now you `git` it! (day ii of ii)

wintersession winter 2022, january 14 @ 10:00

by dev dabke (ddabke@princeton.edu)

me (@DbCrWk, ddabke)

- he/him/his
- education
  - applied math phd @ Princeton
  - math/cs undergrad @ Duke
- industry
  - NASA, BlackRock, Airtable
- started with svn, then realized how git works and why it’s awesome

0. recap

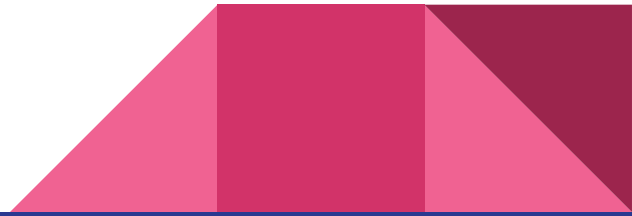
COMMENT	DATE
CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
ENABLED CONFIG FILE PARSING	9 HOURS AGO
MISC BUGFIXES	5 HOURS AGO
CODE ADDITIONS/EDITS	4 HOURS AGO
MORE CODE	4 HOURS AGO
HERE HAVE CODE	4 HOURS AGO
AAAAAAAA	3 HOURS AGO
ADKFJSLKDFJSDKLFJ	3 HOURS AGO
MY HANDS ARE TYPING WORDS	2 HOURS AGO
HAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

citation: xkcd and https://github.com/qw3rtman/git-fire

workshop structure

1. overall style
   a. feel free to jump in with questions at any time (don't be afraid!)
   b. i will poll the audience a lot
2. start with basic concepts and theory; then practical exercises
   a. discuss basic ideas; play with a few
   b. learn git for working with ourselves and other people
3. high-level
   a. day 1: basics, toy examples, theory
   b. **day 2**: exercises; personal and team workflows
4. works cited: https://git-scm.com/book/en/v2



centralized vs. decentralized

centralized

- one “master” repo
- central server or machine hosts the “canonical code”
- checkout / checkin to one place
- checkout only parts of a repo
- easy to manage permissions

decentralized

- everyone has the **entire** repo (yes, submodules exist in git, etc., but c’mon)
- checkout / checkin from anywhere
- “canonical code” is established by convention (`master` or `origin` aren’t special names in git)

immutable vs. mutable

immutable

- cannot rewrite the “history” or “record” of what has happened
- history reflects what actually happened
- safer

mutable

- possible to change history
- history can be beautified, changed to be easier to read for the future
- more flexible
- can be more dangerous

isn’t github a centralized system, though? wait . . . what is github?

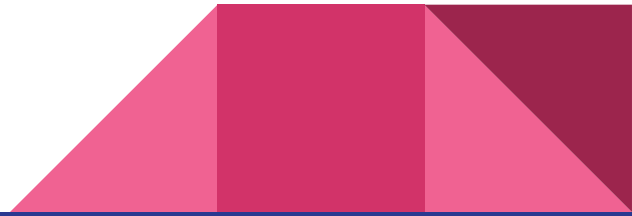
what is a commit?

what is a branch?

how do we reconcile differences between commits?

recap

1. git is decentralized, mutable
2. github (or equivalent) is a hosting service
3. everything in git is an object, which has
    a. content
    b. pointer
4. branches are labels



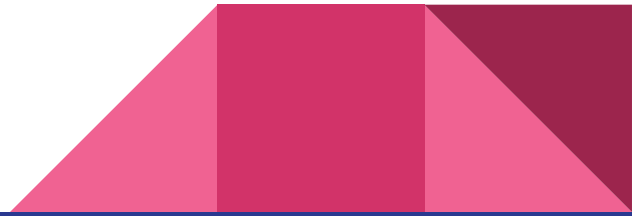
1. introduction

repository

1. a **folder** that has been **initialized** by git
2. contains
    a. a working directory, i.e. what your filesystem sees
    b. a .git folder that maintains the version control information

remote

1. location (or address) of a git repository
2. can be on
    a. your computer
    b. a local network
    c. over the internet with
        i. ssh
        ii. http(s)
3. composed of
    a. a url
    b. a name

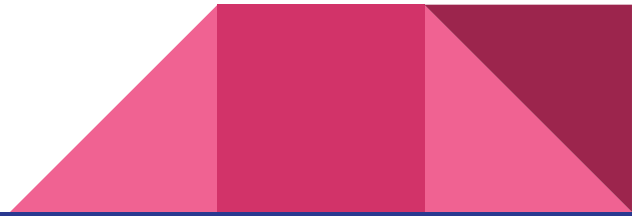


ref

1. unique way to identify a resource
2. composed of a
    a. remote
    b. object identifier

ref: HEAD

1. what current working directory is pointing at
2. what is actually being changed upon `checkout`



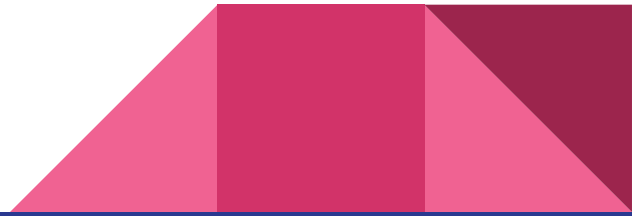
2. individual work

preamble

1. setup a github account
   a. associate and verify a personal email address
   b. turn on two-factor
   c. associate and verify your princeton email if you have one; activate education account
2. ssh keys
   a. if you don't have them: https://help.github.com/en/github/authenticating-to-github/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent
   b. once you've created them or if you already have them: https://help.github.com/en/github/authenticating-to-github/adding-a-new-ssh-key-to-your-github-account

repository creation

1. create a new (public) repository on github
2. initialize it with a readme file



`git clone`

setup a local replica repository from a remote

pop quiz: what is special about `origin`?

`git fetch`

check for changes in your local repository against a remote

`git push`

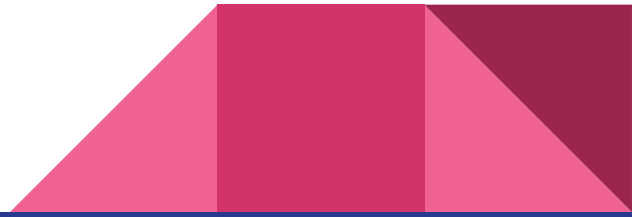
push a commit to a remote

`git pull`

performs a fetch and merge at the same time (**avoid this**)

local first, then remote

1. create a local repository
2. then, create one on github
    a. decline all automated configurations
3. push up your local repository



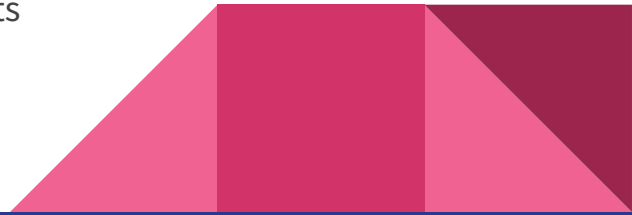
3. workflows

preamble

1. for each exercise
    a. create a repository with **no default initialization**

general working principles

1. branches are cheap
   a. create personal working branches
   b. even for collaborative work, create focused “feature” branches
   c. be fearless: branch off of branches
2. keep `master` clean (so there is always one working version)
3. stay in-sync with `master`
   a. if ready, `merge`
   b. otherwise `rebase`
4. first commit to `master` wins
   a. commit merging into master is responsible for reconciling conflicts
5. `fetch` before you `push`

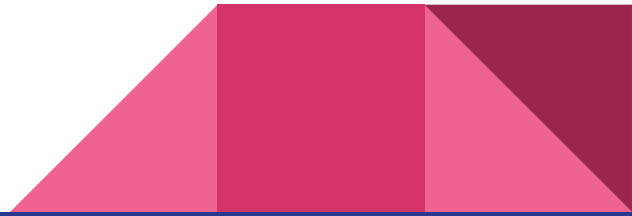


ex. 1

1. create a README with your name

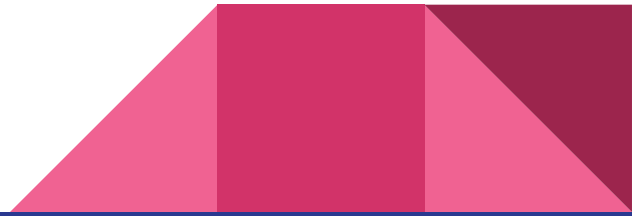
ex. 2

1. create a README with your name
2. create your own file
    a. use your own feature branch
    b. merge into master



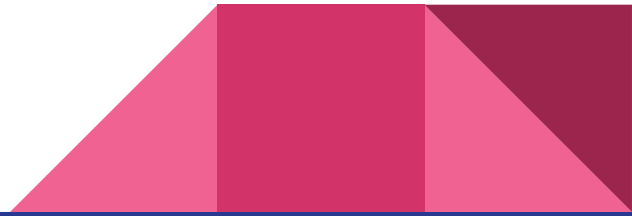
ex. 3

1. create a README with your name
2. create your own file
3. initiate a pull request
    a. push up your feature branch
    b. on github, click on **pull request (pr)** to master
    c. once accepted, merge in through the UI



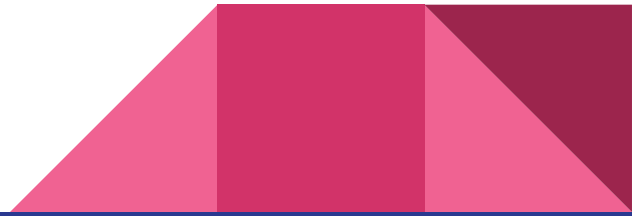
fork

1. github concept
2. replica of a repository but not integrated via git
3. a way to handle permissions, since anyone with access to a repo can access the whole thing



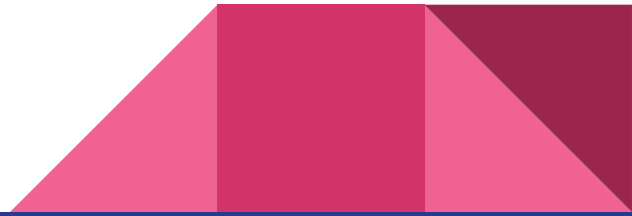
ex. 4: massive fork experiment

1. fork my repository
   a. https://github.com/DbCrWk/pu-bootcamp-2019-ex-fork
2. submit a pr by editing the README
3. merge in changes



wrap up

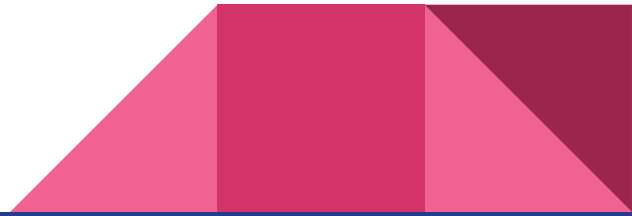
1. git is **not**
    a. a substitute for project management
    b. a tool for communication
2. many ways to use git
3. be generous to your collaborators, strict with yourself



4. advanced concepts

project setup

1. gitignore
2. README
3. permissions

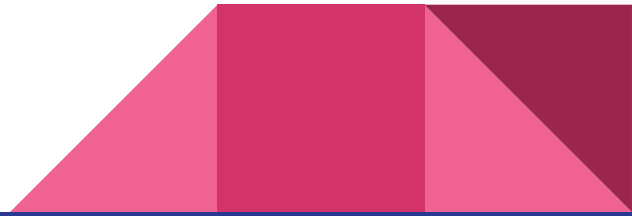


hooks

1. scripts that active on events

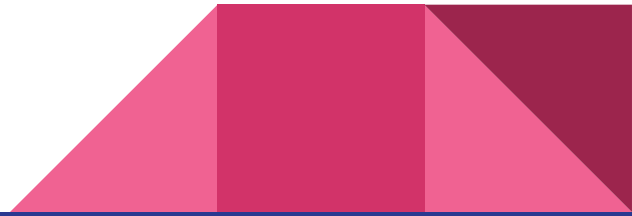
integrations

1. ci/cd
2. communication
3. project management



ci/cd

1. “continuous integration, continuous delivery”
    a. ambitious
    b. broad
2. circleci
    a. integrate checks to github
    b. “verifies” code
3. others
    a. style
    b. test coverage
    c. more?

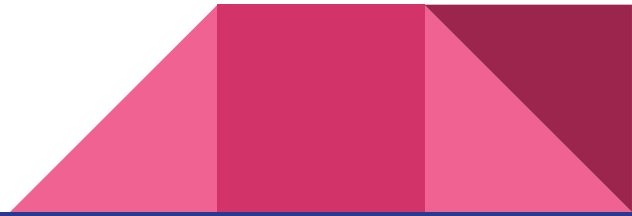


communication

1. slack
2. more?

project management

1. github itself
2. more?



references

https://git-scm.com/book/en/v2/

now you `git` it!