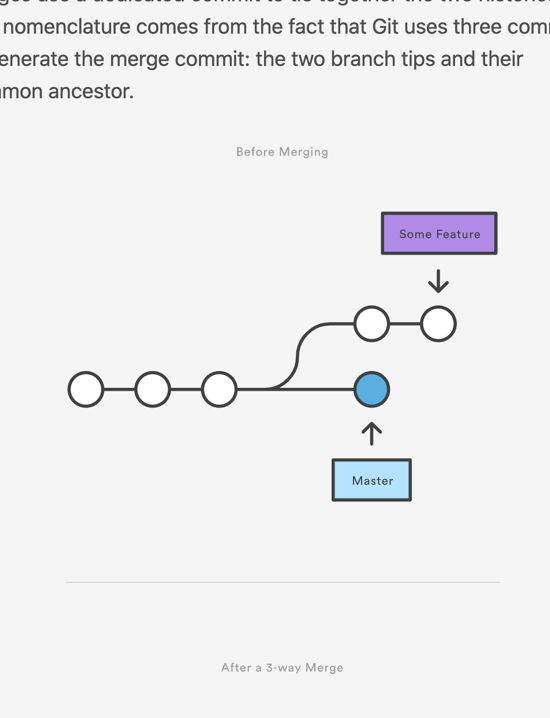
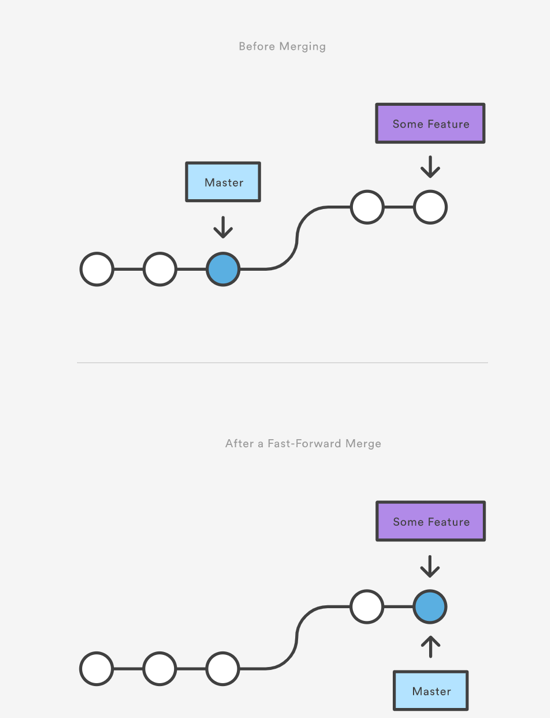
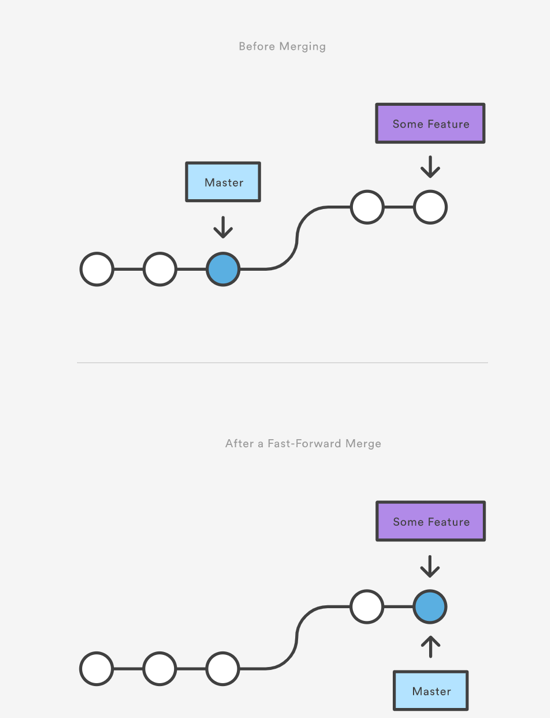
We are adding from [alias] down to get a final product that looks like this.

# This is Git's per-user configuration file.  
[user]  
# Please adapt and uncomment the following lines:  
    name = apetruc  
    email = [apetruc@purdue.edu](mailto:apetruc@purdue.edu)  
[alias]  
lol = log --graph --decorate --pretty=oneline --abbrev-commit  
lola = log --graph --decorate --pretty=oneline --abbrev-commit --all  
#  
# Working with branches  
#  
# Get the current branch name (not so useful in itself, but used in  
# other aliases)  
branch-name = "!git rev-parse --abbrev-ref HEAD"  
# Push current branch  
put = "!git push origin $(git branch-name)"  
# Pull without merging  
get = "!git pull origin $(git branch-name) --ff-only"  
# Pull Master without switching branches  
got = "!f() { CURRENT\_BRANCH=$(git branch-name) && git checkout $1 && git pull origin $1 --ff-only && git checkout $CURRENT\_BRANCH;  }; f"

You can use Esc :wq to save and exit the file when you are done editing.

This is creating a few aliases, the most important for us are get and put.

You can see get is defined as git pull origin $(git branch-name) --ff-only. This will pull the code of the branch you are on from the online repository, but —ff-only means it will only do fast forward. Fast forward will only work if the two branches haven’t diverged. This is just there to make sure no weird merges happen on accident.

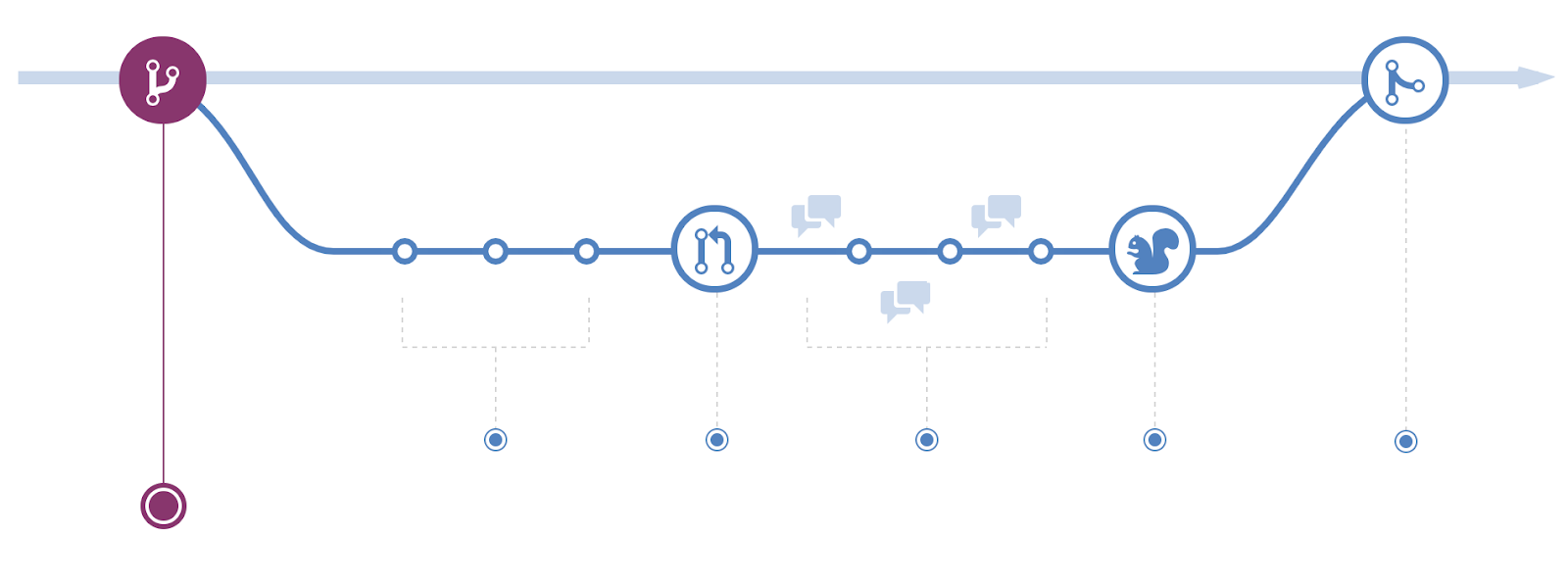


These images show an acceptable ff merge before and after and a merge that would cause a problem if ff was used. Please see <https://www.atlassian.com/git/tutorials/using-branches/git-merge> for more info. You can also see put is defined as git push origin $(git branch-name). So, it just pushes to the online repository of the branch you are on. Due to the alias code these can be used on any branch without typing branch names.

**General git flow of the lab**

*Theoretical Description*

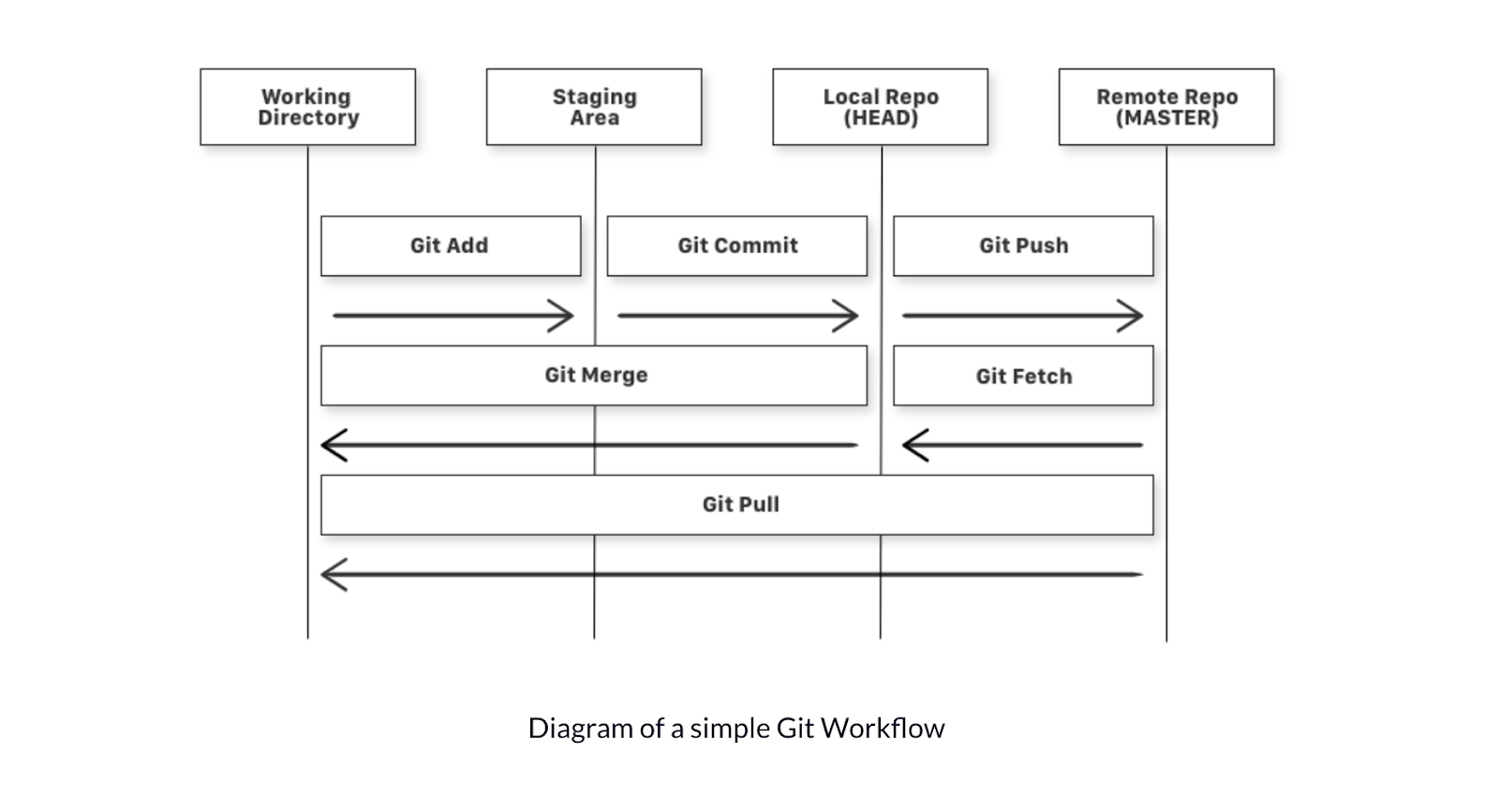
Now that we are done getting set up let’s talk a little about standard work flow. Our workflow is similar to what is described in this guide (<https://guides.github.com/introduction/flow/>).



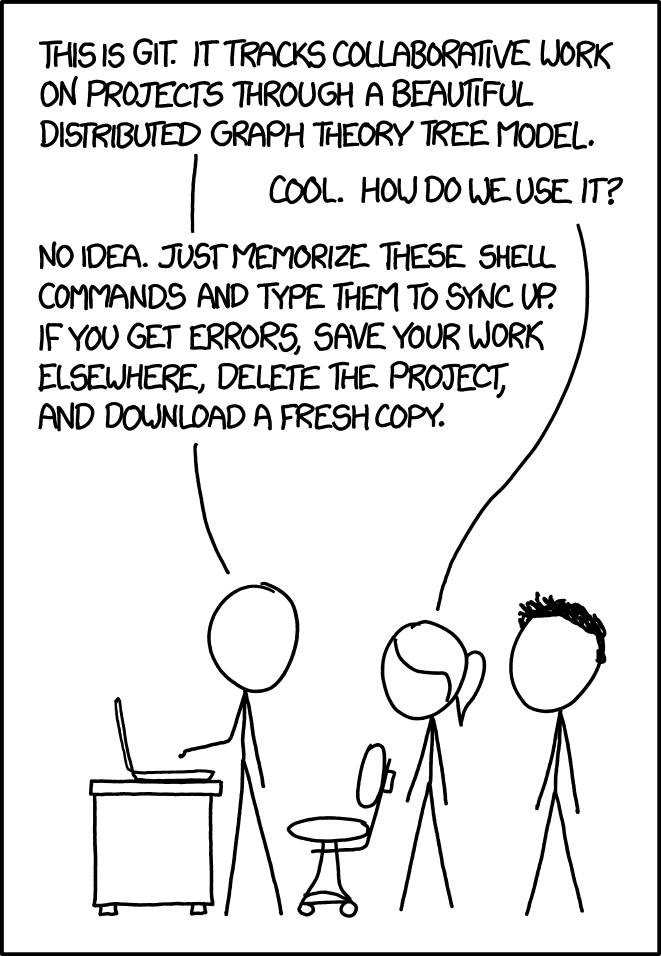
The basic steps are creating a new branch, making commits, opening a pull request (PR), reviewing code, and merging back into master/main.

*Practical Description*

Now, let’s actually do this in practice. Starting in the master branch, you will type git get. This will ensure you are up to date with the online repository on the master branch. Next, you will create and move to a new branch with branch name by typing git checkout -b BRANCHNAME. After this you are on a new branch that is up to date with the master but separate so you can make changes without messing with the master unintentionally. This is where you will make changes, stage them (git add), and commit them (git commit -m “message”). Next you will run git put to push your changes to the online repository. This will keep the remote repository up to date with your local commits. How git add, git commit, and git push/put are related can be seen below.



This is a helpful visualization of where things are in git and how different commands update things. (<https://www.freecodecamp.org/news/learn-the-basics-of-git-in-under-10-minutes-da548267cc91/>)

After you have made all the changes/commits to your feature branch that you desire, you must go online and open a pull request for merging your branch to the master. Navigate to the repository, click on branches, and then “new pull request” of the desired branch. You can see before merging if there will be any errors. After taking care of any merge conflicts either online or in the command window, please squash merge into master.

*Short summary of workflow*

git get

git checkout -b BRANCH

# make changes/commit etc

git put

# go online to squash merge - new pull request

xkcd.com

**Helpful terminal/git commands**

We leave you with a table of commands we found useful. If you need help or get lost, just ask one of us or google it. ☺ If you every feel bad just remember this xkcd comic. Happy coding!

|  |  |  |
| --- | --- | --- |
| Command | Function | Notes |
| cd folderName | Moves into folder | You can use tab to autocomplete folder/file names |
| cd .. | Moves up one directory | . refers to the current directory and .. refers to the parent directory |
| cd - | Goes to the previous directory | This – can also be used to switch to the previous branch in git |
| ls | List the contents of the directory |  |
| mkdir folderName | Makes a new directory with folder name |  |
| Esc :wq enter | To exit vi editor | Escape will let you leave insert mode (can enter insert mode by pressing i), w stands for write which is like save, and q is for quit. |
| git status | Will show you what changes have been made and whether they are staged for commit |  |
| git add fileName | Stages changes in filename to be committed |  |
| git add -A | Stages all changes to be committed |  |
| git stash | Will store staged changes as work in progress | This can be used as a trash can to get rid of changes you don’t wanna keep. Just add changes and stash them. |
| git commit -m “message” | Updates changes (still needs to be pushed to remote repository) |  |
| git push | Updates remote repository with local repository | Can use git put alias in place of this |
| git put | Branch specific push |  |
| git pull | Updates local repository with remote repository | Can use git get alias in place of this |
| git get | Branch specific pull |  |
| git checkout branchName | Moves you from one branch into another | git checkout master will return you to the main branch. Also, you shouldn’t change branches if you have uncommitted work, either commit or use stash to “delete.” This can also be used to check out specific commits by replacing the branchName with the commit number. |
| git checkout -b branchName | Create a new branch with branchName and move you to that branch |  |
| git merge branchName | Merges branchName into current branch | We don’t use this as much because we merge with pull requests online. |
| git checkout – fileName | Reverts file to previous commit during merge |  |
| git checkout –ours/--theirs fileName | Keep ours or theirs file during merge conflict |  |
| git merge –abort | Ends merge without commit |  |
| git reset –hard HEAD | Will reset your local repository to match remote repository | This is a dangerous command because it will delete any local changes. We only use it to refresh code after merged pull requests. |