

git

an introduction & explanation by

Nikit “Nefari0uss” Malkan



disclaimer

disclaimer

this presentation is very long

disclaimer

this presentation is very long
essentially condensed a book into one presentation

disclaimer

this presentation is very long

essentially condensed a book into one presentation

git can be very confusing

disclaimer

this presentation is very long
essentially condensed a book into one presentation
git can be very confusing
ask questions

Table of Contents

- ❖ Installing Git
- ❖ What is it?
 - ❖ Source control analogy
- ❖ Why use source control?
 - ❖ Specifically Git?
 - ❖ Terminal vs GUI interface
- ❖ Git internals
- ❖ Set up Git, GitHub, BitBucket
 - ❖ Add ssh keys, make repos
- ❖ Put your knowledge to work!
 - ❖ Mix with Commands
 - ❖ Demonstration w/audience

installing git

installing git

<https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

Windows/Mac: run installer

Linux: use your package manager

installing git

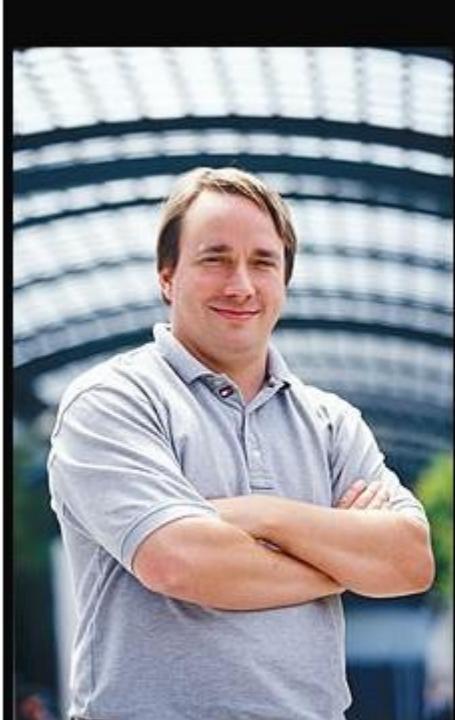
use git from Windows Command Prompt
commit Windows style, commit Unix-style

what is a git

what is a git

git: [informal] [British] an unpleasant or contemptible person.

what is a git



I'm an egotistical bastard, and I name all my projects after myself. First Linux, now git.

(Linus Torvalds)

what is git

git: A distributed revision control system with...support for distributed, non-linear workflows. [Wikipedia – Git (software)]

what is git

...

what is git

...

Try this: a version control system that allows a user to store, update, and maintain files and projects.

why use version control

let's talk workflow

let's talk workflow

imagine writing an essay

let's talk workflow

imagine writing an essay

what's the process like

the writing process

the writing process

write the whole thing at once

the writing process

write the whole thing at once

CLG are NALCS champs. TSM is trash.

the writing process

write the whole thing at once

unlikely

the writing process

write the whole thing at once

unlikely

write bits and pieces in multiple settings

the writing process

write

CLG are NALCS champs.

the writing process

write → save

CLG are NALCS champs.

the writing process

write → save → s5secretsgg.docx

CLG are NALCS champs.

the writing process

write → save → s5secretsgg.docx

procrastinate

CLG are NALCS champs.

the writing process

write → save → write

CLG are NALCS champs. TSM is trash.

the writing process

write → save → write → save

CLG are NALCS champs. TSM is trash.

the writing process

write → save → write → save → s5secretsgg.docx

CLG are NALCS champs. TSM is trash.

the writing process

write →

:g.docx



the writing process

write → save → write → save → s5secretsgg.docx
rip progress

CLG are NALCS champs. TSM is trash.

the writing process (trial 2)

write → save → s5secretsgg.docx

CLG are NALCS champs. TSM is trash.

the writing process (trial 2)

write → save → s5secretsgg.docx

save backup → s5secretsgg_backup.docx

CLG are NALCS champs. TSM is trash.

the writing process (trial 2)

write → save → s5secretsgg.docx

save backup

s5secretsgg.docx

CLG are NALCS champs. TSM is trash.

s5secretsgg_backup.docx

CLG are NALCS champs. TSM is trash.

the writing process (trial 2)

s5secretsgg.docx

CLG are NALCS champs. TSM is trash.

s5secretsgg_backup.docx

CLG are NALCS champs. TSM is trash.

the writing process (trial 2)

s5secretsgg.docx

CLG are NALCS champs. TSM is trash.

s5secretsgg_backup.docx

CLG are NALCS champs. TSM is trash.

what if we mess up our active copy

the writing process (trial 2)

s5secretsgg.docx

TSM are NALCS champs. No thread is safe.

s5secretsgg_backup.docx

CLG are NALCS champs. TSM is trash.

what if we mess up our active copy

the writing process (trial 2)

s5secretsgg.docx

s5secretsgg_backup.docx

TSM are NALCS champs. No thread is safe.

CLG are NALCS champs. TSM is trash.

what if we mess up our active copy
restore from backup

the writing process (trial 2)

s5secretsgg.docx

s5secretsgg_backup.docx

TSM are NALCS champs. No thread is safe.

CLG are NALCS champs. TSM is trash.

what if we mess up our active copy
restore from backup

the writing process (trial 2)

s5secretsgg.docx

CLG are NALCS champs. TSM is trash.

s5secretsgg_backup.docx

CLG are NALCS champs. TSM is trash.

what if we mess up our active copy
restore from backup

the writing process (trial 2)

s5secretsgg.docx

CLG are NALCS champs. TSM is trash.

s5secretsgg_backup.docx

CLG are NALCS champs. TSM is trash.



the writing process (trial 2)

s5secretsgg.docx

s5secretsgg_backup.docx

what if we mess up active copy and overwrite the backup

the writing process (trial 2)

s5secretsgg.docx

→ save →

s5secretsgg_backup.docx

TSM are NALCS champs. No thread is safe.

→ TSM are NALCS champs. No thread is safe.

the writing process (trial 2)

s5secretsgg.docx

s5secretsgg_backup.docx

uh-oh

the writing process (trial 2)

s5secretsgg.docx s5secretsgg_backup.docx

uh-oh

we overwrote our backup with the mess up

the writing process (trial 3)

write → save → s5secretsgg.docx

CLG are NALCS champs. TSM is trash.

the writing process (trial 3)

write → save → s5secretsgg.docx

make iterative backups

CLG are NALCS champs. TSM is trash.

the writing process (trial 3)

s5secretsgg

the writing process (trial 3)

s5secretsgg

s5secretsgg_2015-09-01

the writing process (trial 3)

s5secretsgg

s5secretsgg_2015-09-01

s5secretsgg_2015-09-10

the writing process (trial 3)

s5secretsgg

s5secretsgg_2015-09-01

s5secretsgg_2015-09-10

s5secretsgg_2015-09-19

the writing process (trial 3)

s5secretsgg

s5secretsgg_2015-09-01

s5secretsgg_2015-09-10

s5secretsgg_2015-09-19 s5secretsgg_2015-09-28

the writing process (trial 3)

s5secretsgg

s5secretsgg_2015-09-01

s5secretsgg_2015-09-10

s5secretsgg_2015-09-19

s5secretsgg_2015-09-28

this gets messy very quickly

the writing process (trial 3)

s5secretsgg

s5secretsgg_2015-09-01

s5secretsgg_2015-09-10

s5secretsgg_2015-09-19 s5secretsgg_2015-09-28

this gets messy very quickly

however, this system allows us to restore from any given point

new problem

new problem

which backup has the info we need

new problem

which backup has the info we need

s5secretsgg

s5secretsgg_2015-09-01

s5secretsgg_2015-09-10

s5secretsgg_2015-09-19

s5secretsgg_2015-09-28

ponder that question for a bit...

let's shift gears for a minute...

ponder that question for a bit...

let's shift gears for a minute...

let's talk *git*

why use git

why use git

you have all these iterative backups

why use git

you have all these iterative backups

how do you manage them all

why use git

git pros

git cons

git pro

helps you manage version control

git pro

helps you manage version control
distributed (explained later)

git pro

helps you manage version control
distributed (explained later)

fast

git pro

helps you manage version control

distributed (explained later)

fast

used nearly everywhere in software world

git pro

helps you manage version control

distributed (explained later)

fast

used nearly everywhere in software world

can be used to non-software projects (like this)

git pro

helps you manage version control

distributed (explained later)

fast

used nearly everywhere in software world

can be used to non-software projects (like this)

it's not SVN

git pro

helps you manage version control

distributed (explained later)

fast

used nearly everywhere in software world

can be used to non-software projects (like this)

it's not SVN (centralized)

(the fact that my opinion happens to be the right opinion is a happy coincidence)

git con

considered very hard to learn

git con

considered very hard to learn

lots of weird syntax

git con

considered very hard to learn

lots of weird syntax

understanding git internals helps remedy this tremendously

terminal vs GUI

terminal vs GUI

terminal way is best way

terminal vs GUI

terminal way is best way

but nefari0uss – that gui application looks so nice and pretty

terminal vs GUI

terminal way is best way

but nefari0uss – that gui application looks so nice and pretty

and that terminal looks scary and hard

terminal vs GUI

terminal way is best way

but nefari0uss – that gui application looks so nice and pretty

and that terminal looks scary and hard

I don't care

gitg - linux-git

File Edit View Help

History Commit

Branch: Select branch

Subject Date

Subject	Author	Date
Merge branch 'gb/gitweb-opml'	Junio C Hamano	Sun Jan 18 08:07:19 2009
Merge branch 'mv/apply-parse-opt'	Junio C Hamano	Sun Jan 18 08:06:53 2009
Merge branch 'tr/rebase-root'	Junio C Hamano	Sun Jan 18 08:06:38 2009
Merge branch 'gb/gitweb-patch'	Junio C Hamano	Sun Jan 18 08:06:19 2009
Merge branch 'ap/clone-into-empty'	Junio C Hamano	Sun Jan 18 08:05:54 2009
Merge branch 'jc/maint-format-patch'	Junio C Hamano	Sun Jan 18 08:05:50 2009
Merge branch 'tr/maint-no-index-fixes'	Junio C Hamano	Sun Jan 18 08:05:38 2009
Merge branch 'as/autocorrect-alias'	Junio C Hamano	Sun Jan 18 08:05:34 2009
Merge branch 'rs/fgrep'	Junio C Hamano	Sun Jan 18 08:05:28 2009
Merge branch 'rs/maint-shortlog-foldline'	Junio C Hamano	Sun Jan 18 08:05:23 2009
Merge branch 'mh/maint-commit-color-status'	Junio C Hamano	Sun Jan 18 08:05:19 2009
Merge branch 'maint'	Junio C Hamano	Sun Jan 18 08:04:40 2009
Update draft release notes for 1.6.1.1	Junio C Hamano	Sun Jan 18 08:04:35 2009
bundle: allow the same ref to be given more than once	Junio C Hamano	Sun Jan 18 07:27:08 2009
Merge branch 'maint-1.6.0' into maint	Junio C Hamano	Sun Jan 18 07:39:49 2009
builtin-fsck: fix off by one head count	Christian Couder	Sun Jan 18 04:46:09 2009
revision walker: include a detached HEAD in --all	Johannes Schindelin	Fri Jan 16 13:52:53 2009
Fix aitdir_detection when in subdir of aitdir	SZEDER Gábor	Fri Jan 16 16:37:33 2009

Details Tree

SHA: b2a6d1c6868b6d5e7d2b4fa9129341220ale848a
 Author: Junio C Hamano
 Date: Sun Jan 18 07:27:08 2009
 Subject: **bundle: allow the same ref to be given more than once**
 Parent: [f0298cf1c6a7b5cc8b79d84a03b0ce07df2d9e6b](#) (revision walker: include a detached HEAD in --all)

```

bundle: allow the same ref to be given more than once

"git bundle create x master master" used to create a bundle that lists
the same branch (master) twice. Cloning from such a bundle resulted in
a needless warning "warning: Duplicated ref: refs/remotes/origin/master".

Signed-off-by: Junio C Hamano <gitster@pobox.com>

diff --git a/bundle.c b/bundle.c
index daecd8e..b20f210 100644
--- a/bundle.c
+++ b/bundle.c
@@ -240,6 +240,8 @@ int create_bundle(struct bundle_header *header, const char *path,
240         return error("unrecognized argument: %s'", argv[i]);
241     }
242 
243     object_array_remove_duplicates(&revs.pending);
244+
245     for (i = 0; i < revs.pending.nr; i++) {
246         struct object_array_entry *e = revs.pending.objects + i;
247         unsigned char_shall[20];
diff --git a/object.c b/object.c
index 50b6528..7e6a92c 100644
--- a/object.c
+++ b/object.c

```

Loaded 17336 revisions in 0.75s

Terminal

```
2012-11-23 10:51 James Winters      M [master] [origin/HEAD] [origin/master] Merge branch 'atlassian_jira_6_0_0~
2012-11-23 10:49 James Winters      M Merge commit 'a6710b1b1148a3f66816df1ddde193d6a6d7d6f2' from 6.0_OD1 rele~
2012-11-23 10:42 James Winters      o [origin/atlassian_jira_6_0_OD_01_release_branch] ROTP-609: Now store th~
2012-11-23 10:25 Geoffrey Wong     o Update Baseline Images - changes from ROTP-614: show custom content lin~
2012-11-23 09:57 Jeroen De Raedt   M [origin/JRADEV-15983] Merge branch 'master' into JRADEV-15983
2012-11-22 16:35 Filip Rogaczewski o Revert merged from stable causing functional test break.
2012-11-22 18:02 Eric Dalglish    M Merged from stable.
2012-11-22 16:28 Martin Meinholt  o ROTP-578 Fix WebDriver test when common header is on
2012-11-22 16:28 Graeme Smith     o Fixed merge weirdness. Somehow this got removed.
2012-11-22 16:23 Graeme Smith     o Revert "Revert "Merge branch 'JRADEV-15983'"""
[main] 647478e4b89612425641d3102b6feadeb5137fc8 - commit 8 of 47622 (0%)
commit 647478e4b89612425641d3102b6feadeb5137fc8
Refs: atlassian_jira_5_2-773-g647478e
Author: Martin Meinholt <mmeinholt@atlassian.com>
AuthorDate: Thu Nov 22 16:28:23 2012 +1100
Commit: Martin Meinholt <mmeinholt@atlassian.com>
CommitDate: Thu Nov 22 16:28:23 2012 +1100

ROTP-578 Fix WebDriver test when common header is on

* Add common header specific issues menu
* Fix wrong selector in JiraCommonHeader page object
---
.../jira/pageobjects/components/JiraCommonHeader.java |  2 +-+
.../jira/pageobjects/components/JiraHeader.java       | 17 ++++++-----+
.../components/menu/CommonHeaderIssuesMenu.java     | 14 ++++++-----+
.../jira/pageobjects/components/menu/IssuesMenu.java |  5 +++++
4 files changed, 34 insertions(+), 4 deletions(-)

diff --git a/jira-page-objects/src/main/java/com/atlassian/jira/pageobjects/components/JiraCommonHeader.java b/ji~
index 6b551d6..8f3c3b0 100644
--- a/jira-page-objects/src/main/java/com/atlassian/jira/pageobjects/components/JiraCommonHeader.java
[diff] 647478e4b89612425641d3102b6feadeb5137fc8 - line 1 of 119 (17%)
```

terminal vs GUI

gui applications hide what's happening

terminal vs GUI

gui applications hide what's happening

terminal way forces you to learn what's happening

terminal vs GUI

gui applications hide what's happening

terminal way forces you to learn what's happening

Murphy's law *will* happen

terminal vs GUI

gui applications hide what's happening

terminal way forces you to learn what's happening

Murphy's law *will* happen

your code base will become borked

terminal vs GUI

gui applications hide what's happening

terminal way forces you to learn what's happening

Murphy's law *will* happen

your code base will become borked

knowing how git works will help you fix it

THIS IS GIT. IT TRACKS COLLABORATIVE WORK
ON PROJECTS THROUGH A BEAUTIFUL
DISTRIBUTED GRAPH THEORY TREE MODEL.

COOL. HOW DO WE USE IT?

NO IDEA. JUST MEMORIZIZE THESE SHELL
COMMANDS AND TYPE THEM TO SYNC UP.
IF YOU GET ERRORS, SAVE YOUR WORK
ELSEWHERE, DELETE THE PROJECT,
AND DOWNLOAD A FRESH COPY.



<https://xkcd.com/1597/>

terminal vs GUI

gui may not always be available

terminal vs GUI

gui may not always be available

terminal will

terminal vs GUI

gui may not always be available

terminal will

↙ ↗ ↘ ↙ *terminal way is best way* ↙ ↗ ↘ ↙

going back...

going back...

remember this?

new problem

which backup has the info we need

s5secretsgg

s5secretsgg_2015-09-01

s5secretsgg_2015-09-10

s5secretsgg_2015-09-19

s5secretsgg_2015-09-28

new problem

which backup has the info we need

answer is to look at the *metadata*

git internals

git internals

this section is **very** important to understand

git internals

this section is **very** important to understand
seriously – pay attention to this section

metadata

metadata

a set of data that describes and gives information about other data

metadata

a set of data that describes and gives information about other data

backups should contain metadata

metadata

who

what

when

metadata

who: author of the change

what:

when:

metadata

who: author of the change

what: what was changed

when:

metadata

who: author of the change

what: what was changed

when: date and time of change

metadata

who: author of the change

what: what was changed

when: date and time of change

where are we going with this

metadata

who: author of the change

what: what was changed

when: date and time of change

where are we going with this

commit: record change to the repository (workspace)

repository

repository

the workspace containing all files of a project

git repository

git repository

git stores information about the project in the repo.

git repository

git stores information about the project in the repo.

commit objects

references to commit objects (heads)

git repository

git stores information about the project in the repo.

commit objects

references to commit objects (heads)

info stored inside project root directory in .git folder

commits

commits

before we talk about commit objects, what's a commit

commits

before we talk about commit objects, what's a commit
record change to the repository (workspace)

commits

before we talk about commit objects, what's a commit
record change to the repository (workspace)

more specifically, it's a snapshot of the repo

commits
imagine a game

commits

imagine a game

you have multiple options (branches) you can take

commits

imagine a game

you have multiple options (branches) you can take

you save the game every time you have a branch (you make a commit)

commits

imagine a game

you have multiple options (branches) you can take

you save the game every time you have a branch (you make a commit)

every new save (commit) is a record of all the actions you've taken

commits

imagine a game

you have multiple options (branches) you can take

you save the game every time you have a branch (you make a commit)

every new save (commit) is a record of all the actions you've taken

sometimes you make a bad decision and don't realize it till later

commits

imagine a game

you have multiple options (branches) you can take

you save the game every time you have a branch (you make a commit)

every new save (commit) is a record of all the actions you've taken

sometimes you make a bad decision and don't realize it till later

you want to see where you made a bad decision

commits
remember metadata

commits

remember metadata

each save lets you know some information about the decision you made

commits

remember metadata

each save lets you know some information about the decision you made

you can go back to a commit and start from there

commits

remember metadata

each save lets you know some information about the decision you made
you can go back to a commit and start from there

later on: you can take all the good save results and remove the bad
(merging/rebasing/cherry picking)

commit objects

commit objects

keep in mind the game save analogy

commit objects

3 things

commit objects

3 things

the files in the repo (character data)

commit objects

3 things

the files in the repo (character data)

parent commits (previous save files)

commit objects

3 things

the files in the repo (character data)

parent commits (previous save files)

SHA-1 hash (unique identifier)

hashing

hashing

a quick explanation

hashing

mapping data of arbitrary size to data of fixed size

hashing

mapping data of arbitrary size to data of fixed size
without going too much into it...

hashing

mapping data of arbitrary size to data of fixed size

without going too much into it...

data → hash function → data mapped to a value (hash)

hashing

mapping data of arbitrary size to data of fixed size
without going too much into it...

data → hash function → data mapped to a value (hash)
example hash function: $\text{hash} = \text{sum}(\text{string.length} / 7)$

keys

**hash
function**

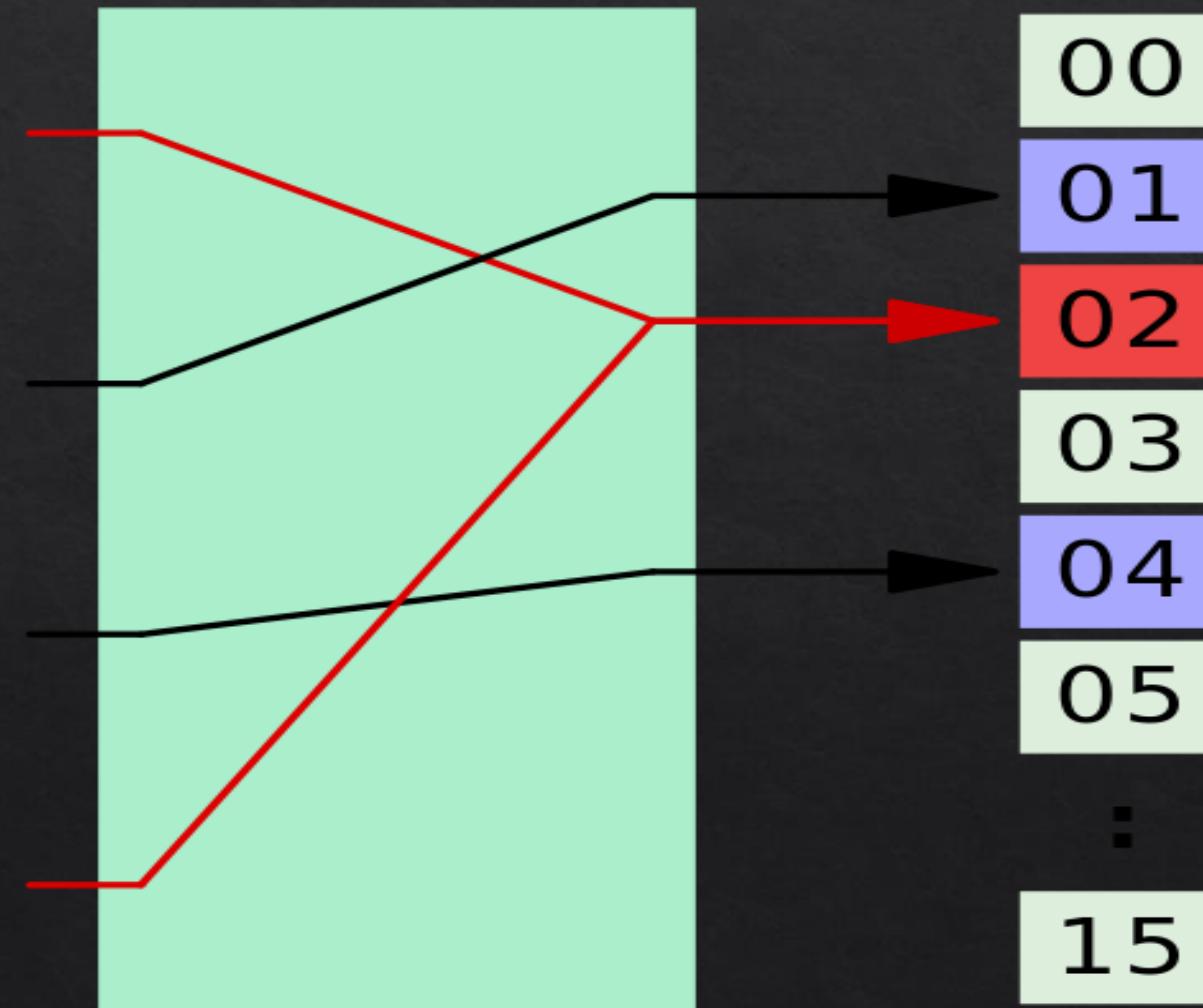
hashes

John Smith

Lisa Smith

Sam Doe

Sandra Dee



hashing

notice how we have two red lines going to the same thing

hashing

notice how we have two red lines going to the same thing

this is called a collision

hashing

notice how we have two red lines going to the same thing

this is called a collision

in security collision is bad

hashing

notice how we have two red lines going to the same thing

this is called a collision

in security collision is bad

imagine if your security key is shares the same hash as someone else's

cryptographic hashing

cryptographic hashing

cryptographic hashing allows you to verify input to a known hash value

cryptographic hashing

cryptographic hashing allows you to verify input to a known hash value
very difficult to guess/reconstruct without input data

Input

Fox

cryptographic
hash
function

DFCD 3454 BBEA 788A 751A
696C 24D9 7009 CA99 2D17

The red fox
jumps over
the blue dog

cryptographic
hash
function

0086 46BB FB7D CBE2 823C
ACC7 6CD1 90B1 EE6E 3ABC

The red fox
jumps ouer
the blue dog

cryptographic
hash
function

8FD8 7558 7851 4F32 D1C6
76B1 79A9 0DA4 AEFE 4819

The red fox
jumps oevr
the blue dog

cryptographic
hash
function

FCD3 7FDB 5AF2 C6FF 915F
D401 C0A9 7D9A 46AF FB45

The red fox
jumps oer
the blue dog

cryptographic
hash
function

8ACA D682 D588 4C75 4BF4
1799 7D88 BCF8 92B9 6A6C

Digest

SHA-1

SHA-1

type of cryptographic hashing

SHA-1

type of cryptographic hashing

has two uses in git

SHA-1

type of cryptographic hashing

has two uses in git

verify that two files are the same

SHA-1

type of cryptographic hashing

has two uses in git

verify that two files are the same

verify that a file hasn't been altered

SHA-1

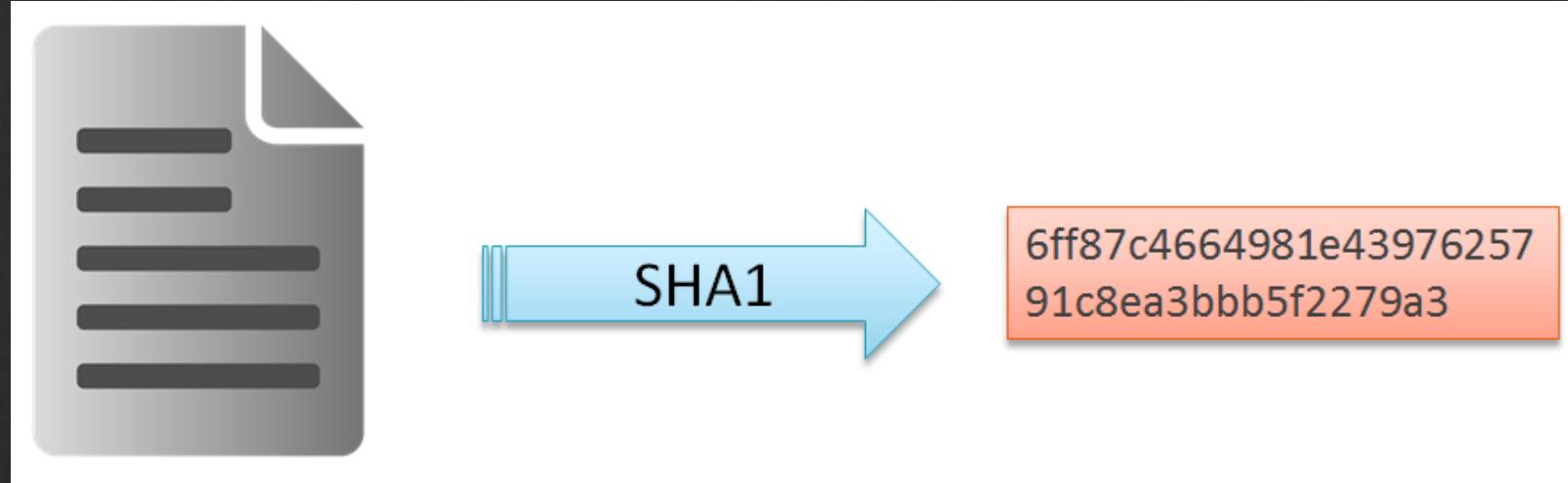
type of cryptographic hashing

has two uses in git

verify that two files are the same

verify that a file hasn't been altered

40-character string



SHA-1

type of cryptographic hashing

has two uses in git

verify that two files are the same

verify that a file hasn't been altered

40-character string

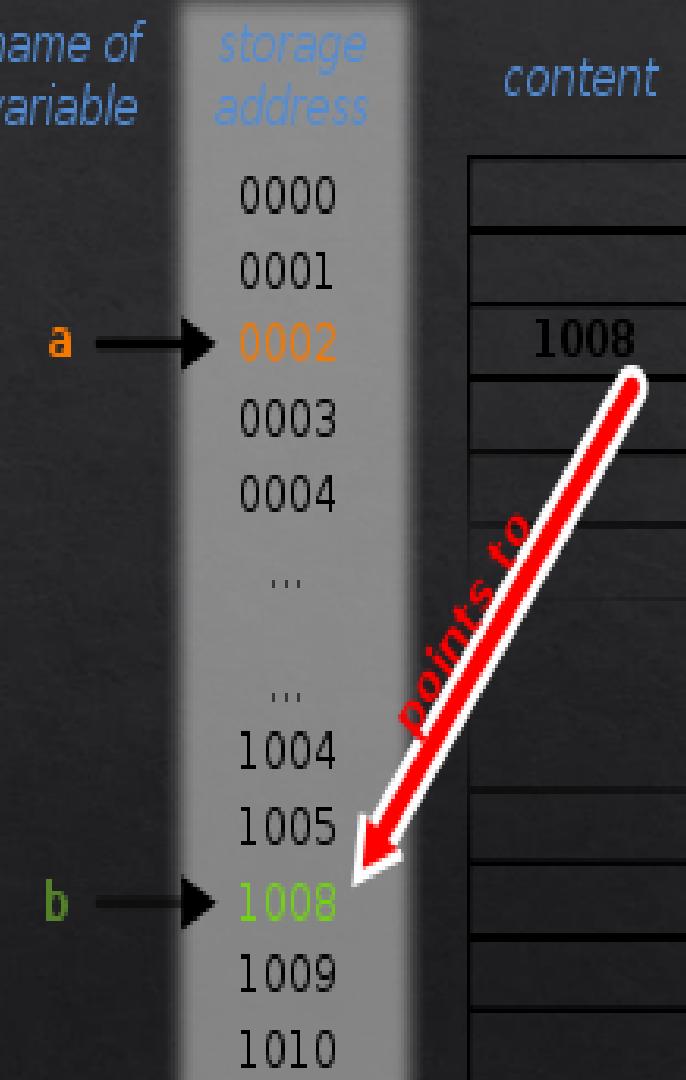
pointers

pointers

a value that references a memory location

pointers

a value that references a memory location



heads

heads

pointer to a commit object

heads

pointer to a commit object

each head has a name

heads

pointer to a commit object

each head has a name

each branch has a head

heads

pointer to a commit object

each head has a name

each branch has a head

current head is aliased as *HEAD* (always in caps)

heads

think of heads as a name of the save file

heads

think of heads as a name of the save file
it links you to the backup (commit)

tags

tags

essentially a special name given to a commit

tags

essentially a special name given to a commit

generally used to mark a special release/edition/milestone

tags

essentially a special name given to a commit

generally used to mark a special release/edition/milestone

ex. v1.0

branching

branching

this part can get confusing so please ask questions

branching

this part can get confusing so please ask questions

where are going with this?

branching

this part can get confusing so please ask questions

where are going with this?

collaboration

branching

think back to commits and saving when faced with options

branching

think back to commits and saving when faced with options

imagine if you wanted to take option a but a friend wanted to take option b

branching

think back to commits and saving when faced with options
imagine if you wanted to take option a but a friend wanted to take option b

example:

you want to help an NPC for a potential reward

branching

think back to commits and saving when faced with options

imagine if you wanted to take option a but a friend wanted to take option b

example:

you want to help an NPC for a potential reward

friend wants to kill the NPC because you can loot the body

branching

you make a save and both do different things

branching

you make a save and both do different things
(commit your work and then make a new branch)

branching

so let's call the original story line "origin"

branching

so let's call the original story line "origin"

now let's assume that you're the one who is in charge, you make the official decision

branching

so let's call the original story line "origin"

now let's assume that you're the one who is in charge, you make the official decision

let's call this branch of the story line "master"

branching

so let's call the original story line "origin"

now let's assume that you're the one who is in charge, you make the official decision

let's call this branch of the story line "master"

your friend's story line (branch) is now called "dev"

branching

so let's call the original story line "origin"

now let's assume that you're the one who is in charge, you make the official decision

let's call this branch of the story line "master"

your friend's story line (branch) is now called "dev"

with me so far?

let's diagram this

branching

branching

origin

branching

origin time

branching

origin → time →

(A)

you two both agree on choice a

branching

origin time

master
↓
(A)

right now there is no “dev”,
only master

branching



HEAD points to the latest save

branching

origin time

HEAD



master



(A)

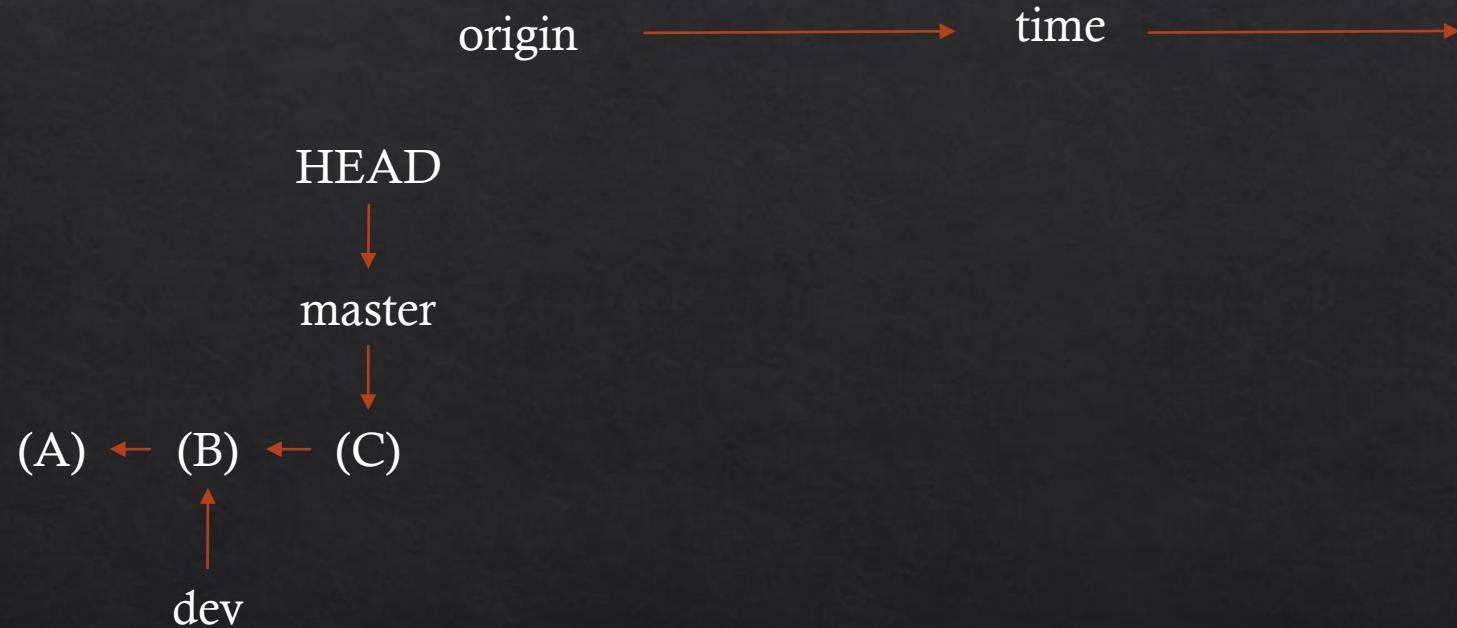
you two both agree on choice a

branching



you two both disagree on choice b
you make a save

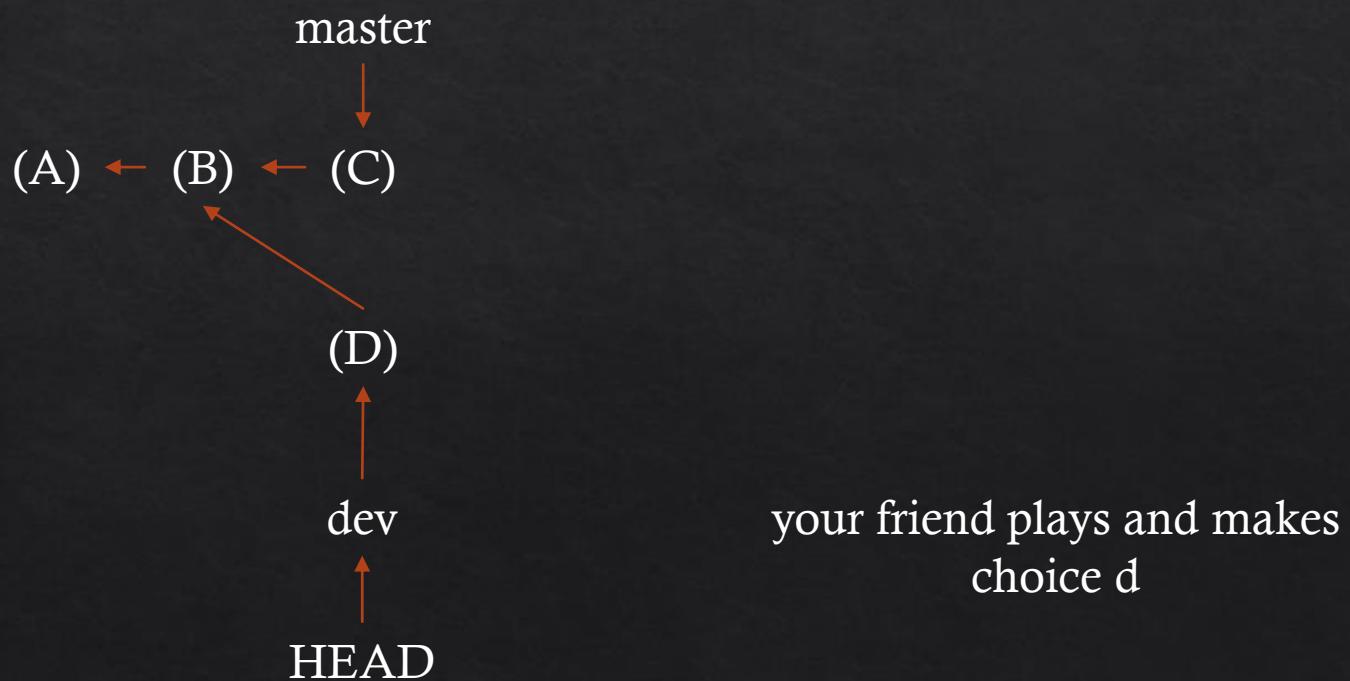
branching



you play and make choice c

branching

origin time



merging branches

merging branches

you want to combine the two story lines but how

merging branches

you want to combine the two story lines but how
both of you made two disjoint decisions

merging branches

you want to combine the two story lines but how

both of you made two disjoint decisions

both decisions have effects on the game that are radically different

merging branches

you want to combine the two story lines but how

both of you made two disjoint decisions

both decisions have effects on the game that are radically different

if you were to merge the two story lines (branches) you'd have *merge conflicts*

merging branches

goal: so you want to remove these conflicts

merging branches

goal: so you want to remove these conflicts

you want all the good parts of *both* decisions kept

merging branches

goal: so you want to remove these conflicts

you want all the good parts of *both* decisions kept

you want all the bad parts of *both* decisions removed

merging branches

imagine taking the story and doing just that

merging branches

imagine taking the story and doing just that
you keep what you want and remove what you don't want

merging branches

imagine taking the story and doing just that

you keep what you want and remove what you don't want

you also make some modifications as necessary to cover plot holes

merging branches

let's relate this back to code

merging branches

let's relate this back to code

let's say you worked on a file

merging branches

let's relate this back to code

let's say you worked on a file

your friend worked on the same file

merging branches

you combine the file

merging branches

you combine the file

pick the parts changed that work

merging branches

you combine the file

pick the parts changed that work

remove the conflicting parts

merging branches

you combine the file

pick the parts changed that work

remove the conflicting parts

make some modifications to make it cleaner/work together (refactoring, et cetera)

merging branches

you combine the file

pick the parts changed that work

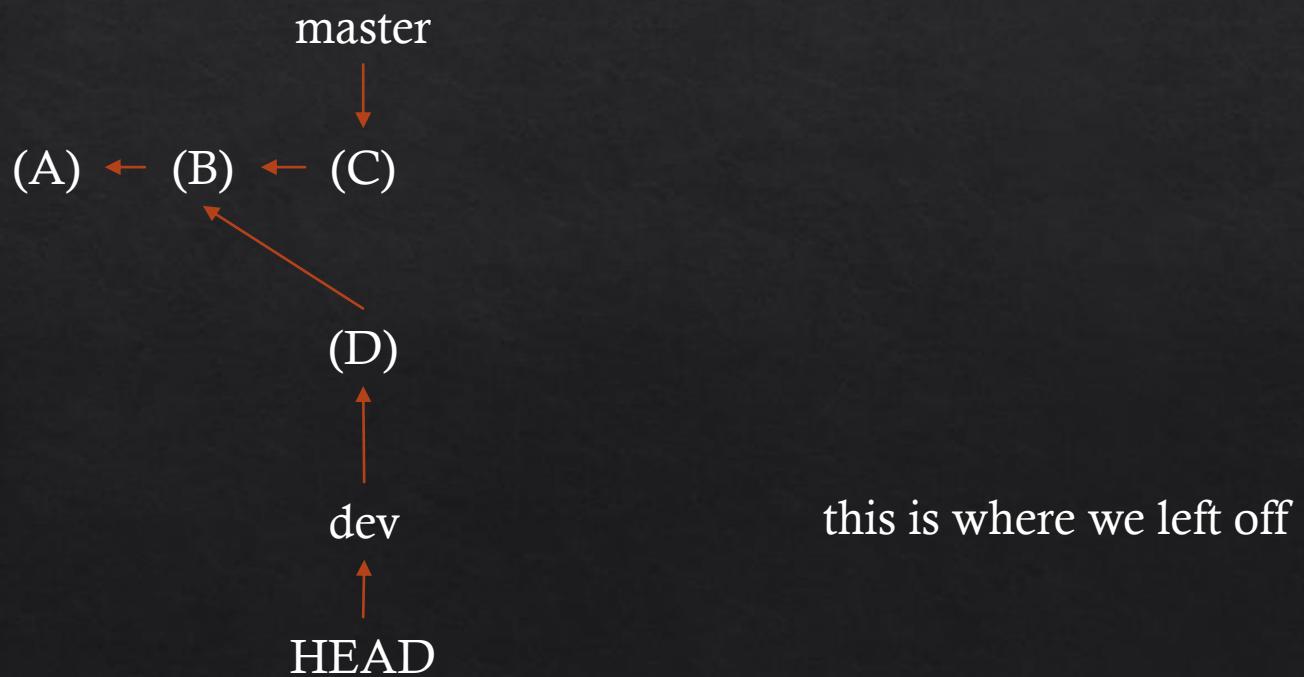
remove the conflicting parts

make some modifications to make it cleaner/work together (refactoring, et cetera)

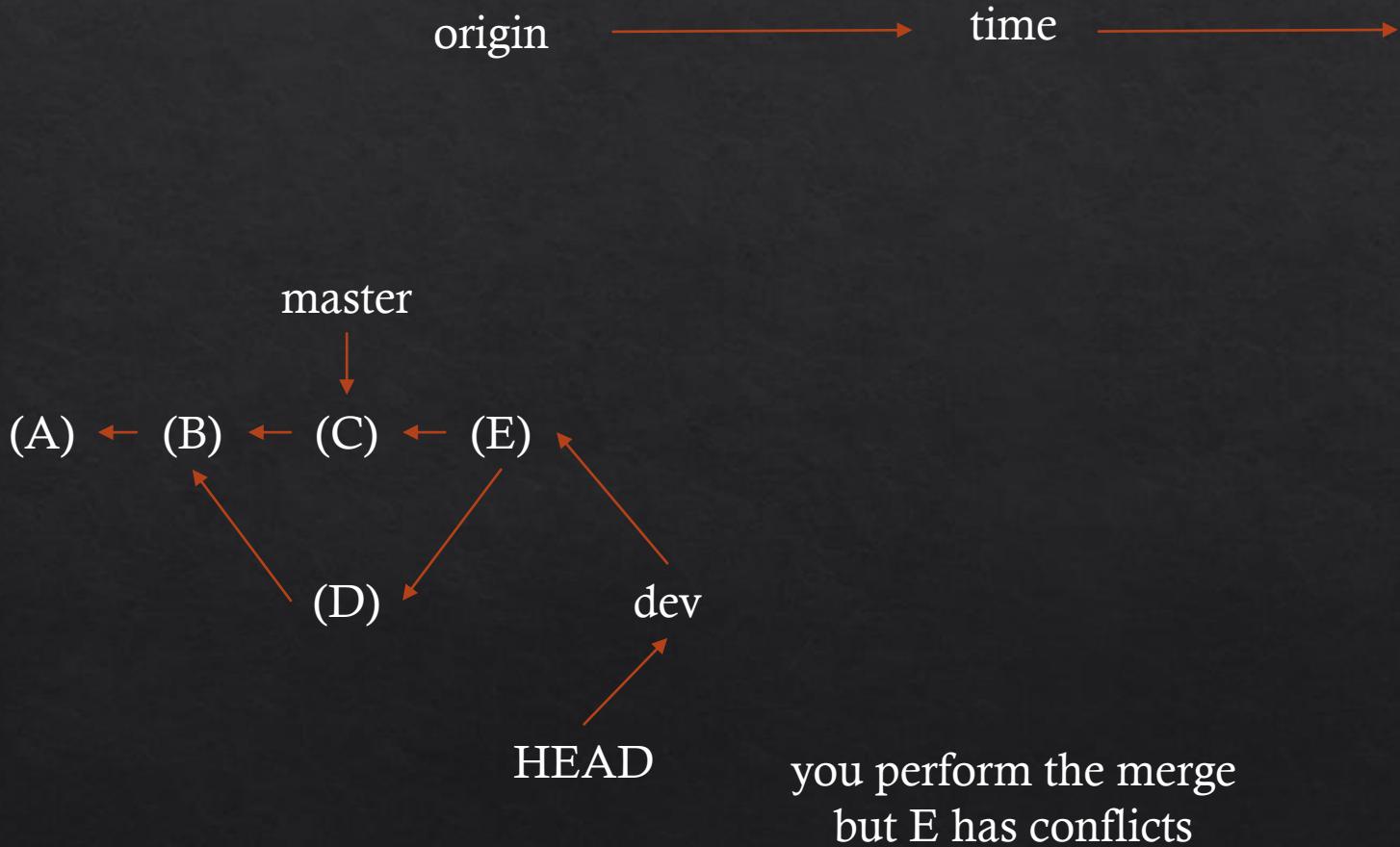
let's diagram this

branching

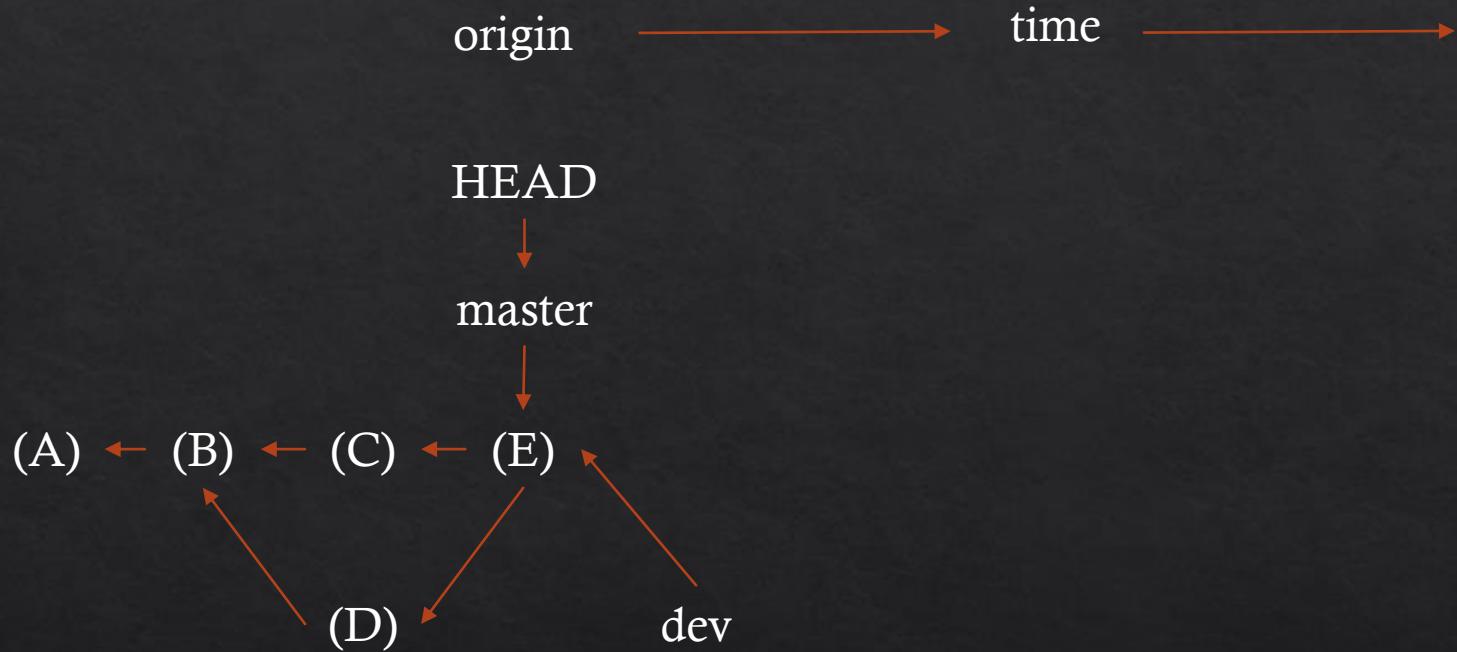
origin time



branching



branching



post-merge
no conflicts

distributed vs centralized

distributed vs centralized

centralized work flow vs distributed work flow

distributed vs centralized

centralized work flow vs distributed work flow

centralized: all work is done locally and synchronized on a server

distributed vs centralized

centralized work flow vs distributed work flow

centralized: all work is done locally and synchronized on a server

distributed: everyone working on the project has a copy of the project and makes changes locally.

distributed vs centralized

centralized work flow vs distributed work flow

centralized: all work is done locally and synchronized on a server

distributed: everyone working on the project has a copy of the project and makes changes locally.

let's diagram differences

centralized

→ time →

friend local

main server

my local

imagine working on a project

centralized

→ time →

friend local

main server

(A)

my local

this is what start with
(main server ~ cloud storage)

centralized

→ time →

friend local

main server

(A)

my local

(A)

you get a local copy

centralized

→ time →

friend local

main server

(A)

my local

(A) → (B)

you make local changes

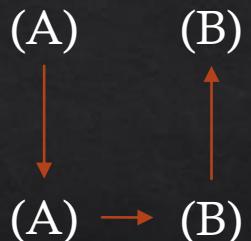
centralized

time →

friend local

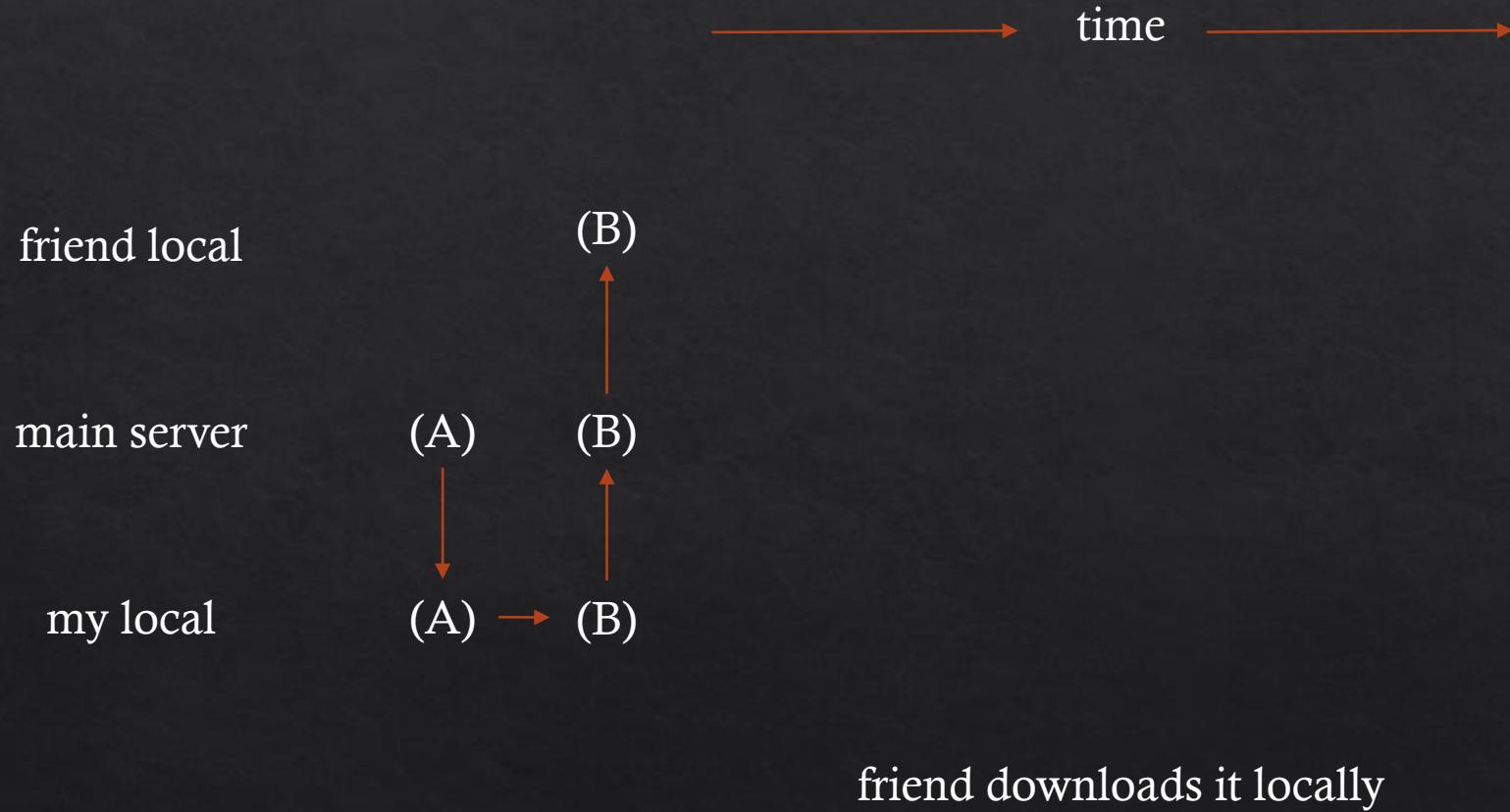
main server

my local

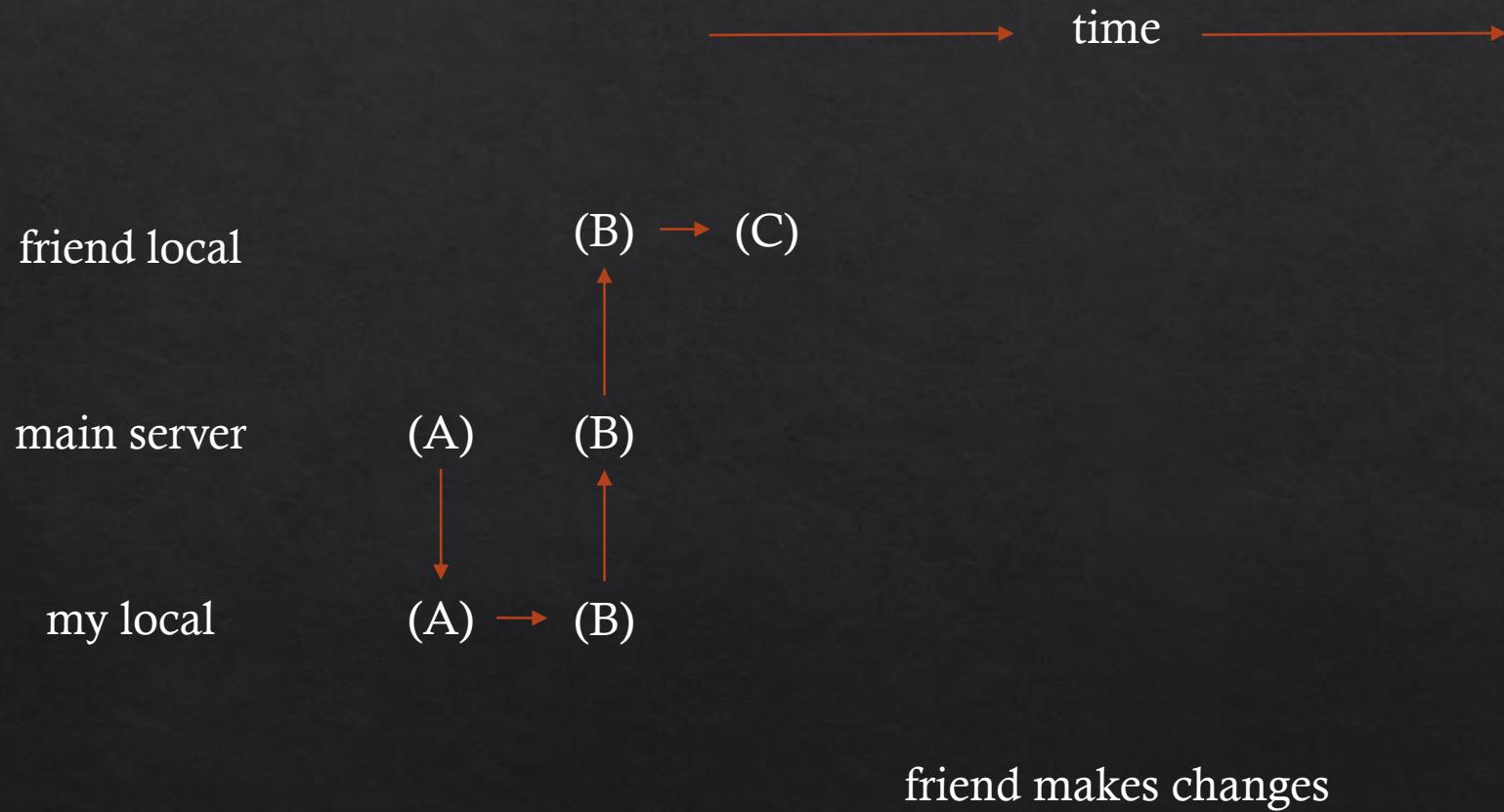


you upload it to the server

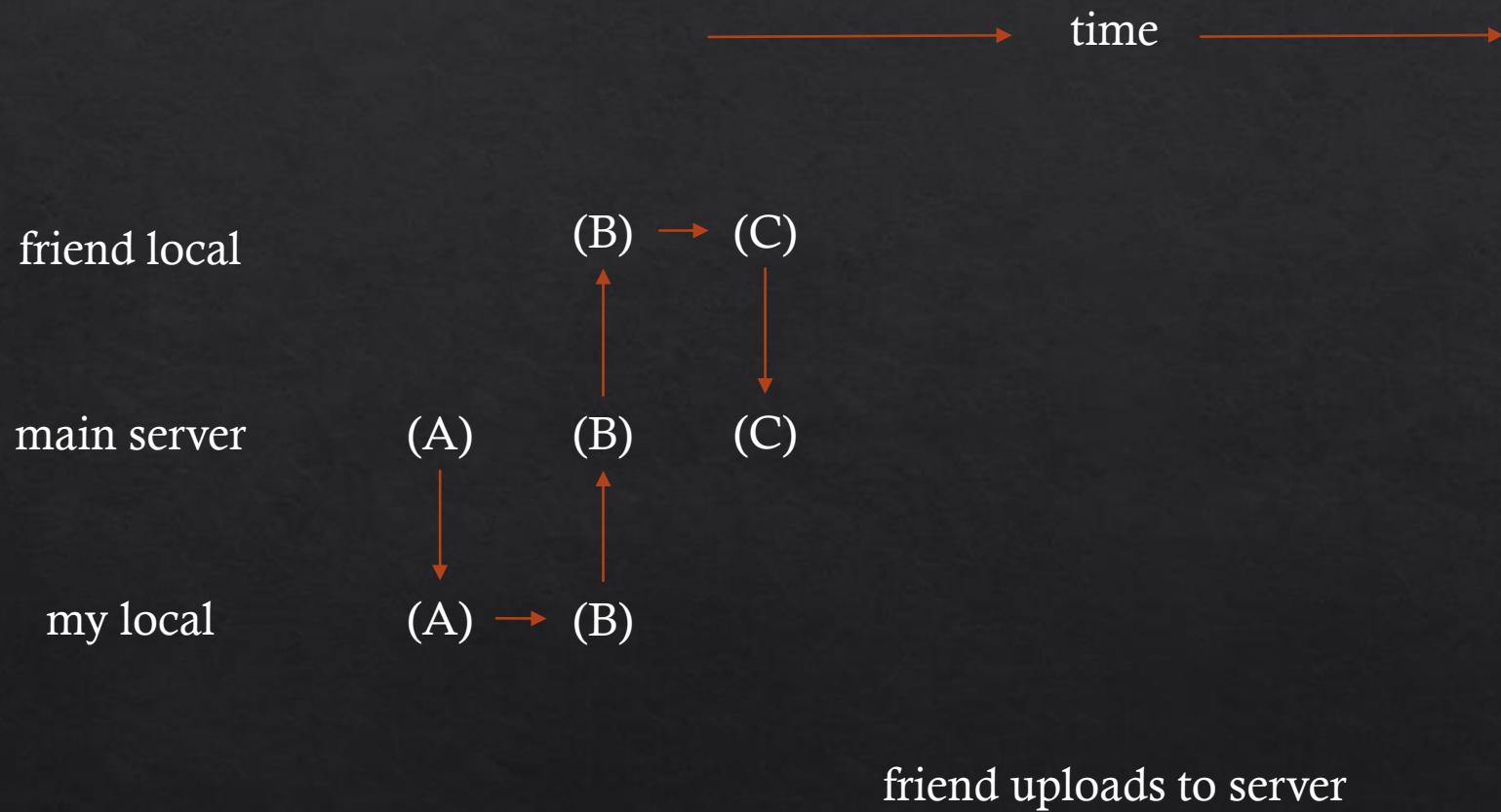
centralized



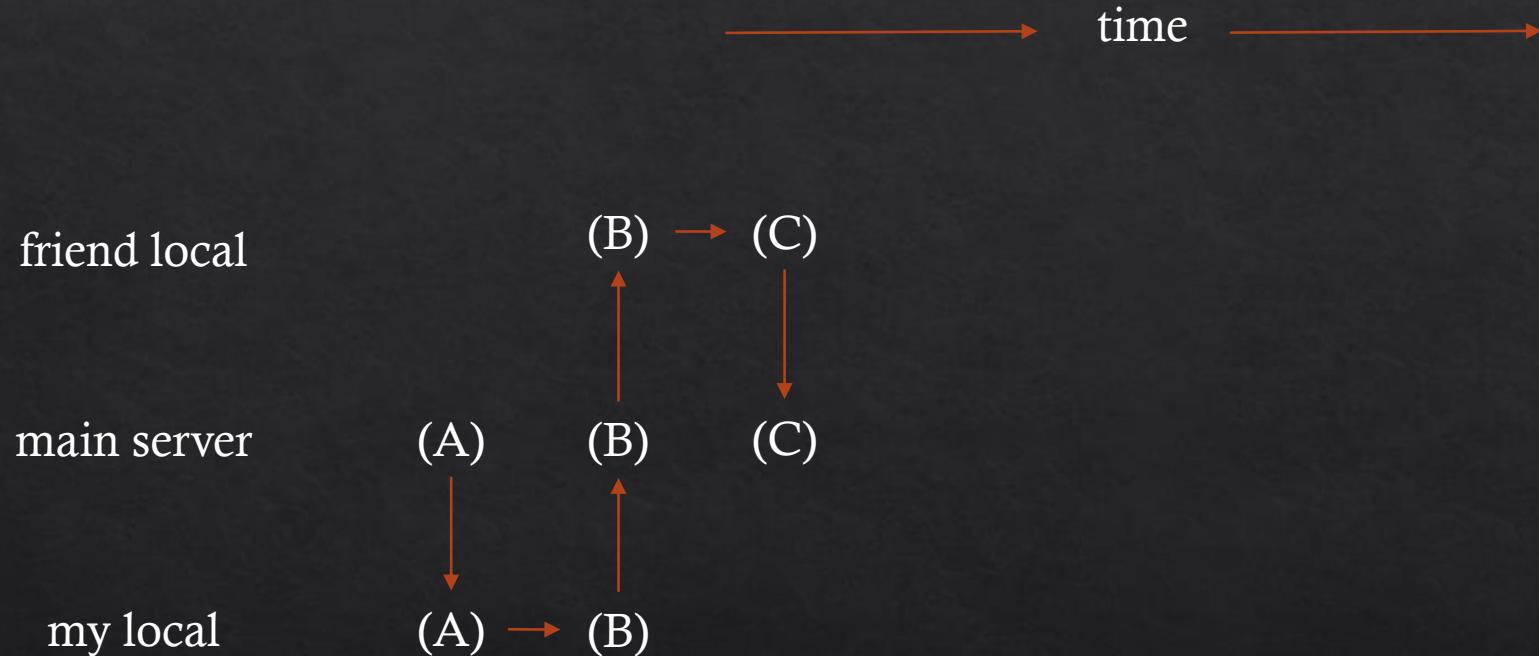
centralized



centralized

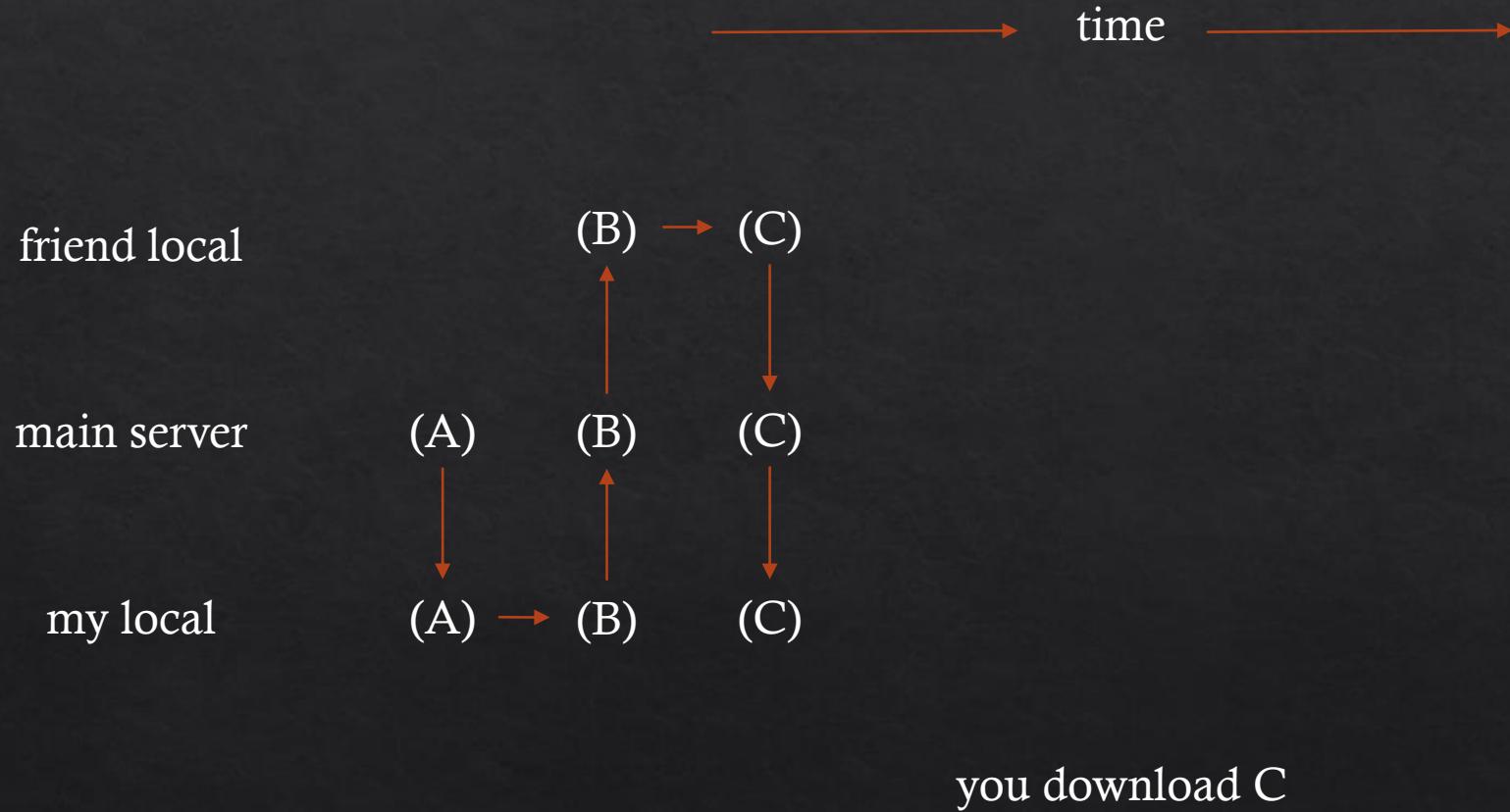


centralized

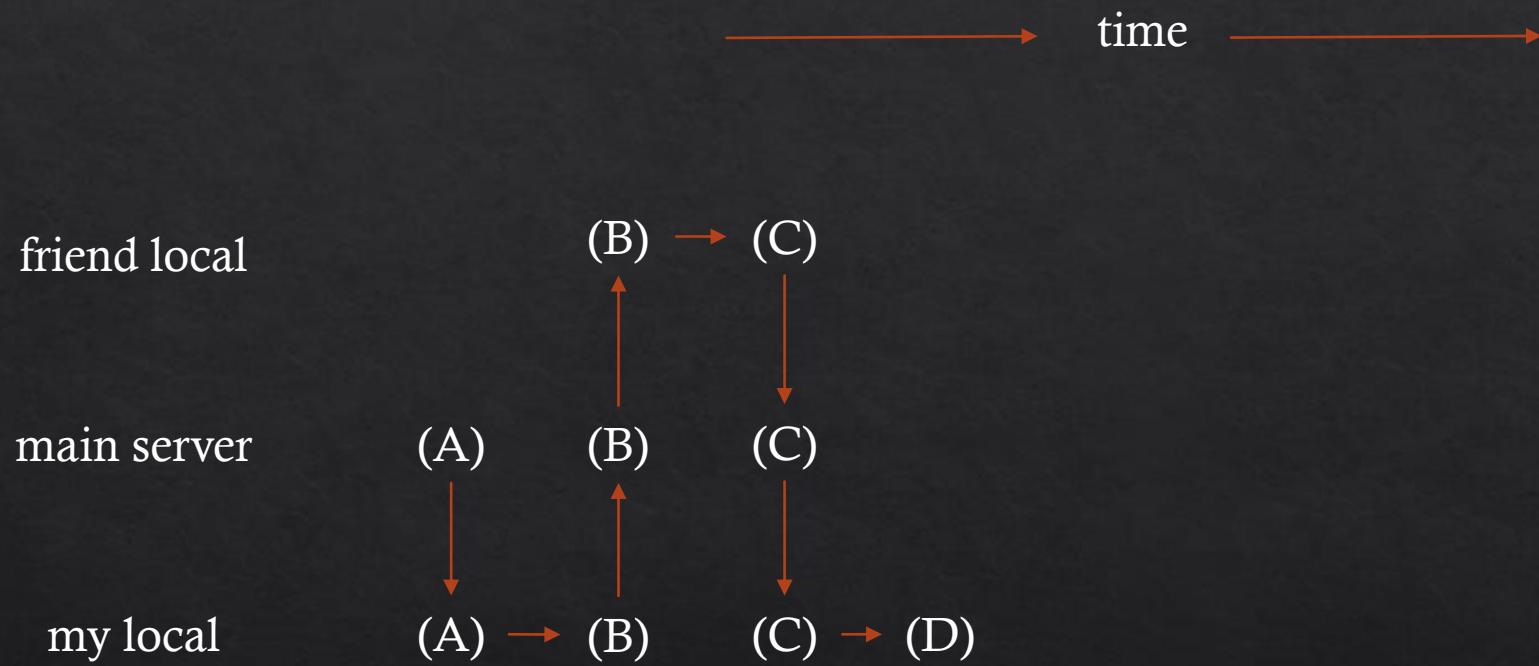


what if you both want to make
changes

centralized

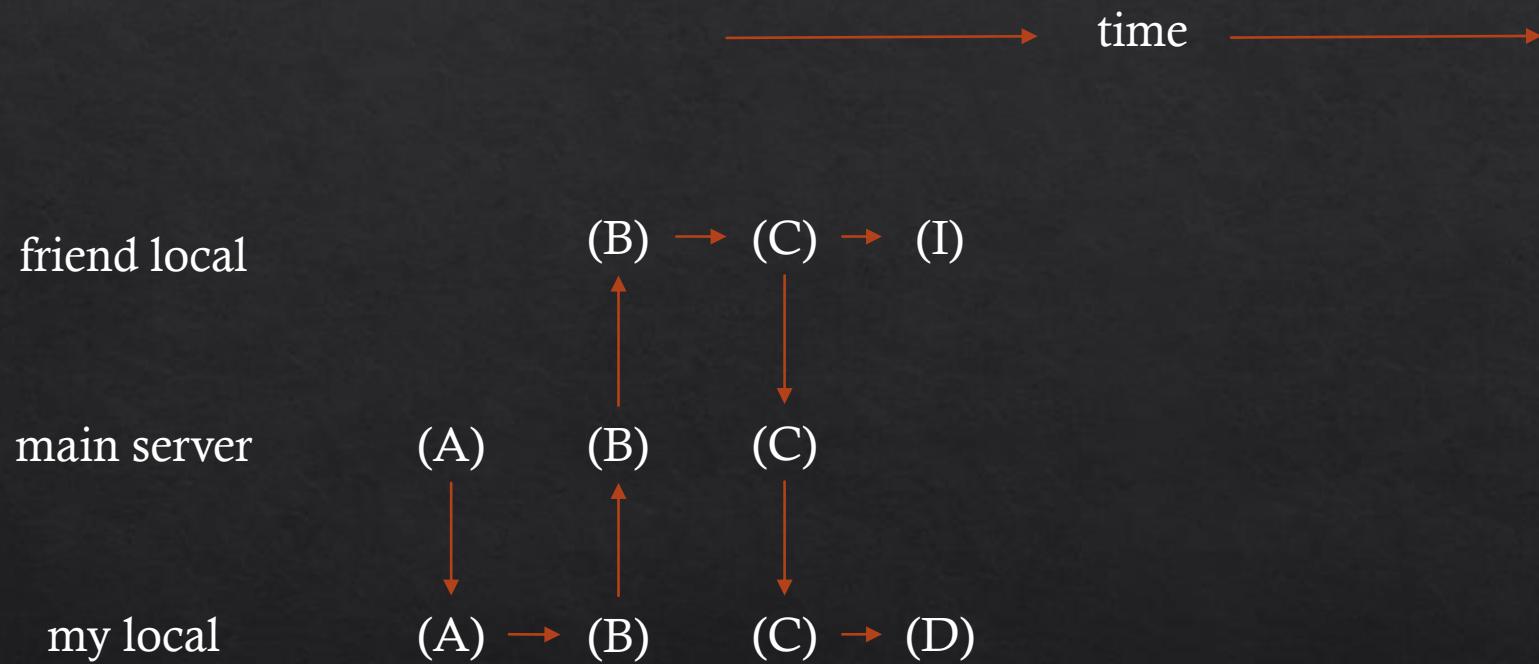


centralized



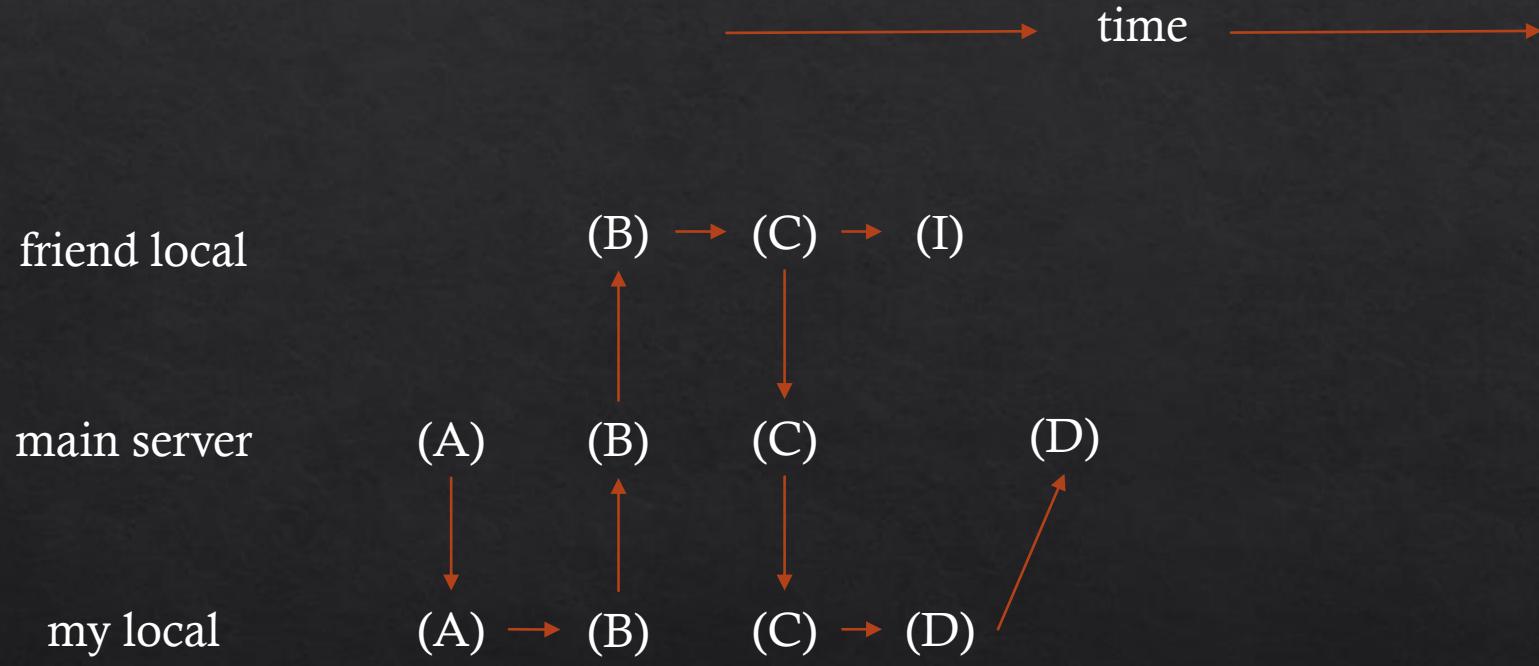
keep working on D

centralized



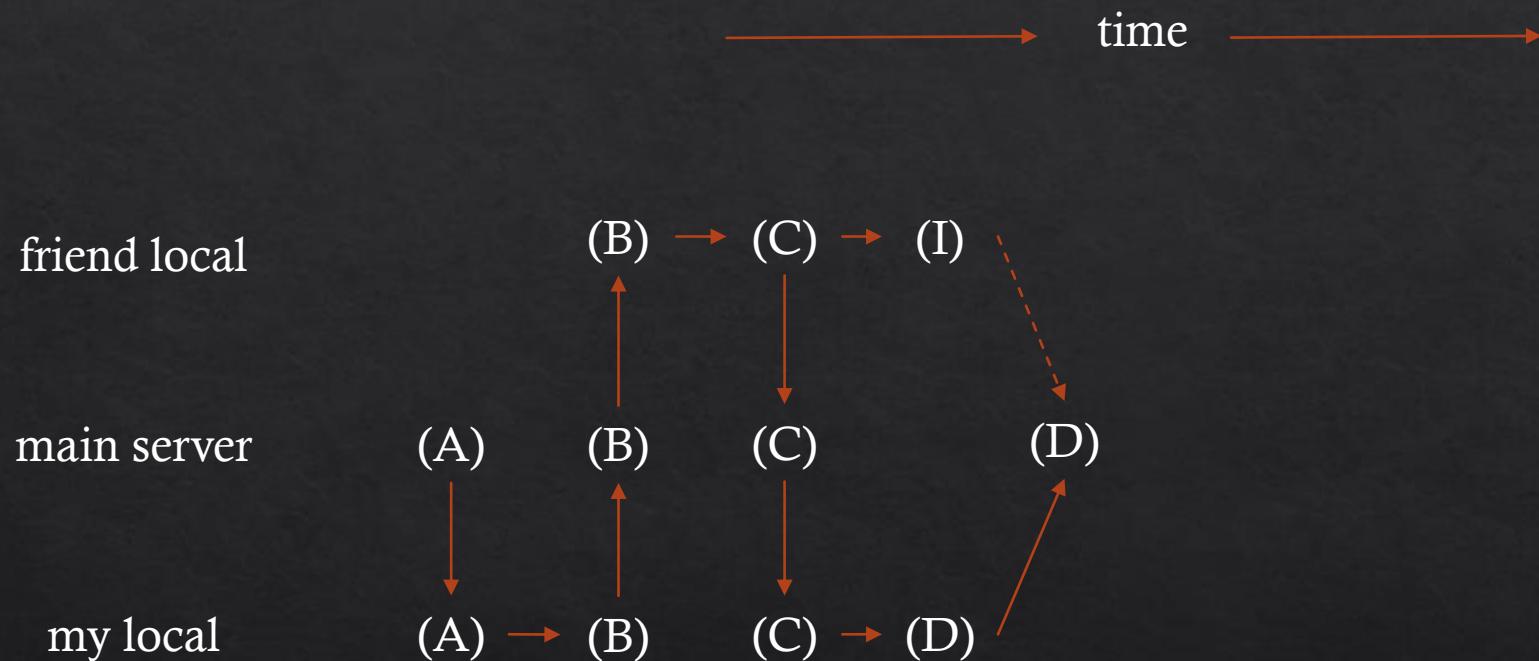
your friend works on I

centralized



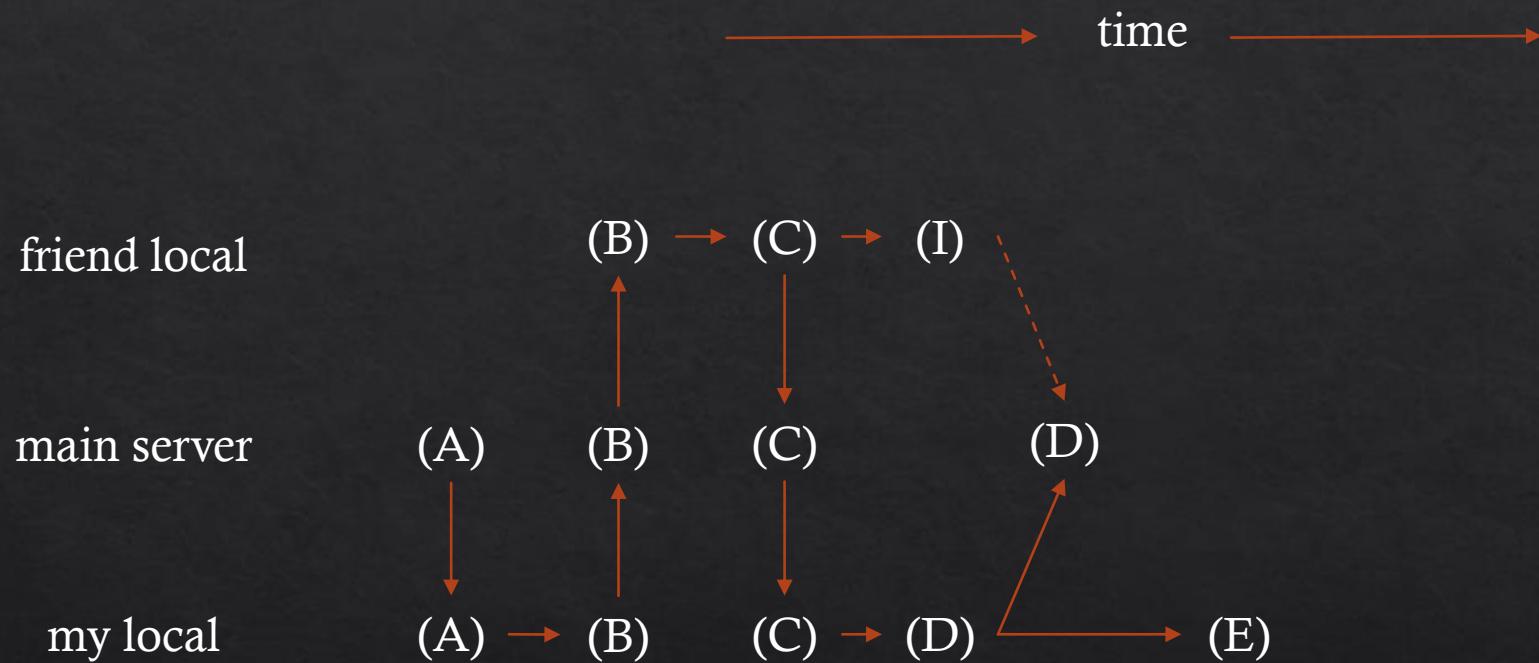
you push your code to the server

centralized



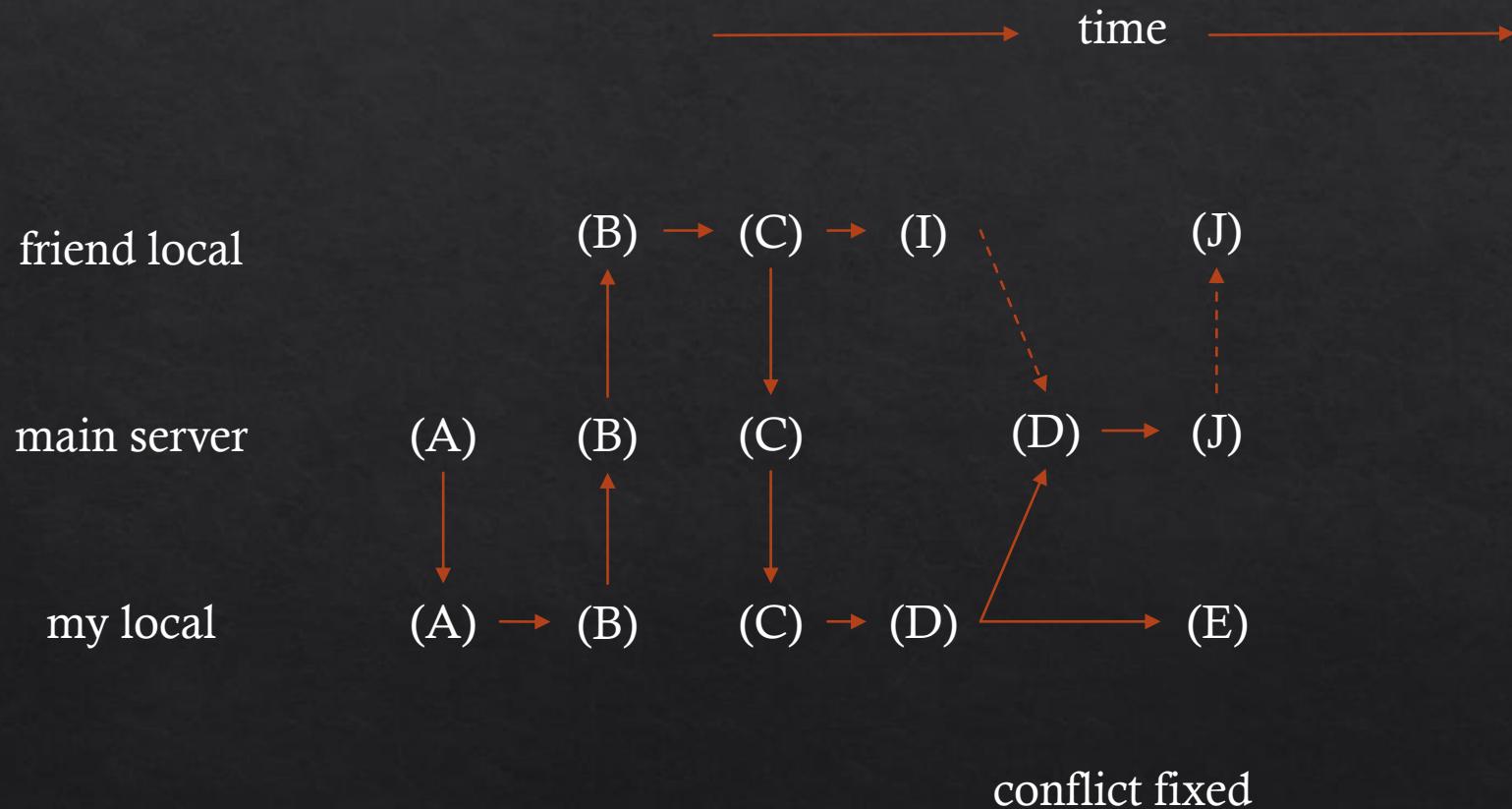
conflict – the bases are not the same
(common base is C)

centralized

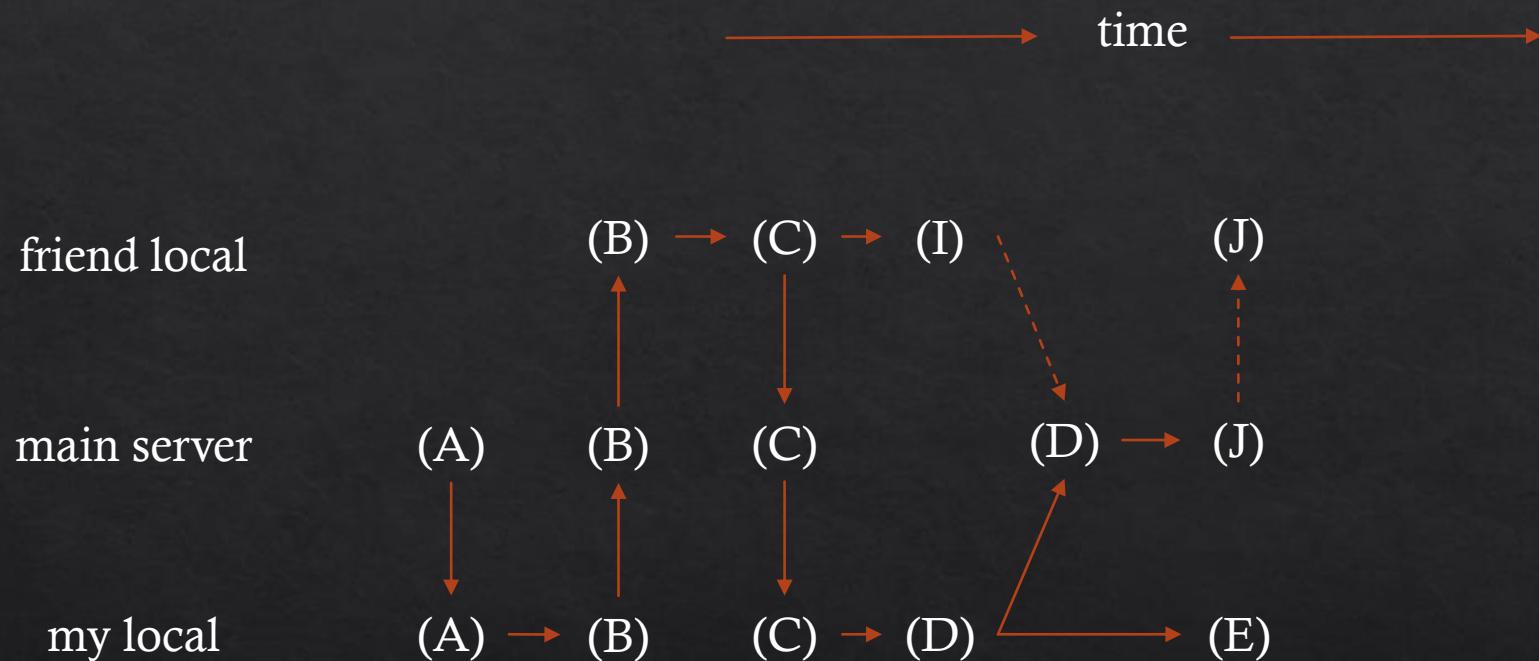


meanwhile, you have code to push

centralized

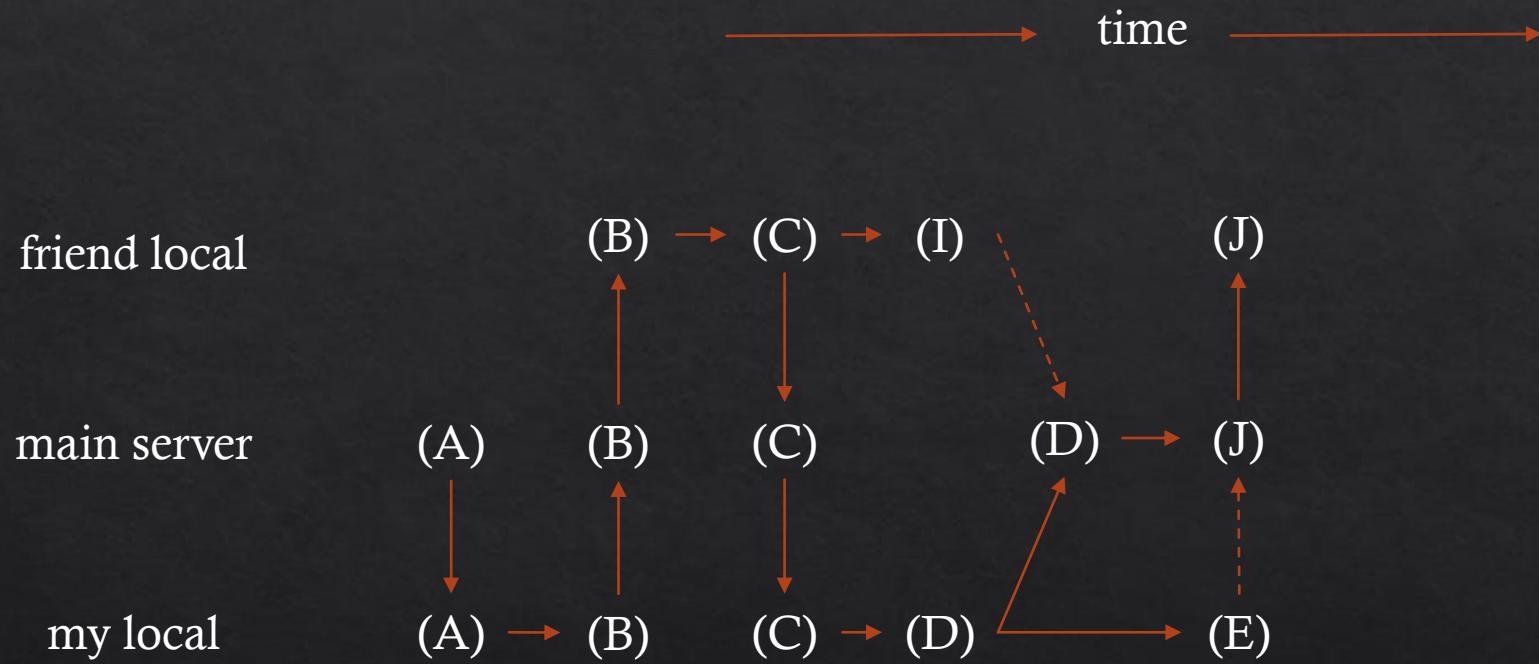


centralized



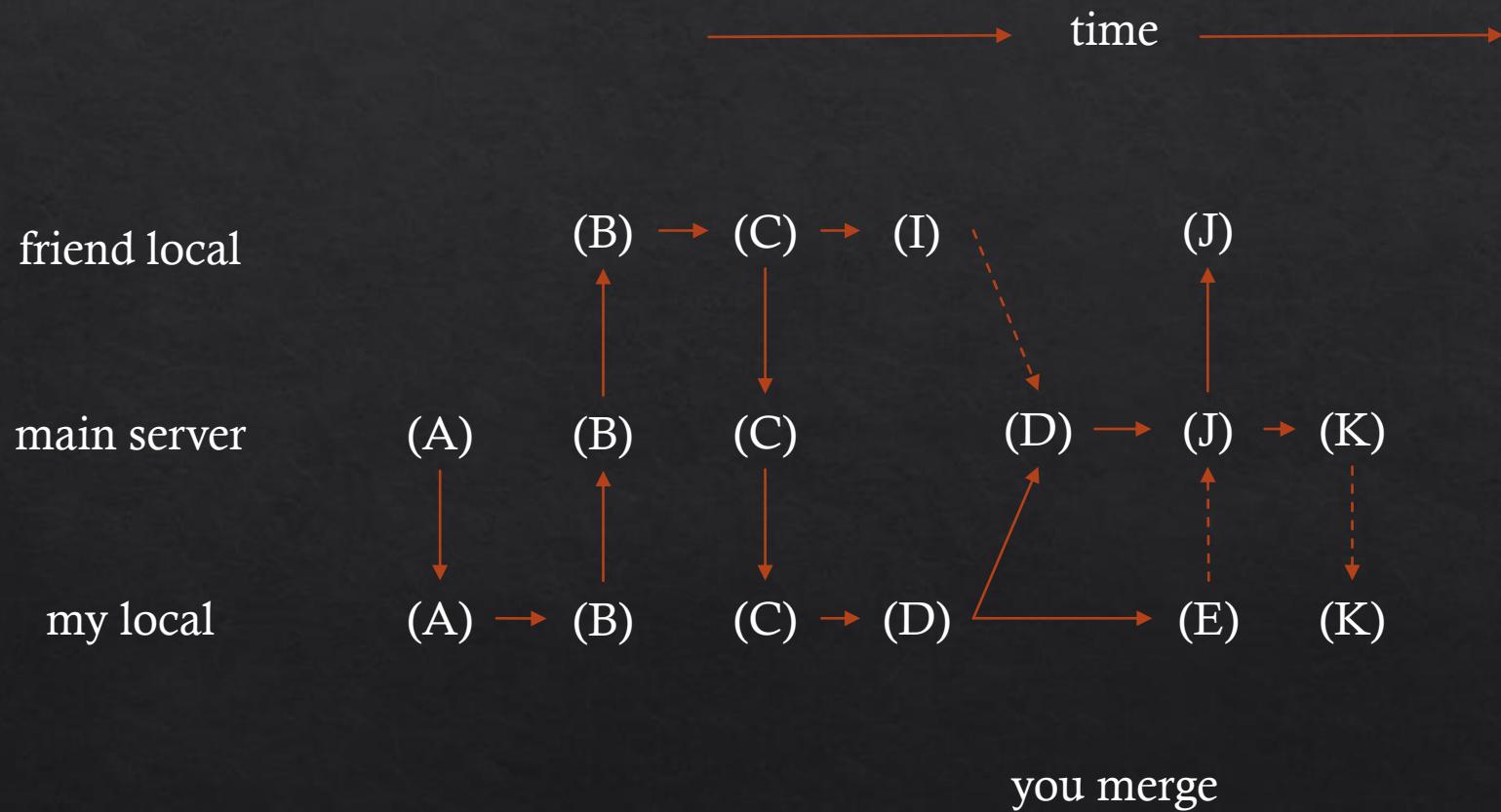
conflict fixed
but now your code is out of sync

centralized

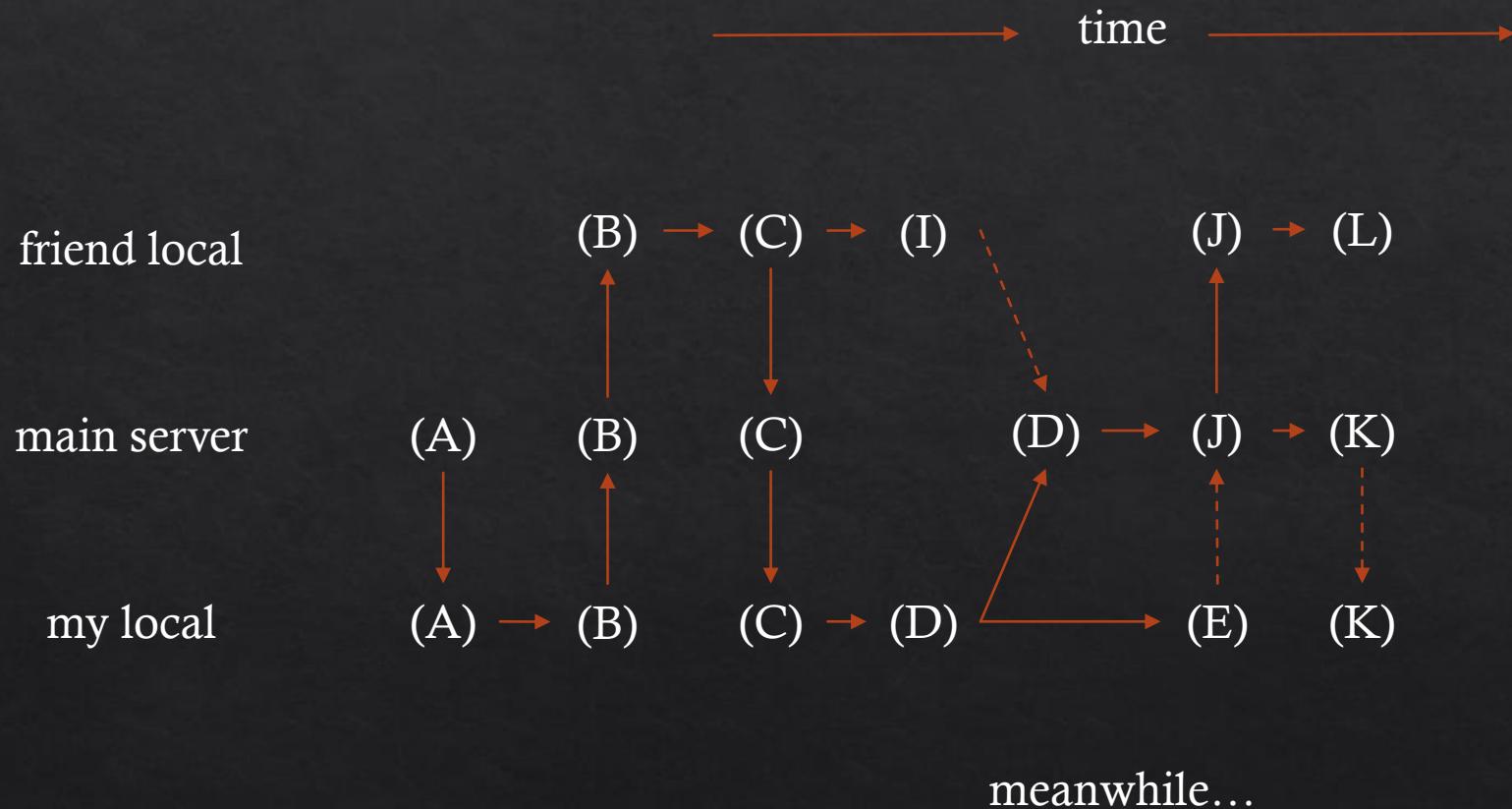


now you have to merge

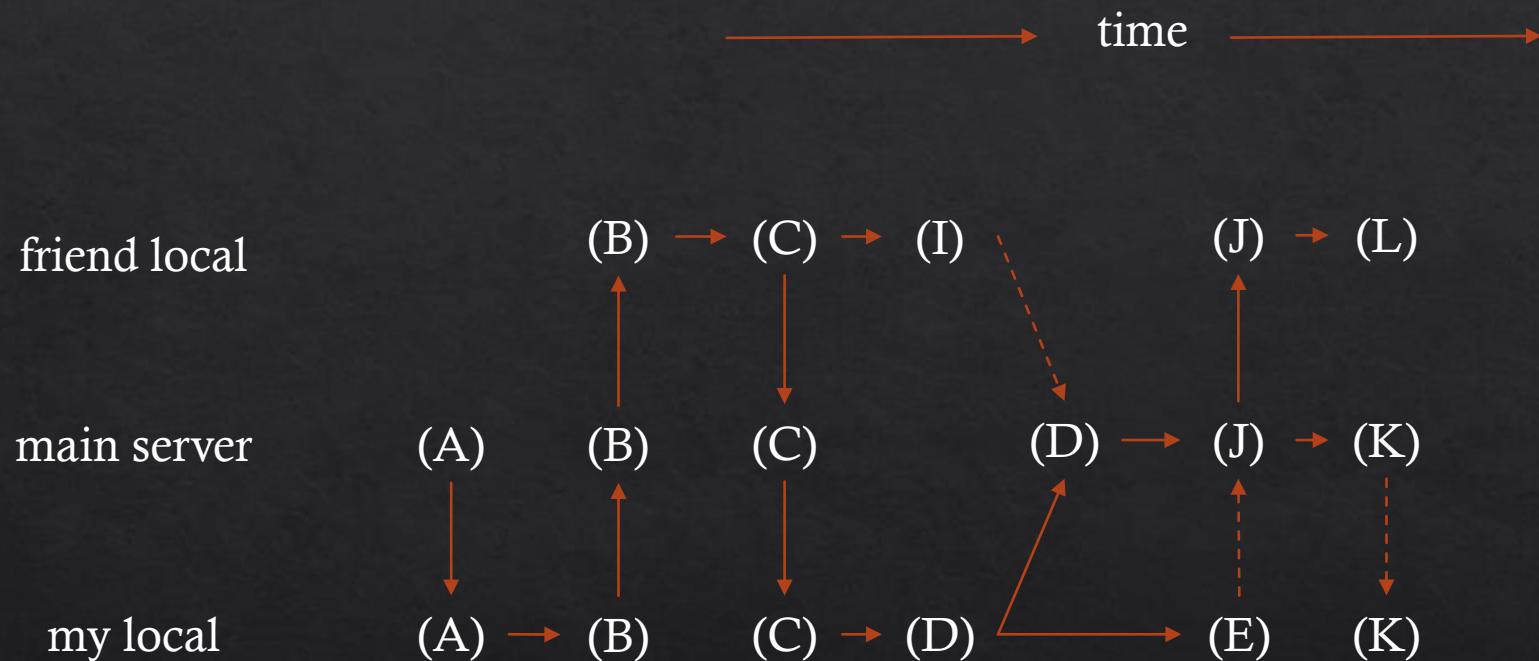
centralized



centralized



centralized



meanwhile...
the process continues

centralized

can you see why this might be a problem

centralized

can you see why this might be a problem

you need network access

centralized

can you see why this might be a problem

you need network access

it's also very easy for your code base to be out of sync

centralized

can you see why this might be a problem

you need network access

it's also very easy for your code base to be out of sync

what if we could always have an updated copy of the codebase

distributed

what if we could always have an updated copy of the codebase

distributed

what if we could always have an updated copy of the codebase
you store this locally and can fix all conflicts locally

distributed

what if we could always have an updated copy of the codebase
you store this locally and can fix all conflicts locally

let's diagram this

distributed

→ time →

origin

let “origin” be the name of the main repository

distributed

→ time →

origin-master

let “origin” be the name of the main repository
the branch name is “master”

distributed

→ time →

origin-master

let “origin” be the name of the main repository
the branch name is “master”
this is handled by the administrator

distributed

→ time →

origin-master

note: origin and master are conventions

distributed

→ time →

origin-master

note: origin and master are conventions
they do not have to be strictly followed

distributed

→ time →

origin-master

note: origin and master are conventions
they do not have to be strictly followed
you can call them whatever you want

distributed

→ time →

origin-master

note: origin and master are conventions
they do not have to be strictly followed
you can call them whatever you want
best practice dictates that you use origin-master

distributed

→ time →

origin-master

nefari0uss

let “nefari0uss” be the name of the my repository

distributed

→ time →

origin-master

nefari0uss-master

let “nefari0uss” be the name of the my repository
the branch name is “master”

distributed

→ time →

origin-master

nefari0uss-master

let “nefari0uss” be the name of the my repository
the branch name is “master”
this is handled by me

distributed

→ time →

friend

origin-master

let “friend” be the name of the my repository

nefari0uss-master

distributed

→ time →

friend-master

origin-master

nefari0uss-master

let “friend” be the name of the my repository
the branch name is “master”

distributed

→ time →

friend-master

origin-master

(A)

nefari0uss-master

this is the initial code base

distributed

→ time →

friend-master

origin-master

nefari0uss-master

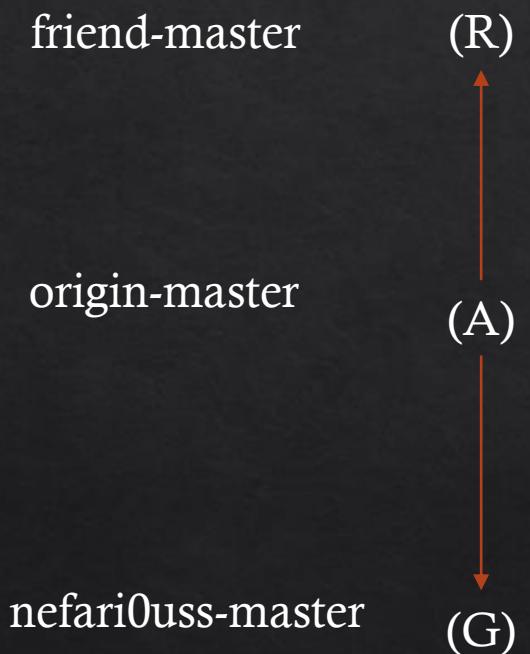
(A)

(G)

you download the entire project locally

distributed

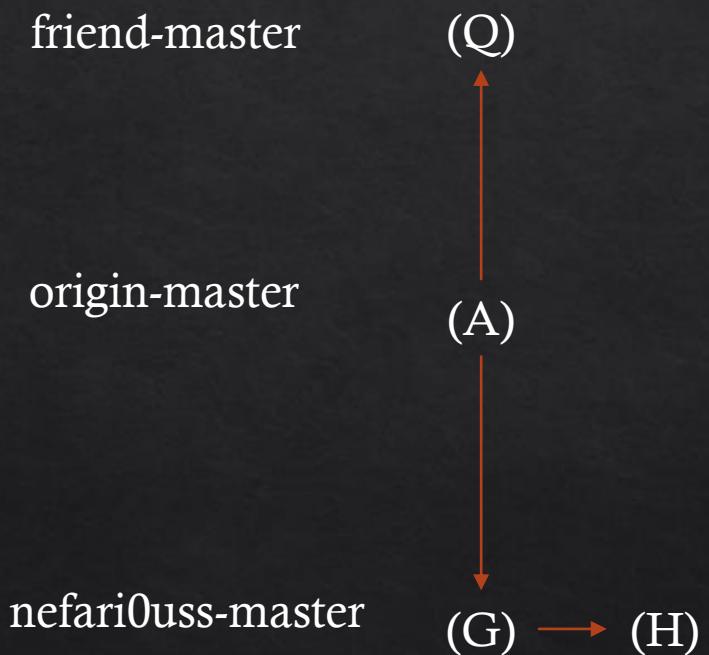
→ time →



as does your friend

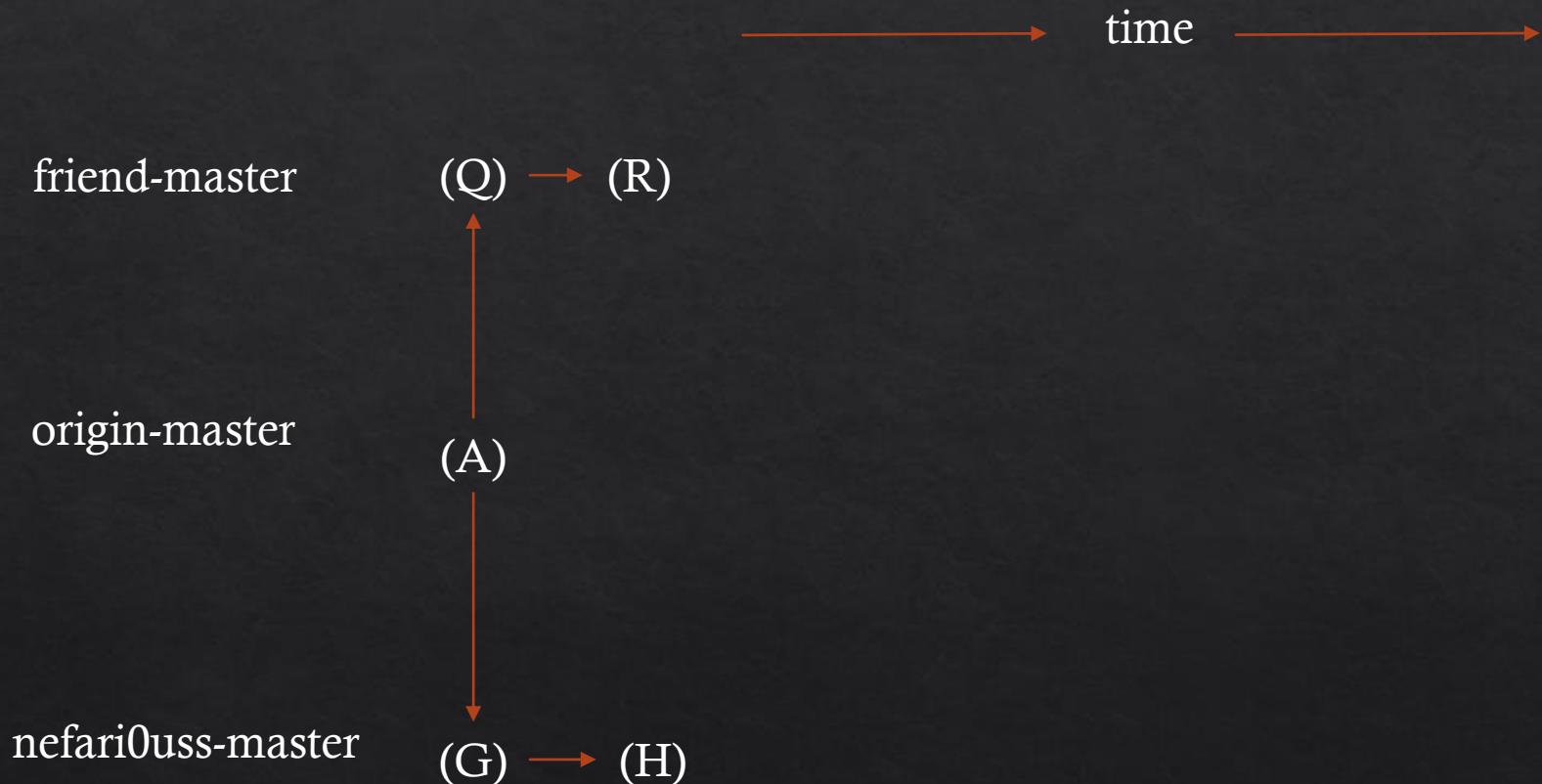
distributed

→ time →



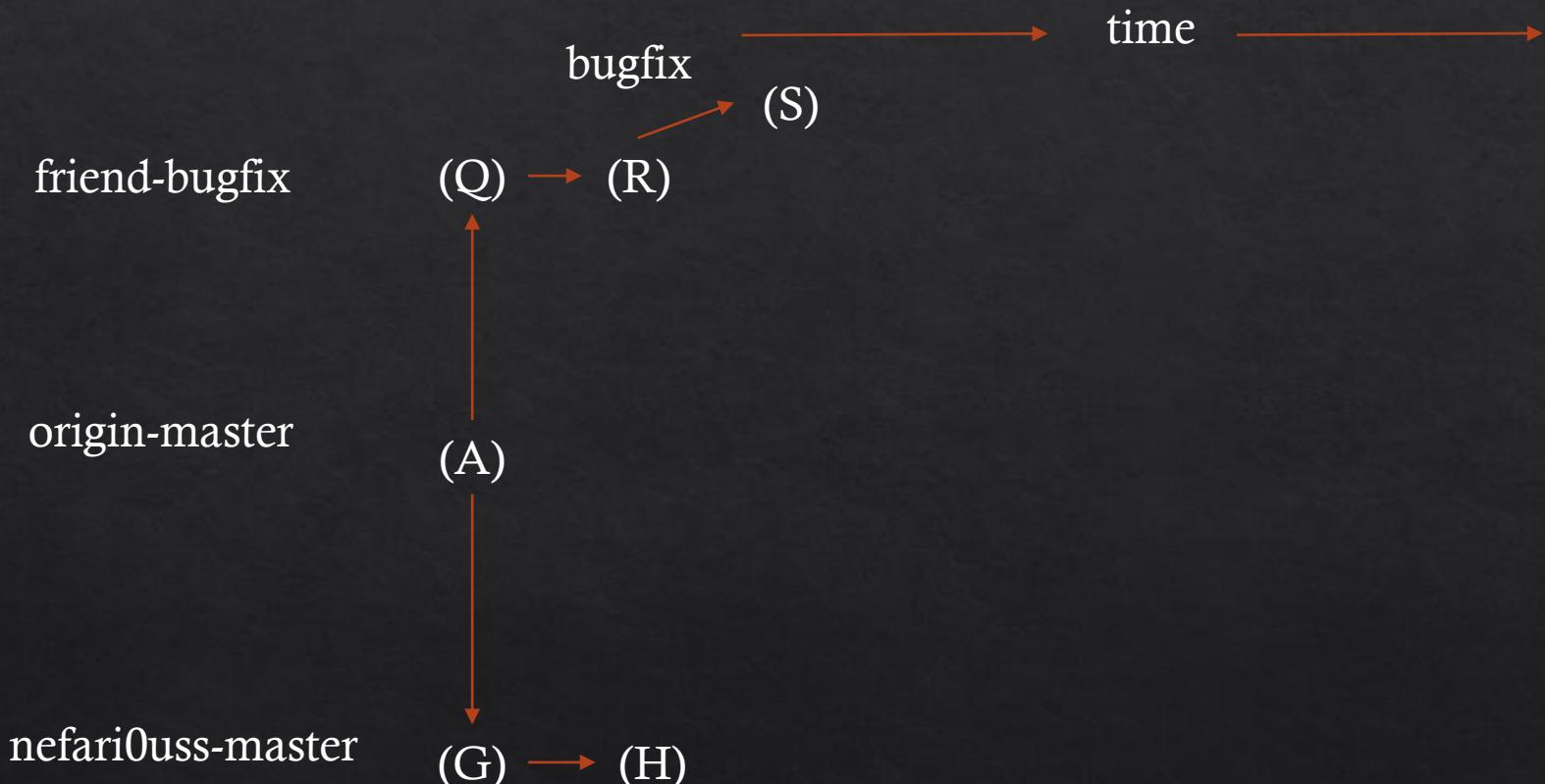
you do some work

distributed



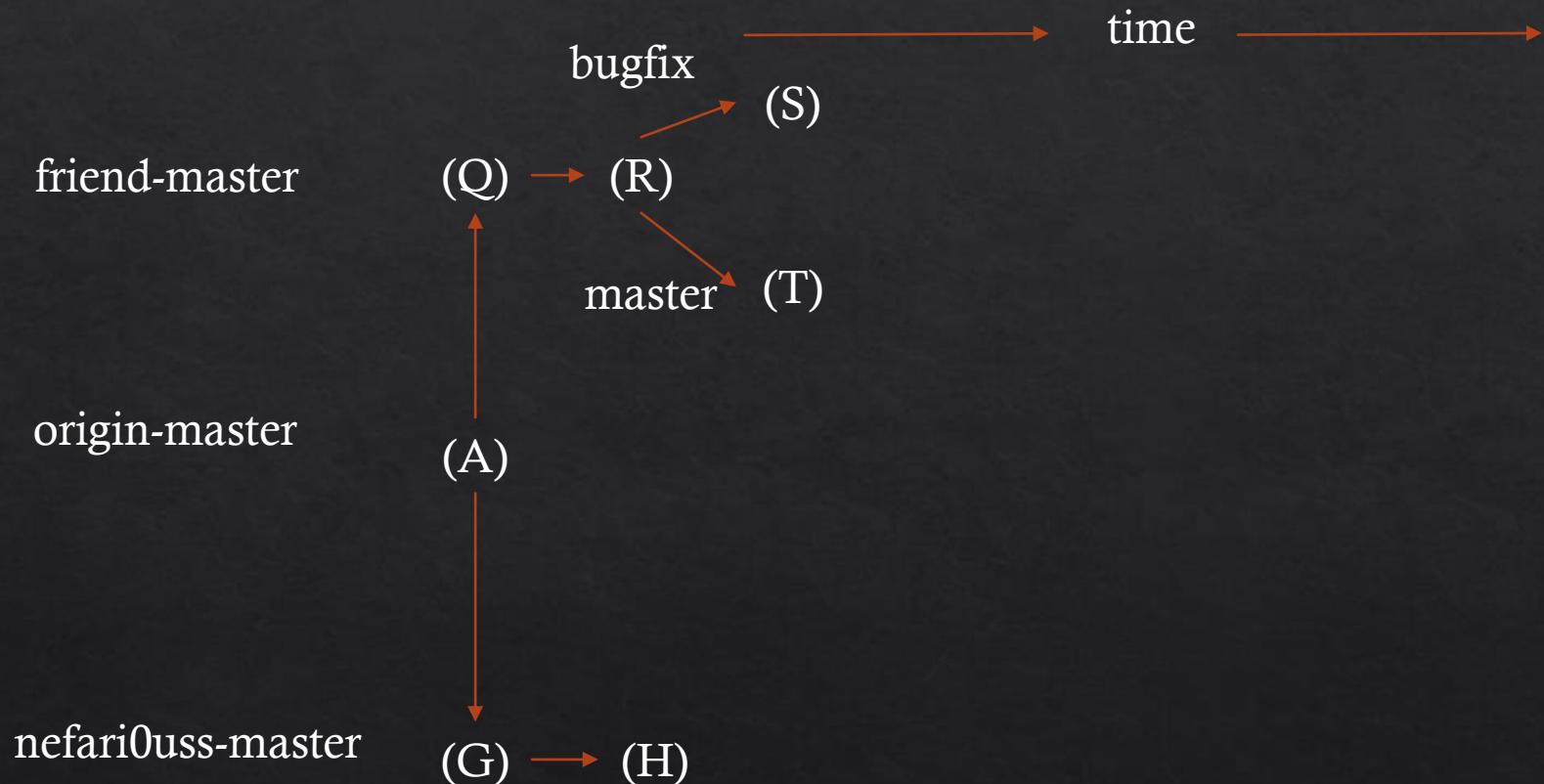
so does your friend

distributed



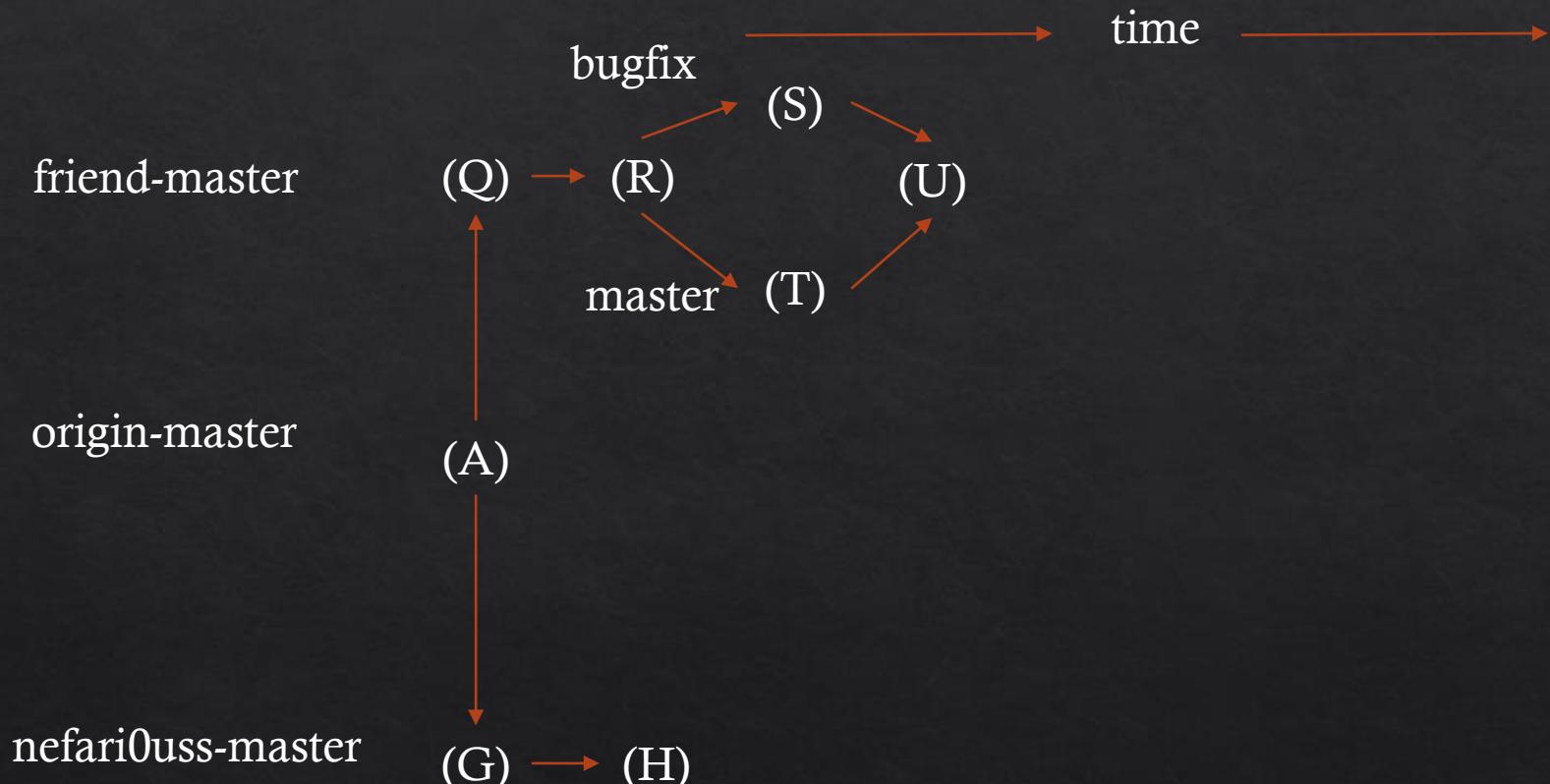
your friend opens a new branch called
bugfix

distributed



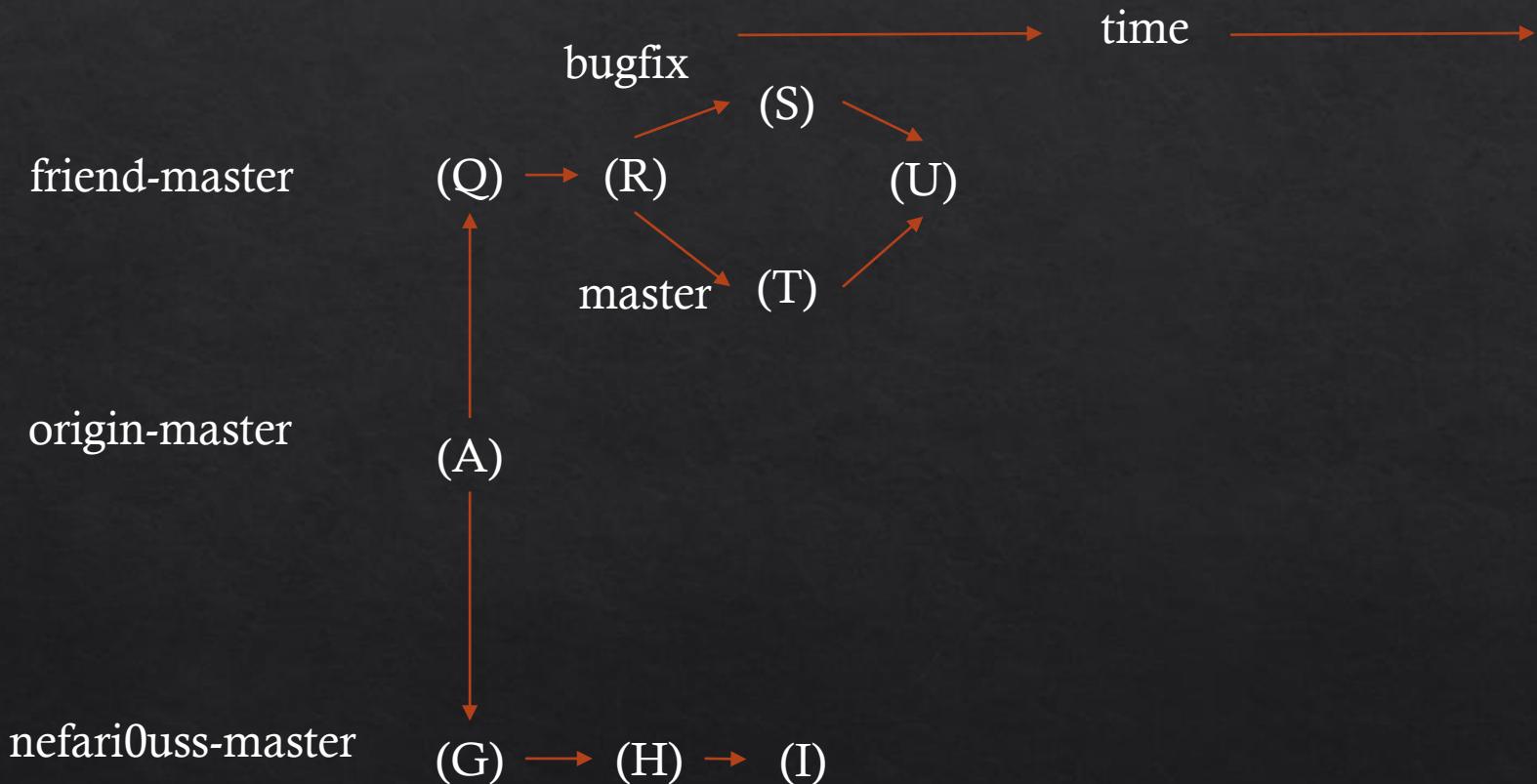
and continues development on master

distributed



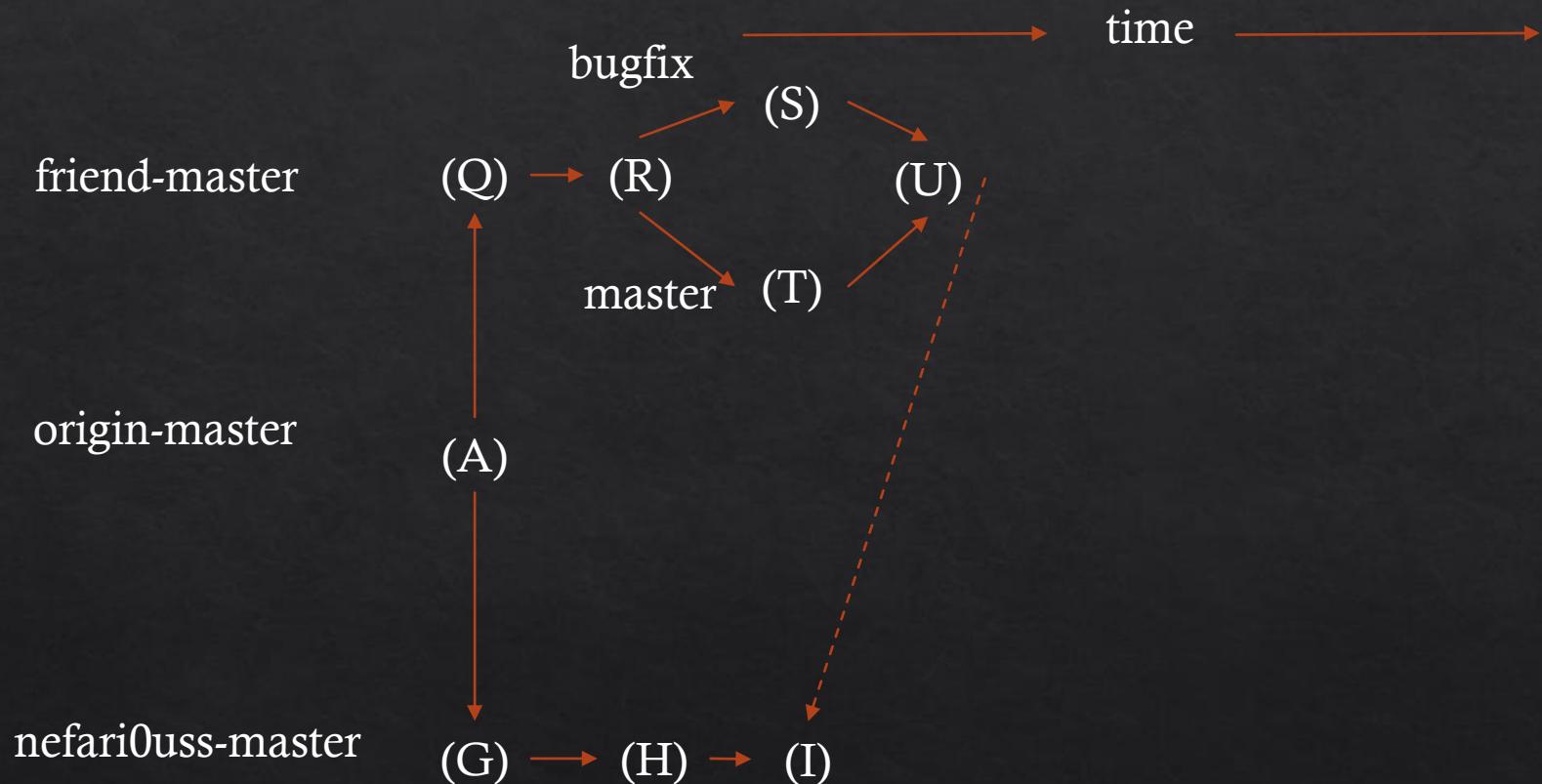
bug fix done
merge with master
bug fix branch closed

distributed



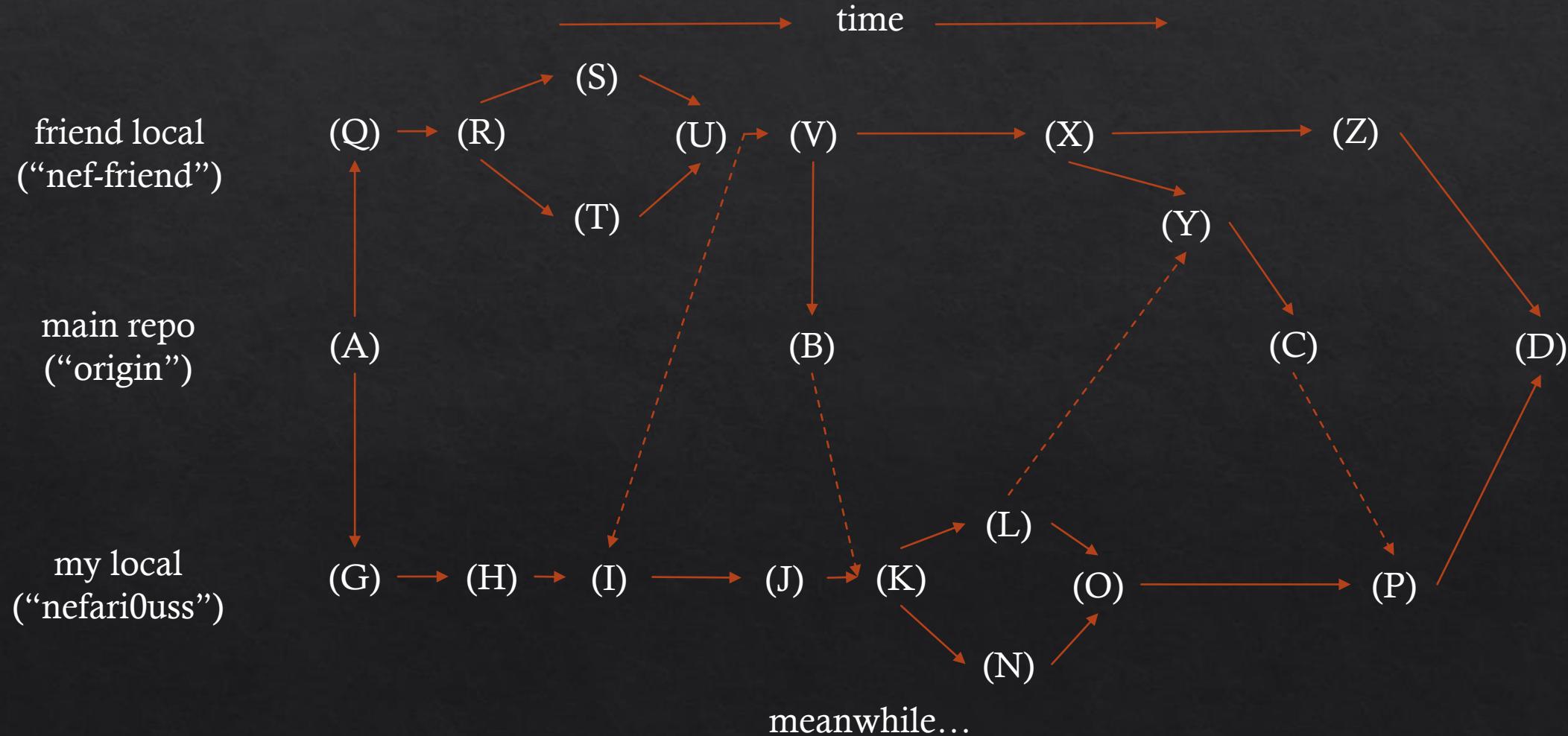
you were waiting on the bug fix

distributed



you pull friend's code

centralized



putting things together

putting things together

will heavily reference game analogy for this

putting things together

repository – the game

putting things together

repository – the game

commit – the game save

putting things together

repository – the game

commit – the game save

metadata – info about the game save

putting things together

repository – the game

commit – the game save

metadata – info about the game save

SHA1 – the internal identifier of the game save

putting things together

repository – the game

commit – the game save

metadata – info about the game save

SHA1 – the internal identifier of the game save

head – the game save name (save 01, save 02, et cetera)

putting things together

repository – the game

commit – the game save

metadata – info about the game save

SHA1 – the internal identifier of the game save

head – the game save name (save 01, save 02, et cetera)

tags – different parts/chapters of the game (part 1, part 2, et cetera)

putting things together

repository – the game

commit – the game save

metadata – info about the game save

SHA1 – the internal identifier of the game save

head – the game save name (save 01, save 02, et cetera)

tags – different parts/chapters of the game (part 1, part 2, et cetera)

branch – an option taken by someone

putting things together

repository – the game

commit – the game save

metadata – info about the game save

SHA1 – the internal identifier of the game save

head – the game save name (save 01, save 02, et cetera)

tags – different parts/chapters of the game (part 1, part 2, et cetera)

branch – an option taken by someone

merging – combining the options taken

one more thing

almost done with internals so stick with me

three states of being

three states of being

working directory

staging area

repository (.git directory)

three states of being

working directory: modify files locally

staging area

repository (.git directory)

three states of being

working directory: modify files locally

staging area: add snapshots/saves to commit

repository (.git directory)

three states of being

working directory: modify files locally

staging area: add snapshots/saves to commit

repository (.git directory): push commit to repository

Working
Directory

Staging
Area

.git directory
(Repository)

Checkout the project

Stage Fixes

Commit

/git-internals

/git-internals

sorry that was so long

/git-internals

sorry that was so long

I wouldn't have spent that long if it wasn't so important

/git-internals

sorry that was so long

I wouldn't have spent that long if it wasn't so important

now on to the for the setup

quick setup

set up git config

set up ssh keys

git config

git config

<https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup>

git config

<https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup>

open bash and type git config

git config

```
$ git config --global user.name "John Doe"
```

```
$ git config --global user.email johndoe@example.com
```

git config

```
$ git config --global user.name "John Doe"
```

```
$ git config --global user.email johndoe@example.com
```

--global sets git to always use these settings

run command without --global to override for specific project

git config

\$ git config --list

will show all settings

ssh keys

ssh keys

ssh keys are a way to identify your computer to the sever

<https://help.github.com/articles/generating-ssh-keys/>

ssh keys

```
$ ls -al ~/.ssh
```

this will list the files in your .ssh directory

(assuming they exist)

ssh keys

```
$ ssh-keygen -t rsa -b 4096 -C "your-git-email"
```

creates a new ssh key using the email as a label

ssh keys

\$ [enter password]

passwords are recommended for security

not necessary for our purposes

general good practise

ssh keys

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

run ssh-agent in background

ssh keys

```
$ ssh-add ~/.ssh/id_rsa
```

add ssh key to ssh-agent

ssh keys

```
$ clip < ~/ssh/id_rsa.pub
```

copy key to clipboard

ssh keys

goto Bitbucket/Github

ssh keys

goto Bitbucket/Github

find settings → ssh keys

ssh keys

goto Bitbucket/Github

find settings → ssh keys

add key → paste

GitHub & BitBucket

GitHub & BitBucket

BitBucket: unlimited private repos

GitHub & BitBucket

BitBucket: unlimited private repos

GitHub: unlimited public repos

GitHub & BitBucket

BitBucket: unlimited private repos

GitHub: unlimited public repos

both have edu plans that are worth using

using git

using git

(this is the part where you get to type stuff)

basic commands

init

clone

status, add, rm

push, pull, merge, fetch

branch, remote

log

reset

`$ git init`

create a new git setup in the current directory

creates .git file

`$ git clone [url]`

clone a git repo in your working directory

should end in .git

`$ git status`

shows you the status of files in working directory

run this command a lot

important side note

important side note

I mentioned staging several times...

important side note

I mentioned staging several times...

let me define it properly

four states of being (pre-commit)

four states of being (pre-commit)

untracked

unmodified

modified

staged

four states of being (pre-commit)

untracked: in the working directory, ignored by git

unmodified

modified

staged

four states of being (pre-commit)

untracked: in the working directory, ignored by git

unmodified: unchanged from last commit; tracked by git

modified:

staged

four states of being (pre-commit)

untracked: in the working directory, ignored by git

unmodified: unchanged from last commit; tracked by git

modified: changed since last commit; tracked by git

staged

four states of being (pre-commit)

untracked: in the working directory, ignored by git

unmodified: unchanged from last commit; tracked by git

modified: changed since last commit; tracked by git

staged: added to staging area; ready to be committed

```
$ git add [filename.ext]
```

add current version of a file to staging index

`$ git add [filename.ext]`

add current version of a file to staging index

CAUTION

\$ git add [filename.ext]

add current version of a file to staging index

CAUTION

if you modify a file after adding it
the old version is staged, not new!

```
$ git rm [filename.ext]
```

remove a file from staging area

`$ git rm [filename.ext]`

remove a file from staging area

CAUTION

`$ git rm [filename.ext]`

remove a file from staging area

CAUTION

`$ rm FileName != $git rm FileName`

`$ git rm [filename.ext]`

remove a file from staging area

CAUTION

`$ rm FileName != $git rm FileName`

deleting a file does not tell git to remove it from staging

going back...

Working
Directory

Staging
Area

.git directory
(Repository)

Checkout the project

Stage Fixes

Commit

let's narrow the scope

what happens between the red and blue boxes

Untracked

Unmodified

Modified

Staged

Add the file

Remove the file

Edit the file

Stage the file

Commit

Working
Directory

Staging
Area

.git directory
(Repository)

Checkout the project

Stage Fixes

Commit

moving forward

what happens between the blue and white boxes

`$ git commit`

create a commit from staged files

`$ git commit`

create a commit from staged files

-m (message) will do inline commit

\$ git commit

create a commit from staged files

-m will do inline commit

Example: \$ git commit -m "this is my commit message"

`$ git commit`

-amend will let you change the last commit

\$ git commit

-amend will let you change the last commit

Example: \$ git commit -m -amend "fixed commit message"

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

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

```
$ git push [remote] [branch]
```

push your changes to a remote

```
$ git push [remote] [branch]
```

push your changes to a remote

Example: \$ git push origin master

`$ git pull [remote] [branch]`

pull a set of changes from a remote

`$ git pull [remote] [branch]`

pull a set of changes from a remote

Example: `$ git pull origin master`

`$ git pull [remote] [branch]`

pull a set of changes from a remote

Example: `$ git pull origin master`

is actually the combination of two commands: fetch and merge

\$ git fetch

update local information from the remote

Example: \$ git fetch origin

\$ git merge

create new commit by combining two branches

\$ git merge

create new commit by combining two branches

Example: (if current branch is master)

```
$ git merge dev
```

how merging works

how merging works

finds most recent common ancestor

how merging works

finds most recent common ancestor

looks at file differences between then and now

how merging works

finds most recent common ancestor

looks at file differences between then and now

applies differences to current branch

merge conflicts

happens when people work on the same file

merge conflicts

happens when people work on the same file

run `$ git status`

merge conflicts

happens when people work on the same file

run \$ git status

edit conflicted file

merge conflicts

happens when people work on the same file

run \$ git status

edit conflicted file

<<< === >>> mark conflicts

merge conflicts

happens when people work on the same file

run \$ git status

edit conflicted file

<<< === >>> mark conflicts

resolve conflicts and then stage files

`$ git branch [arguments]`

does different things depending on the arguments

...

welcome to git

`$ git branch [arguments]`

no arguments: list branches (heads)

`$ git branch`

\$ git branch [arguments]

no arguments: list branches (heads)

```
$ git branch
```

one argument: create new branch and switch to it

```
$ git branch -b dev
```

\$ git branch [arguments]

no arguments: list branches (heads)

```
$ git branch
```

one argument: create new branch and switch to it

```
$ git branch -b dev
```

-d: delete branch

```
$ git branch -d dev
```

\$ git branch [arguments]

CAUTION

\$ git branch [arguments]

CAUTION

\$ git branch -b [branch name]

`$ git branch [arguments]`

CAUTION

`$ git branch -b [branch name]`

will create a new branch off your current working directory

`$ git branch [arguments]`

CAUTION

`$ git branch -b [branch name]`

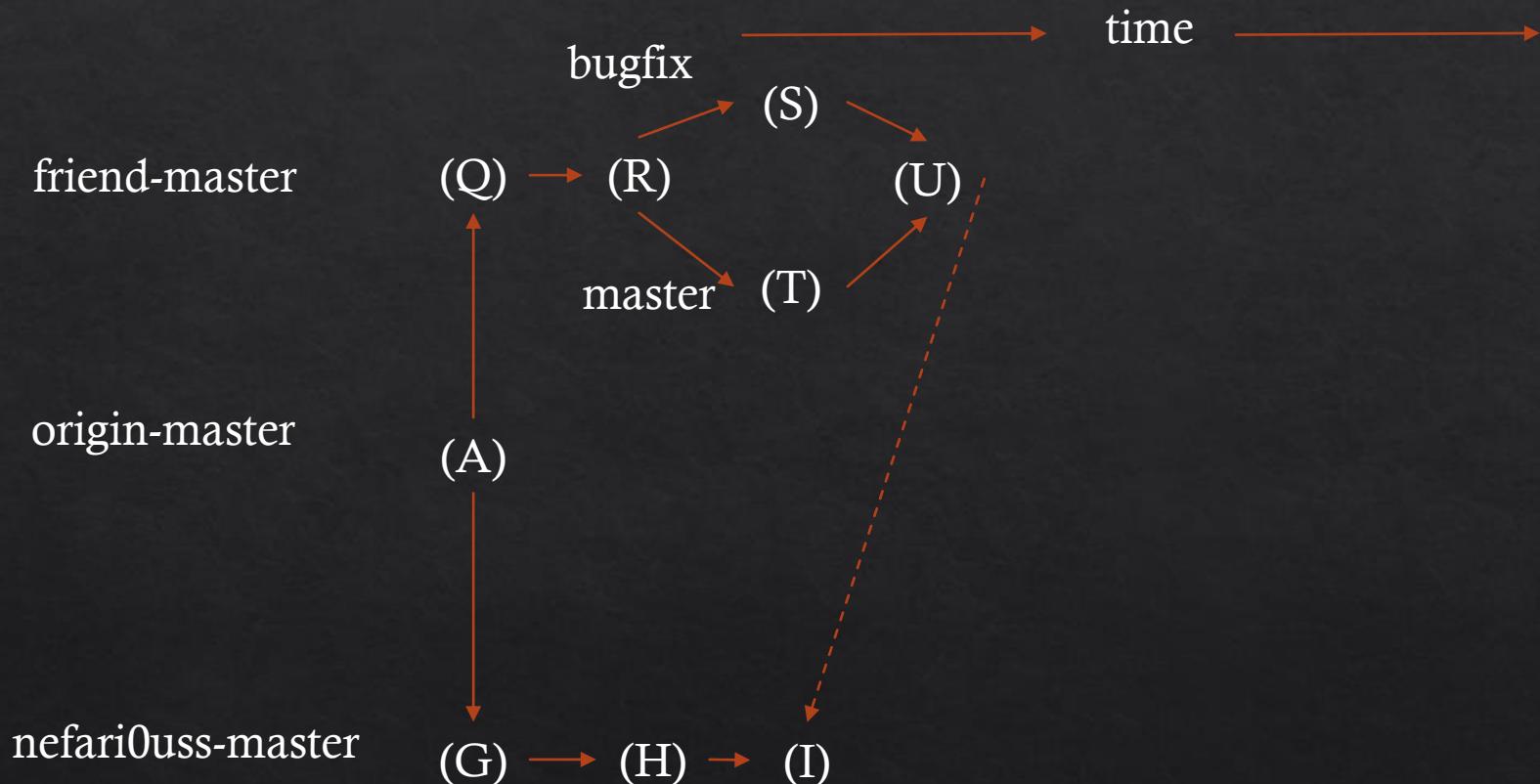
will create a new branch off your current working directory
best to stash/checkout all files before branching

\$ git remote [arguments]

remote is the url in which you are pushing/pulling to
in \$ git push origin/master, “origin” is the remote

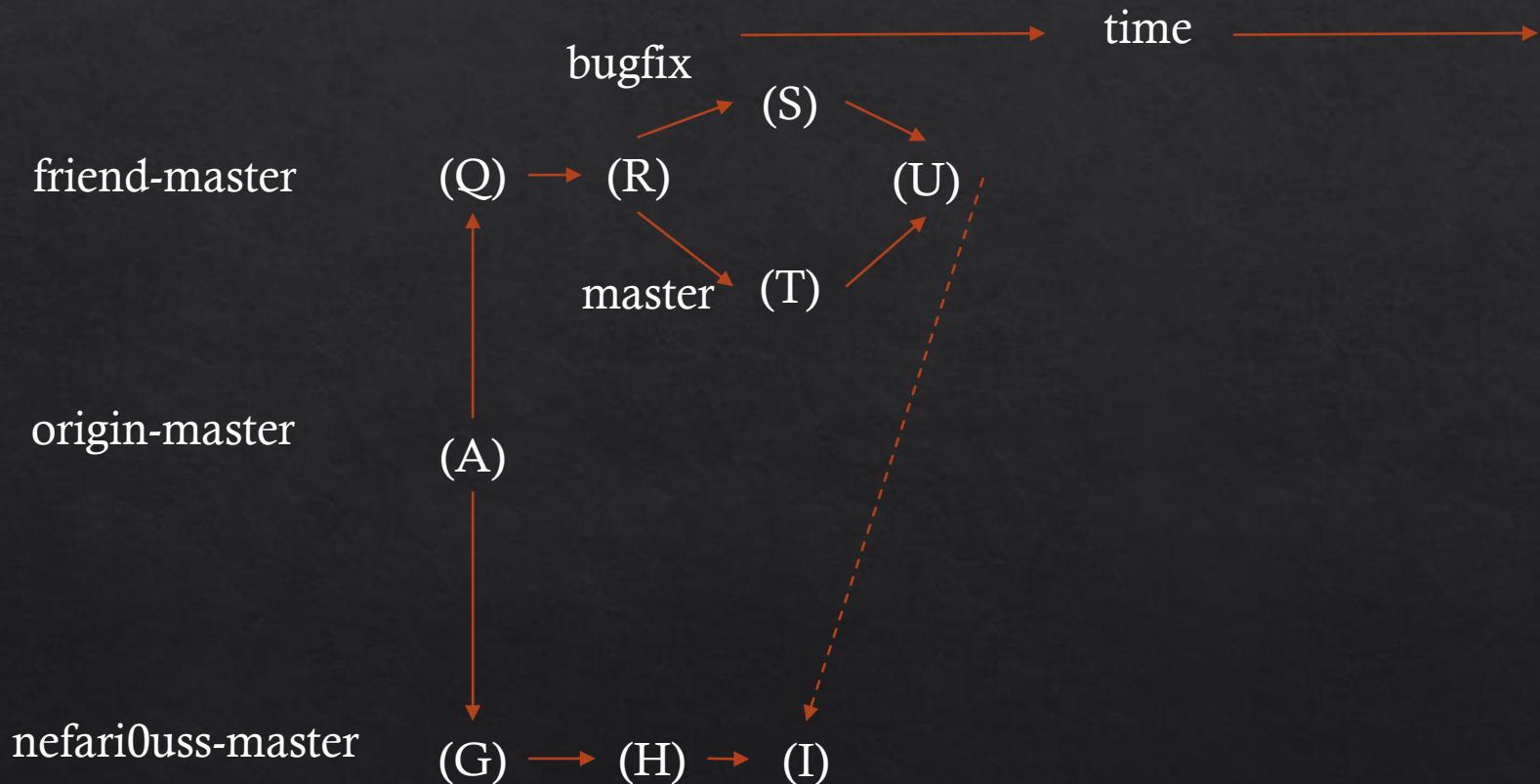
let me diagram this

remotes



friend, origin, and nefari0uss are all remote names

distributed



friend, origin, and nefari0uss are all remote names
master and bugfix are branch names

\$ git checkout [arguments]

two uses

\$ git checkout [arguments]

two uses

- 1) revert modifications to a file
\$ git checkout [filename.ext]

\$ git checkout [arguments]

two uses

- 1) revert modifications to a file
\$ git checkout [filename.ext]

- 2) change branches
\$ git checkout [branch name]

\$ git checkout [arguments]

two uses

- 1) revert modifications to a file
\$ git checkout [filename.ext]

- 2) change branches
\$ git checkout [branch name]

working directory must be clean when changing branches

\$ git log [arguments]

no arguments: show a log of commits

--graph: display tree

\$ git reset [arguments]

reset git state

```
$ git reset [arguments]
```

reset git state

-- soft: does not touch index or working tree
file is reverted to uncommitted (but changed) state

\$ git reset [arguments]

reset git state

-- soft: does not touch index or working tree
file is reverted to uncommitted (but changed) state

-- hard: resets index and working tree
file is to state of previous commit

intermediate commands

stash

blame

\$ git stash

want to save changes but not commit them

`$ git stash`

want to save changes but not commit them

store on a stack (not really)

\$ git stash

want to save changes but not commit them
store on a stack (not really)

(no args): push changes onto stack

list: show items on stash stack

pop [-index]: pop an item off the stack (uses index if provided)

`$ git blame [arguments]`

show who made the code base become borked

advanced commands

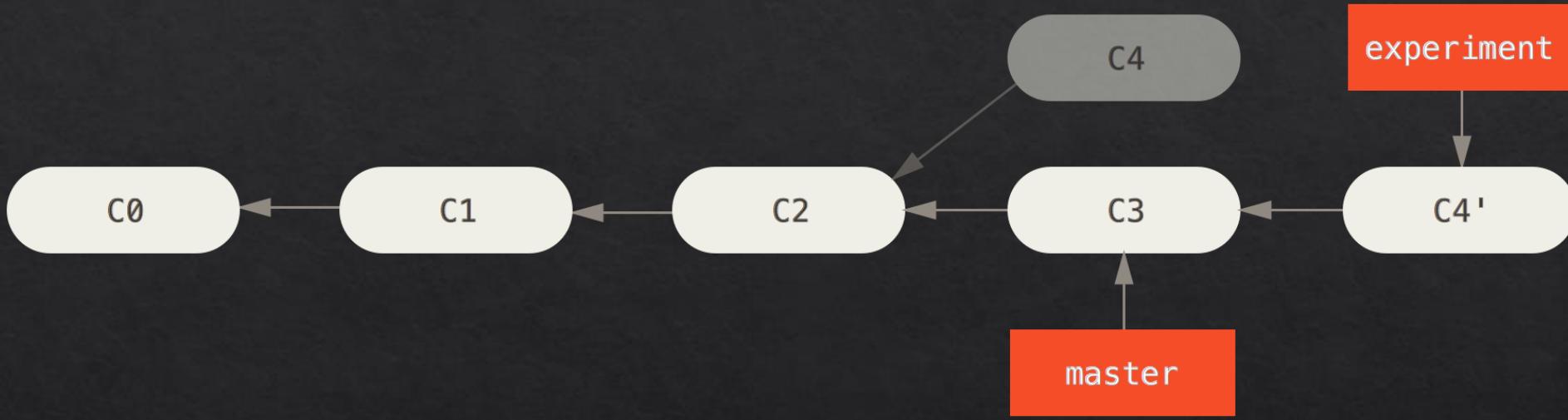
rebase

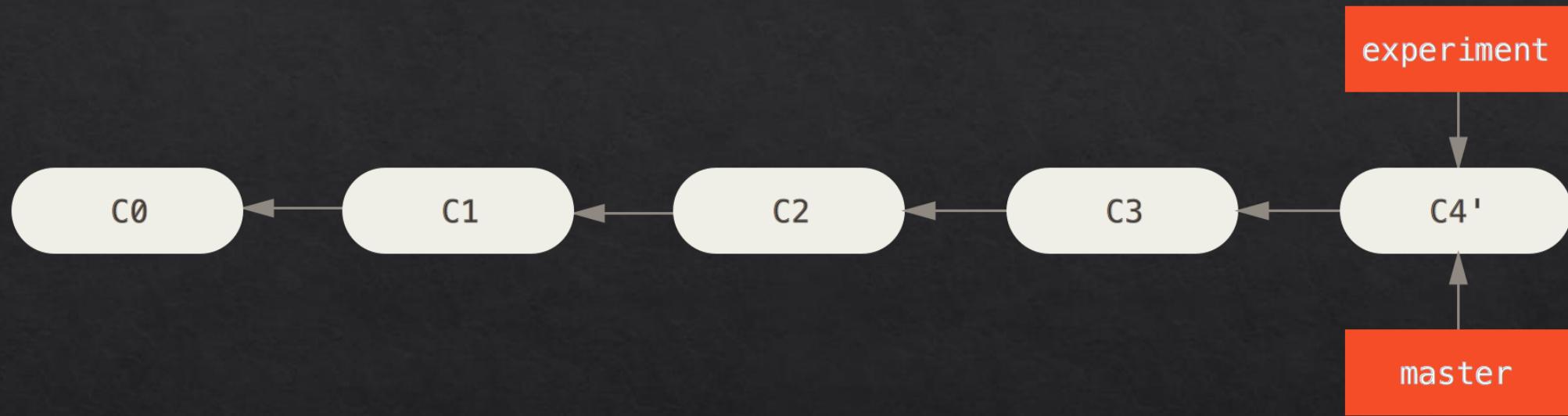
cherrypick

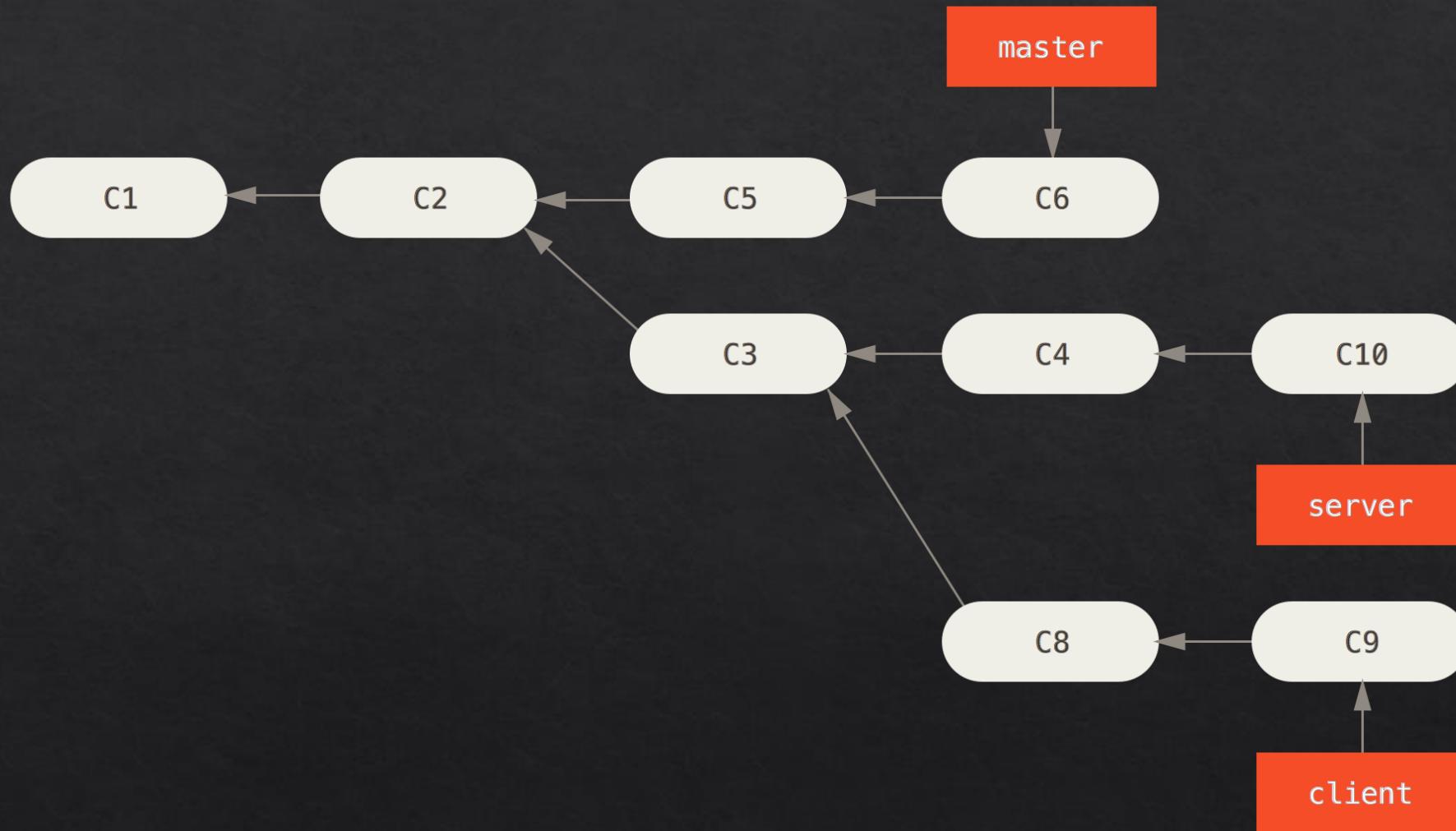
bisect

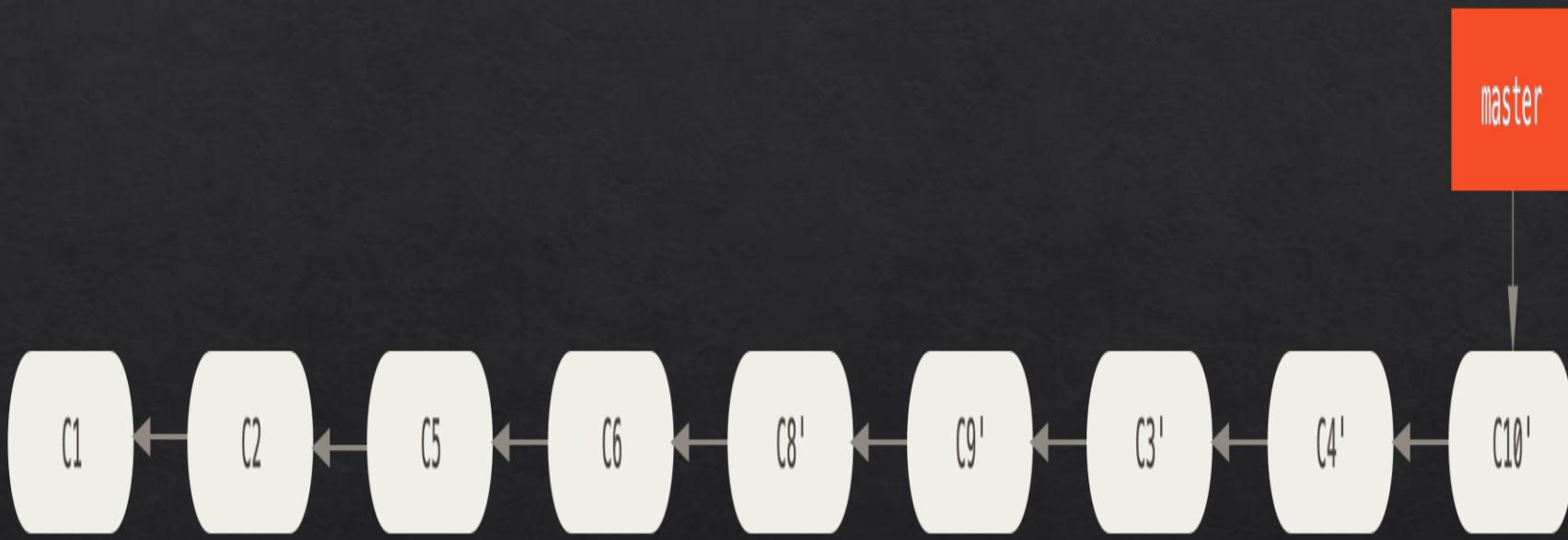
`$ git rebase [arguments]`

merge but create a single tree









`$ git cherry-pick`

pick the commits you want and merge only those

`$ git bisect`

like binary search for finding where the last good commit is

`$ git bisect`

like binary search for finding where the last good commit is

fin

very sorry this was so long

fin

very sorry this was so long
I hope you learned something!

Sources

- ❖ Git-SCM book: <https://git-scm.com/book>
 - ❖ Daniel “paradigm” Thau from Open Source Club at OSU on Git:
 - ❖ <https://opensource.osu.edu/au10/git>
 - ❖ git – the simple guide: <https://rogerdudler.github.io/git-guide/>
 - ❖ Atlassian Git Tutorial: <https://www.atlassian.com/git/tutorials/>
 - ❖ Understanding Git Conceptually: <http://www.sbf5.com/~cduan/technical/git/>
 - ❖ Wikiepda – Git (software): [https://en.wikipedia.org/wiki/Git_\(software\)](https://en.wikipedia.org/wiki/Git_(software))
 - ❖ Git Reference: <http://gitref.org/basic/>
-
- ❖ Lots of pictures from Git-SCM Book, Wikipedia and Google/Bing Image searches