**git**

**github workflow**

# github repo setup

## install git

Mac -- http://git-scm.com/downloads

Windows -- Git Bash (included with Git) for all Terminal and git interactions.

Start Menu -> Programs -> Git -> Git Bash

## create new github account and SSH key

sign up for new github account at:

https://github.com/

create an SSH key for your account

• check to see if you have an existing SSH key

click avatar image (upper right toolbar) > Settings

click SSH and GPS keys in sidebar menu (left)

if you have an existing key it will look lke this:

TomMacbookKey

Fingerprint: 97:51:28:a5:36:74:5f:16:6d:7d:cc:8a:52:e4:f7:ab

Added on Aug 27, 2017

if you have a key, move to "create github REMOTE repo" on net page

if you don't have a key...

• Open Terminal

• Paste the text below into terminal line (substituting in your GitHub email address):

(This creates a new ssh key, using the provided email as a label)

$ ssh-keygen -t rsa -b 4096 -C "your\_email@your\_host.com"

• At "Enter a file in which to save the key," press Enter (accepts the default file location):

[ENTER]

• Enter passphrase (or leave empty for no passphrase):

sample passphrase: "TomMacbookKey"

[Type a passphrase]

• Add your new SSH key to the ssh-agent:

(copy and paste code below into terinal; this uses the default macOS ssh-add command)

$ eval "$(ssh-agent -s)"

• Check your OS version (Apple Icon > About this mac: version)

if version is greater than 10.12.2, follow these steps

• navigate to user root level

$ ~ [tilde]

• open .ssh/config file with emacs

$ emacs ~/.ssh/config

• type these lines into the opened file

Host \*

AddKeysToAgent yes

UseKeychain yes

IdentityFile ~/.ssh/id\_rsa

• save the modified file and close emacs

$ CONTROL x

$ CONTROL c

• Add your SSH key to the ssh-agent and store your passphrase in the keychain

$ ssh-add -K ~/.ssh/id\_rsa

• Copy the SSH key to your clipboard

this copies the contents of the id\_rsa.pub file to your clipboard

$ pbcopy < ~/.ssh/id\_rsa.pub

• Add the SSH key to your GitHub account.

click New SSH key button

add a descriptive label for the new key into the Title text box (example: TomMacbookKey)

Paste your key into the Key field (make sure it has remained on the clipboard)

Click Add SSH key button

enter your github password (if asked) to confirm

## create github REMOTE repo

create a new repository (remote repo)

upper right corner (next to avatar) click "+"

select "New repository"

Name your repository "<repo\_name>" # this is the REMOTE repo

Write a short description # describe the project

deselect "Initialize this repository with a README" # optional (you should add a README later)

click "create repository"

default branch

new repository has one branch named master

options for linking this REMOTE repo to your LOCAL repo:

1) clone the new empty REMOTE repository from github to your LOCAL projects folder

-- OR --

2) create a new LOCAL repository on the command line, then push to REMOTE

see below for details

## 1) clone empty REMOTE repo to LOCAL then push back to REMOTE

navigae to your high-level PROJECTS folder (or exercises, workshops, homework)

create a LOCAL exact copy of the REMOTE repo (note that it is empty) via clone command

$ clone https://github.com/<github\_acct>/<repo\_name>.git

navigate to the newly created PROJECT (<repo\_name>) folder via cd command

$ cd <repo\_name>

add project files to the PROJECT folder (note: this is you LOCAL repo)

example: index.html, styles.css, script.js

push LOCAL repo to REMOTE repo via push command

git push -u origin master

## 2) create new LOCAL repo, link to/push to REMOTE

navigate via cd command to project folder (with project files: index.html, styles.css, script.js)

$ cd <repo\_name>

initialize local repository, check status, add files for staging, varify status, commit files

$ git init

$ git status

$ git add .

$ git status

$ git commit -m "initial commit"

add remote origin (new github repo) to new local repo via remote command

$ git remote add origin https://github.com/<yourRepo>.git

push local to remote repo via push command

$ git push -u origin master

## github links

https://github.com/

https://help.github.com/articles/checking-for-existing-ssh-keys/

https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/

https://help.github.com/articles/adding-a-new-ssh-key-to-your-github-account/

https://github.com/settings/keys

## read more:

https://blog.scottlowe.org/2015/01/27/using-fork-branch-git-workflow/

# git commands

## git config: checking your github account

set/check email address associated with repo via config command

$ git config --global user.email "<your\_email>" # set email

$ git config --global user.email # check current email

## git remote:

List your existing remotes

$ git remote -v # lists urls (only) for fetch and pull

$ git remote show origin # lists urls, shows branches linked for push/pull

set/check remote repo assiciated with local repo via remote command

$ git remote set-url <remote-name> <new-url>

$ git remote set-url origin https://github.com/tomBeach/CSS\_Responsive.git

## git push:

$ git push # sends all commits/branches local repo to remote repo

$ git push <remote> master # all commits in local master branch to remote master branch

git push origin master # example: pushing only master branch to remote named "origin"

$ git push <remote> --all # push all branches to remote

git push origin --all # example: pushes all branches to <remote> (origin)

$ git push <remote> <branch-name> # sends specific local branch up to remote repo

$ git push origin fixbug1 # example: push fixbug1 branch to remote origin\*

$ git push upstream fixbug1 # example: push fixbug1 branch to remote upstream\*

## git pull:

$ git pull <remote> <branch-name> # pull specific branch into local active branch

$ git pull origin newmenu # example: pull newmenu branch to local from <remote> (origin)

# note: pull combines git fetch and git merge into one step

$ git pull --rebase <remote> # use rebase to integrate remote branch with local one

# note: rebase eliminates "extraneous" commits, then merges

## git merge:

this combines/updates the code between two different branches (usually master and <other\_branch>

do this when the branch is successful/complete and can be safely combined with master

• check out the master branch

$ git checkout master # now on master branch

• run the git merge command (to integrate the feature branch)

$ git merge <branch\_name> # merges <branch\_name> into the current branch (master)

• manually repair any merge conflicts

<fix conflicts> # resolve conflicts

• run standard commit sequence

$ git status/add/commit # saves merged version to enable pushing to remote

• push merged and committed version to remote

$ git push

• Once pushed, you can delete the branch (e.g. feature\_branch\_name)

$ git branch -d <feature\_branch\_name>

\* origin is default name for your personal remote repo

upstream is default name for your group/collaborative remote repo

you can be linked to both (and more) remote repos

# managing git branches

## git branch:

• list LOCAL branches

$ git branch

• list REMOTE branches

$ git branch -r # -r = remote

• list list both remote & local branches s

$ git branch -a # -a = all

• create a new branch

$ git branch <branch-name>

## git checkout:

• switch to different branch (branch already exists)

$ git checkout <branch-name>

• create new branch and switch to it

$ git checkout -b <branch-name>

## git log:

• see all log entries

$ git log

• see all log entries without page limitation

$ git --no-pager log # enables long log list display

• log example:

$ git log

commit 61347a3be71b1573a9d4ae5dfd9ecde8bb5e83f9

Author: tomBeach <tbeach2k@gmail.com>

Date: Sat Aug 26 13:51:37 2017 -0700

expanded sidebar

• move to different commit via checkout command:

$ git checkout 61347a3be71b1573 # move to different commit (paste sample of commit id number)

# rebasing

do this *before pushing to remote repo* when working in group

never use rebasing on public branches

before you run git rebase, always ask yourself:

“Is anyone else looking at this branch?”

If the answer is yes, *take your hands off the keyboard*!

## rebase LOCAL master to REMOTE master

move to master branch

$ git checkout master # make sure you are on your master branch

show fetch and push urls to REMOTE repo

$ git remote show origin # make sure LOCAL repo is connected to a REMOTE repo

download (fetch) REMOTE branch to LOCAL repo

$ git fetch <remote>/<branch> # fetch specific REMOTE branch

$ git fetch origin/master # example: get specific branch (master) from REMOTE (origin)

$ git fetch <remote> # get the latest versions for all branches from REMOTE

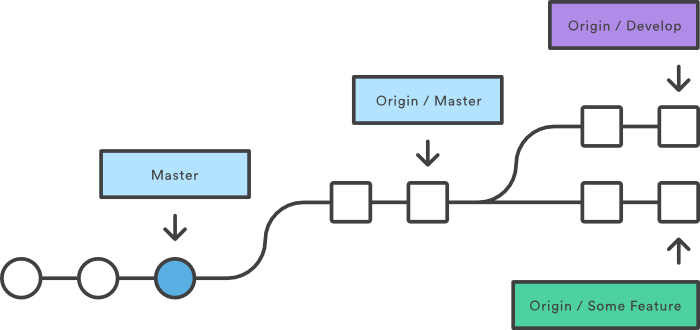
# imports commits from REMOTE repo into LOCAL repo

# commits are stored as remote branches instead of local branches

# remote branches are like local branches...

# ...but represent commits from somebody else’s repository

# example: remote branches indicated by squares (below)



## git rebase:

clean up and merge all LOCAL branches to match REMOTE

$ git rebase <remote>/<branch> # Squash commits, fix up commit messages etc.

$ git rebase origin/master # example: catch up local master branch to remote master

$ git rebase origin/fixbug1 # example: catch up local fixbug1 branch to remote master

-- OR --

$ git rebase -i <remote>/master # Squash commits interactively on master branch

# -i flag -- inspect commits one-by-one while rebasing

$ git pull -r origin master # alternative: pull and rebase to catch up local to remote master

rebase LOCAL feature to LOCAL master

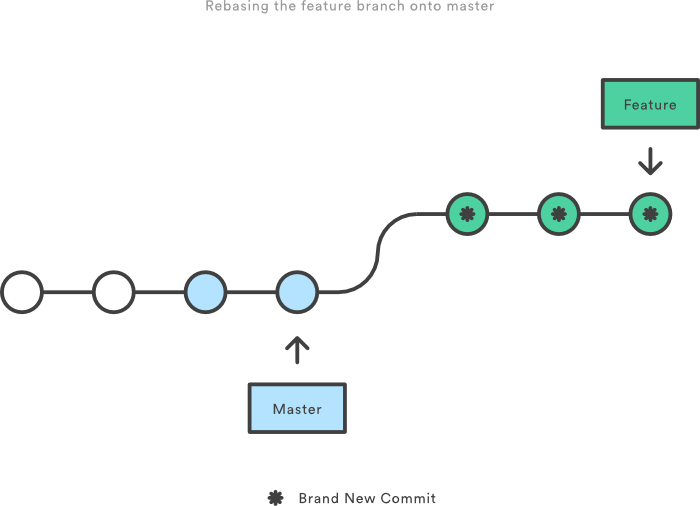
• check out a LOCAL feature branch

$ git checkout <feature\_branch\_name>

• run the git rebase command from feature branch to merge/cleanup with master branch

adds entire feature branch to tip of the master branch; deletes feature commits

$ git rebase master



rebase LOCAL feature branch to REMOTE existing branch

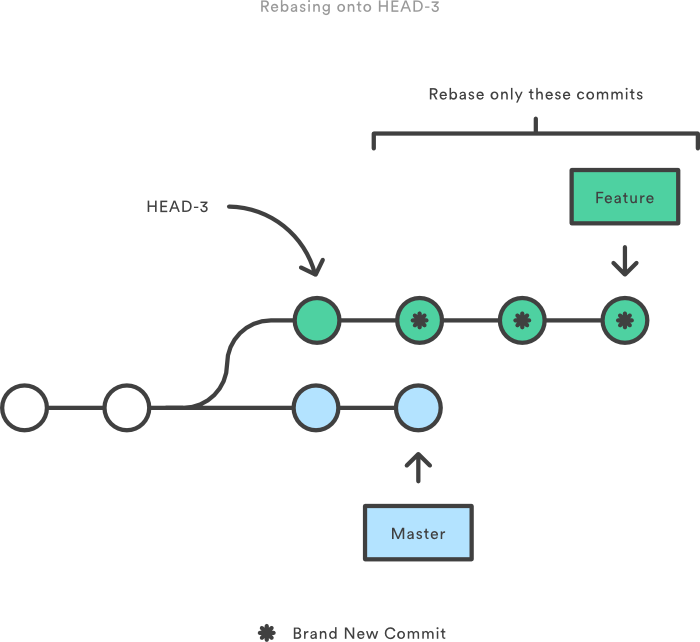
$ git fetch # get latest from remote ???

$ git rebase <remote>/<feature> # rebase our <feature> on top of the latest from remote

$ git rebase origin/fixbug2 # include remote changes to (origin) to our branch (fixbug2)

<fix conflicts> # resolve conflicts then push to remote

rebase LOCAL feature branch to earlier commit on same branch



# github collaborative workflow

diagrams source: https://www.atlassian.com/git/tutorials

set up repos for group/collaborators on github (UPSTREAM, REMOTE) and laptops (LOCAL)

github collaborative workflow steps

• **Organize** your group.

• **Fork** the group repo (on manager's personal account) to your own account.

• **Clone** the repo to your local machine.

• **Check out** a new "topic branch" and make changes.

• **Edit files** make changes to code in selected function/process/file.

organizing your group

• Discuss an idea for a new project (or new feature on an existing project)

• Assign duties and exchange info

- project manager -- holds group repo in personal account (UPSTREAM repo)

- collaborators -- contributors to project (all members)

- personal account -- your github repo account

- info exchange -- share contact info via slack, repo urls, etc.

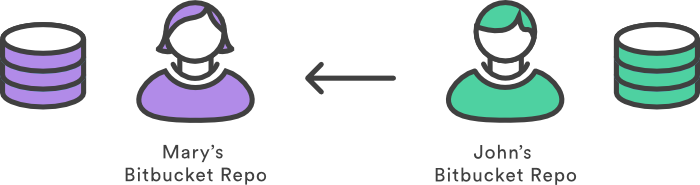
- collaborator authentication -- project manager invites members to become collaborators

• Submit project starters (can be starter files or selected existing project)

• Create an Issue -- a bug, suggestion, question, observation...

• give issue a meaningful name (e.g. "fixIssue53: expand main menu")

fork group project UPSTREAM repo to your personal REMOTE repo



• follow url to group project repo on github (this is the UPSTREAM reop)

https://github.com/WDI\_fall\_group/CSS\_Responsive.git

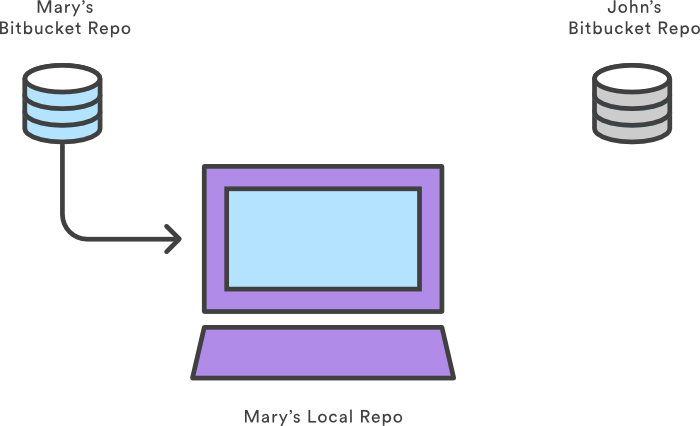
• click Fork button (right side of WDI\_fall\_group repo title-bar)

• answer alert box: Where should we put this repo?

• select repo icon for "target" repo (yours)

repo will be copied into your personal repo (REMOTE)

Create a LOCAL clone of your REMOTE repo (the one forked from UPSTREAM)



• open the forked repo in atom.

right corner of repo table

• use commnd line to make clone.

click Clone // on github account

$ git clone https://github.com/tomBeach/cs-algorithms.git // terminal

• download a zip (compressed) file.

click Download

• unpack zip file and move cloned project files to project folder.

drag -and-drop to project.

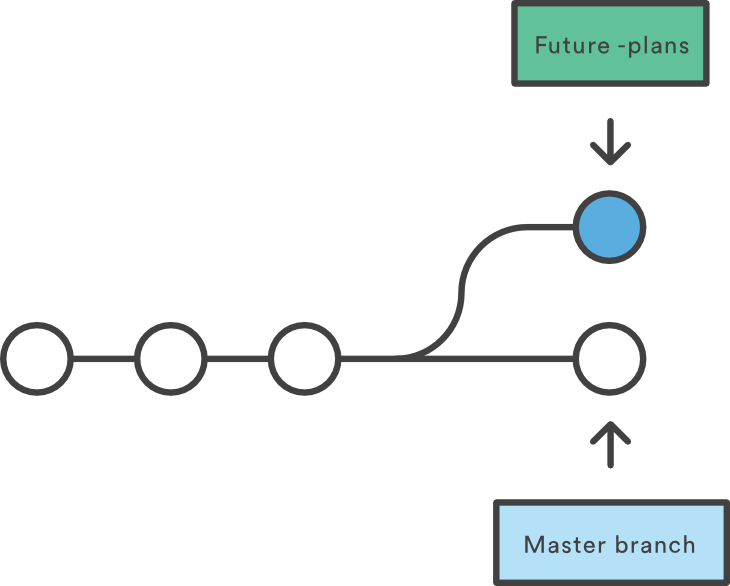
Create a new branch on LOCAL repo

use meaningful name to describe purpose of project/branch/idea

$ git checkout -b fixIssue53

fixIssue53 # Issue 53 listed in github issues collection

# Issue 53 will be automagically linked to this branch...



Make changes/commits locally

index.html, styles.css, script.js

$ git status/add/commit

Rebase LOCAL feature(s) to LOCAL master

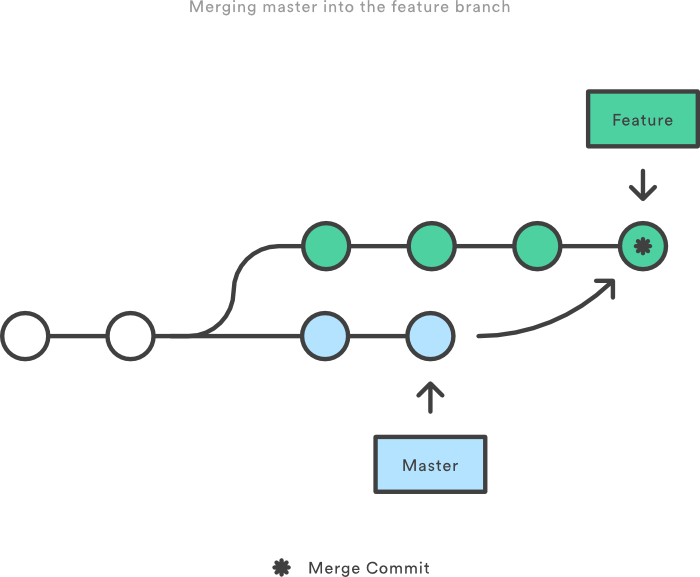
• check out LOCAL feature branch

$ git checkout <feature\_branch\_name>

• run the git rebase command from feature branch to merge/cleanup with master branch

adds entire feature branch to tip of the master branch; deletes feature commits

$ git rebase master

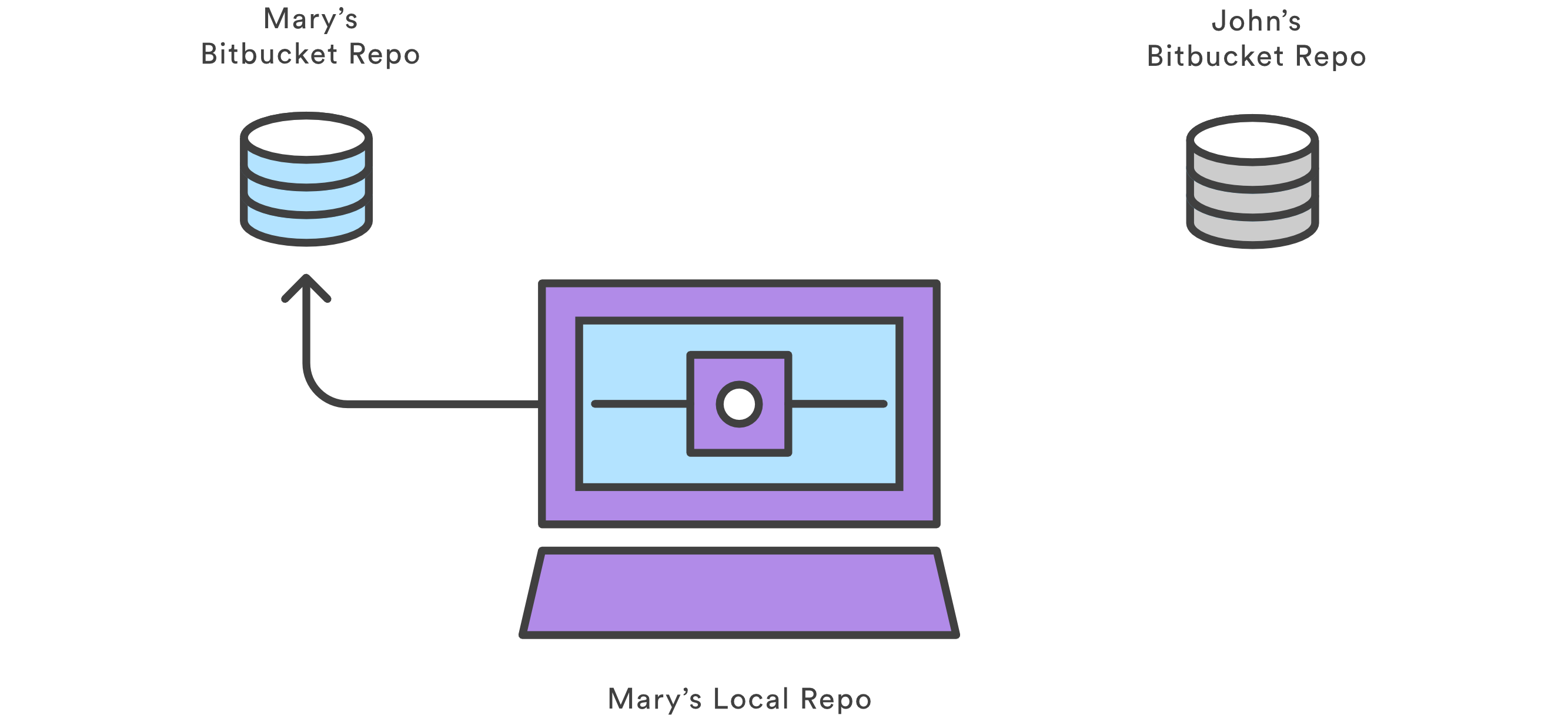


Push LOCAL repo to your REMOTE repo

$ git push origin newFeature // push newFeature (only) to your fork (origin)

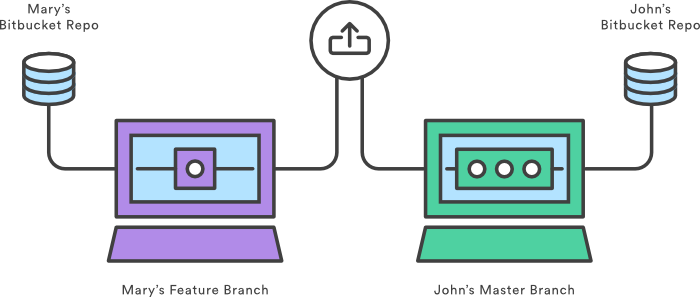
$ git push origin master // push master (only) to your fork (origin)

$ git push origin // push all branches to your fork (origin)

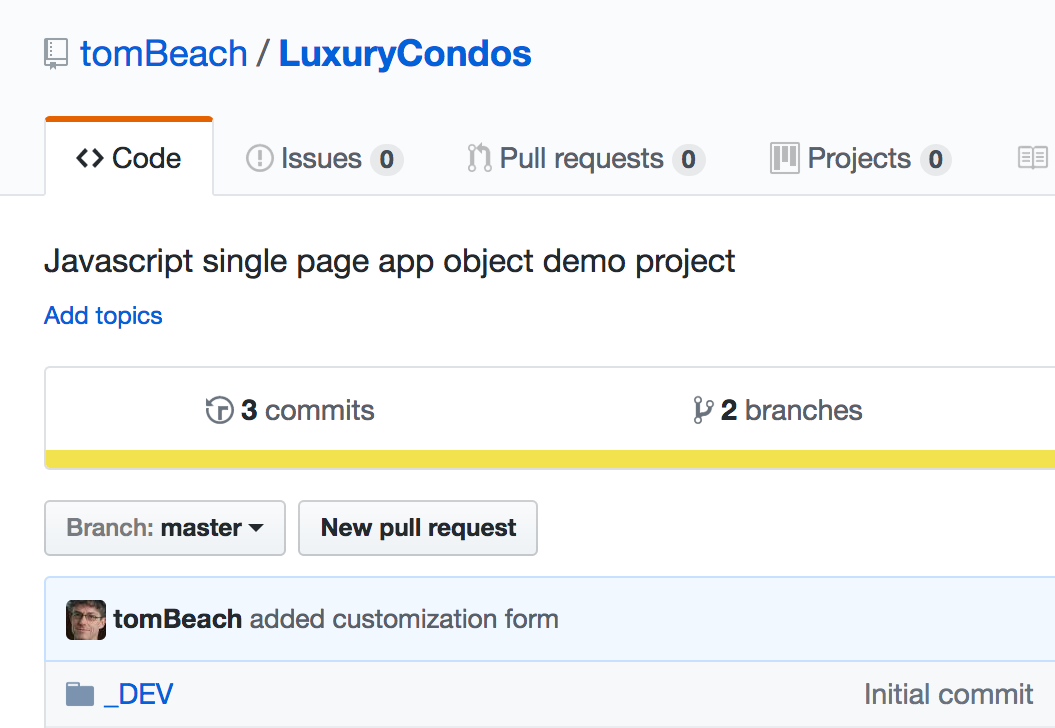


Make a Pull Request

• create a pull request REMOTE repo account



• click New Pull Request button (left side of files header)



Project manager evaluates changes on new branch via Pull Request

• select repo branches to compare (diff view with highlighted code)



• merge new branch into project master if changes are approved

-- OR --

• send comment on pull request to request different changes if merge not approved

# example: steps in code

fork group project repo (UPSTREAM) to your personal repo (REMOTE)

clone project to local computer and edit files after making new branch

$ git clone https://github.com/tomBeach/cs-algorithms.git // 1 clone your fork

Clone from REMOTE repo to LOCAL repo

$ cd cs-algorithms

$ git checkout -b newFeature // 2 create new branch

Switched to a new branch newFeature

< write code for new feature > // 3 changes to files

$ git diff --word-diff // 4 check changes

displays differences in code between old/new versions

> diff --git a/newFeature.ino b/newFeature.ino

> index 15b9911..a6cc5a5 100644

> --- a/newFeature.ino

> +++ b/newFeature.ino

> < code details >

$ git commit -a -m 'describe new feature changes' // 5 commit changes

> [newFeature 5ca509d] describe new feature changes

> 1 file changed, 2 insertions(+), 2 deletions(-)

$ git push origin newFeature // 6 push to your fork

> Username for 'https://github.com': yourUserName

> Password for 'https://tonychacon@github.com': yourPassword

> Counting objects: 5, done.

> Delta compression using up to 8 threads.

> Compressing objects: 100% (3/3), done.

> Writing objects: 100% (3/3), 340 bytes | 0 bytes/s, done.

> Total 3 (delta 1), reused 0 (delta 0)

> To https://github.com/tonychacon/blink

> \* [new branch] newFeature -> newFeature

go back to your fork on GitHub

see that GitHub noticed that we pushed a new topic branch up

click New Pull Request (green button) to open a Pull Request to the original project

-- OR --

click on pull requests in top menu

click New Pull Request

select master (left branch dropdown)

select newFeature (right branch dropdown)

shows list of commits

click file changes tab

shows red/green diffs

click Create Pull Request

add descriptive title to pull request

results of clicking Create Pull Request:

the owner of the project you forked will get a notification that someone is suggesting a change

owner gets a link to page that has all of this information on it

project owner can look at the suggested change and merge it, reject it or comment on it

project owner can leave a comment by clicking on any of the lines

anyone else watching the repository will get a notification about owner comment

make any requested changes after review

commit to the topic branch again

push new code to topic branch

automatically updates the Pull Request

catching up with changes on original source repo (UPSTREAM)

Add the original repository as a remote named “upstream”

Fetch the newest work from that remote

Merge the main branch into your topic branch

Fix the conflict that occurred

Push back up to the same topic branch

# basic git cheatsheet

basic git sequence

git config --global user.name # check active user name

git config --global user.email # check active user email

git init

git status

git add .

git status # check that all files added correctly

git commit -a -m "initial commit"

basic managing branches

git --no-pager log # see all log entries without page limitation

git branch # see all branches

git branch branch-name # create new branch

git checkout branch-name # switch to different branch

git checkout -b branch-name # create new branch and switch to it

git push origin branch-name # push branch up to GitHub

git pull origin branch-name # pull specific branch into locl active branch

git branch # list local branches

git branch -r # list remote branches

git branch -a # list both remote & local branches

basic merging branches

• check out the master branch

• run the git merge command (to integrate the feature branch)

git checkout master

git merge < branch\_name> # merges <branch\_name> into the current branch (master)

• Once merged, you can delete the branch

git branch -d <feature\_branch\_name>

# github pages

create new github pages website

log in to gitHub

create a new repository named *username*.github.io

NOTE: username is your username (or organization name) on GitHub

Clone the repository

git clone https://github.com/username/*username*.github.io

Enter the project folder and add website files

cd *username*.github.io

<< add index.html, script, style files to local directory >>

push added files to github after status/add/commit

git add --all

git commit -m "Initial commit"

git push -u origin master

go to http://*username*.github.io to see website

attach project to github pages website

[create new repository on github] # repo is empty for now

git clone github.com/tomBeach/d3samples.git # clone new repo to local

cd d3samples # cd to cloned folder

git checkout --orphan gh-pages # Creates branch without any parents

git rm -rf . # removes all files (e.g. '.gitignore')

[add/edit files] # copy dev version files to new folder

<< add index.html, script, style files to new folder >>

git status # check to see that files are recognized

git add .

git commit -a -m "First pages commit"

git push origin gh-pages

git push --set-upstream origin gh-pages

git remote show origin # check remote url for project

# heroku

deploy simple nodejs server to heroku

heroku login

git clone https://github.com/heroku/node-js-getting-started.git

cd node-js-getting-started

heroku create

git push heroku master

heroku ps:scale web=1

heroku open

heroku logs --tail

use Heroku to deploy app and manage its code in GitHub

• new personal GitHub repo named NodeWebGL

• define a new remote location named git (or other name) for my own GitHub repo

• nameless-harbor-7576.herokuapp.com is heroku acct

$ git remote -v

git https://github.com/jeremytammik/nodewebgl01.git (fetch)

git https://github.com/jeremytammik/nodewebgl01.git (push)

heroku https://git.heroku.com/nameless-harbor-7576.git (fetch)

heroku https://git.heroku.com/nameless-harbor-7576.git (push)

origin https://github.com/heroku/node-js-getting-started.git (fetch)

origin git@github.com:jeremytammik/nodewebgl01.git (push)

$ git remote remove git

$ git remote remove origin

$ git remote -v

heroku https://git.heroku.com/nameless-harbor-7576.git (fetch)

heroku https://git.heroku.com/nameless-harbor-7576.git (push)

$ git remote add git https://github.com/jeremytammik/NodeWebGL.git

$ git remote -v

git https://github.com/jeremytammik/NodeWebGL.git (fetch)

git https://github.com/jeremytammik/NodeWebGL.git (push)

heroku https://git.heroku.com/nameless-harbor-7576.git (fetch)

heroku https://git.heroku.com/nameless-harbor-7576.git (push)

$ git push git master

Counting objects: 394, done.

Delta compression using up to 8 threads.

Compressing objects: 100% (311/311), done.

Writing objects: 100% (394/394), 252.43 KiB | 0 bytes/s, done.

Total 394 (delta 56), reused 381 (delta 52)

To https://github.com/jeremytammik/NodeWebGL.git

\* [new branch] master -> master

to update push files to the two repositories like this:

git add app.json

git commit -m "updated name and description"

git push git master

git push heroku master