Git General Training
Git Best practices
Git manage AI tips

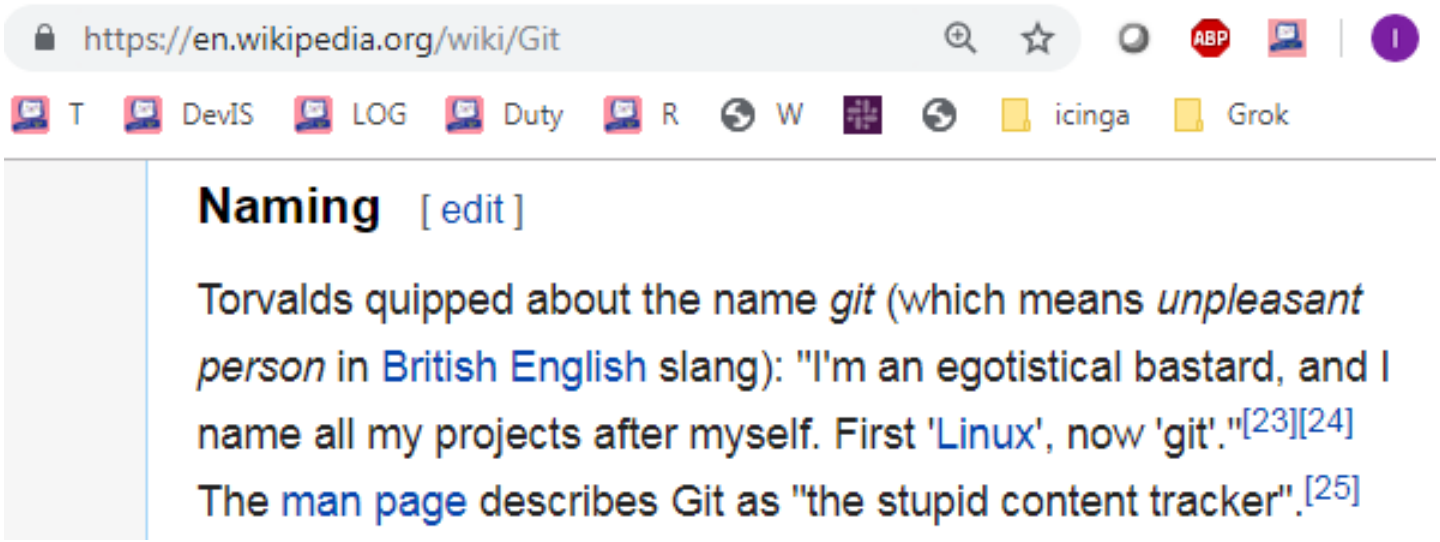
Deep inspection of basic commands

Ilya Rokhkin

Git – meaning?

What does the word Git mean?

GIT Overview



Agenda:

1. Basic concepts and commands
   - Git Architecture, data model
   - DVCS, Repository, Commit, Parent Commit, Tree, Blob, Index.
   - Staged, Modified, and Committed files.
   - Basic commands to work in the Repository and outside.
   - Sharing work with peers.
   - Best practices
   - How to manage AI changes in Git

2. Practical part, lab work

GIT Overview

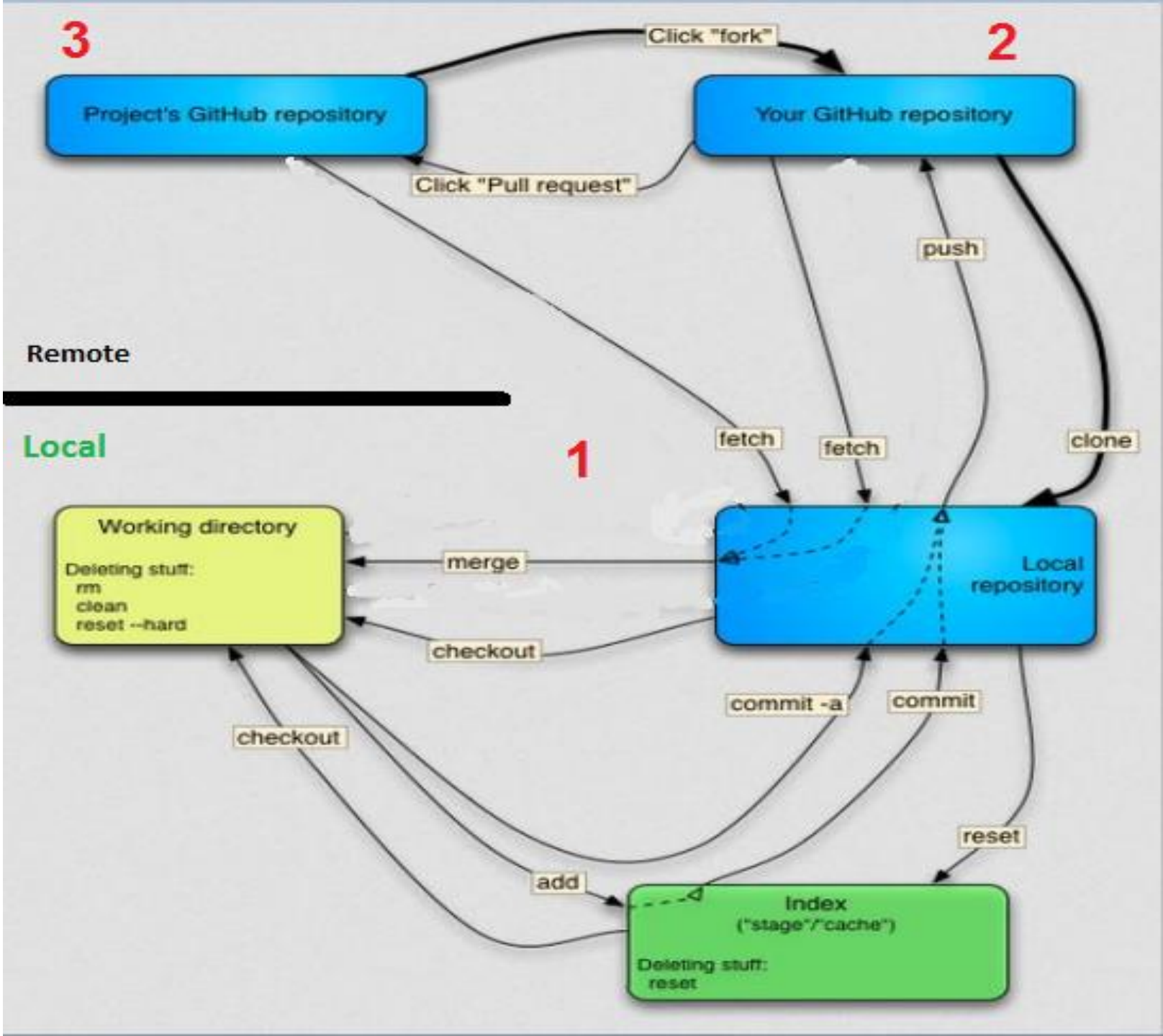
- Quick and efficient
- Expedite distributed development
- Atomic transactions, commit, cross repository
- Commits (Change) management
- A clear internal design
- Suited to handle everything from small to very large projects with speed and efficiency
- Support and encourage branched development

About myself:

+ Author: Ilya Rokhkin
+ Role: DevOps Engineer @ CloudOps Group
- Role: Harmony Connect DevOps
+
+ Experience:
+ 20+ years in VCS
+ Git trainer (CHKP, Freelancer)
- (Intel, Marvell)
+ Hebrew teacher volunteer @ Ulpan

GIT Architecture

1 Local repo
2 Remote (Origin)
3 Common (Community) Remote Origin



Demo
Lab 1 - 2

- We will configure your git user and e-mail

  Will create a remote, bare repository in the home directory

  Clone it to work repository1 in the home directory also, add, commit, and push
- Restructure files, add, commit, and push

Git repository structure (.git)

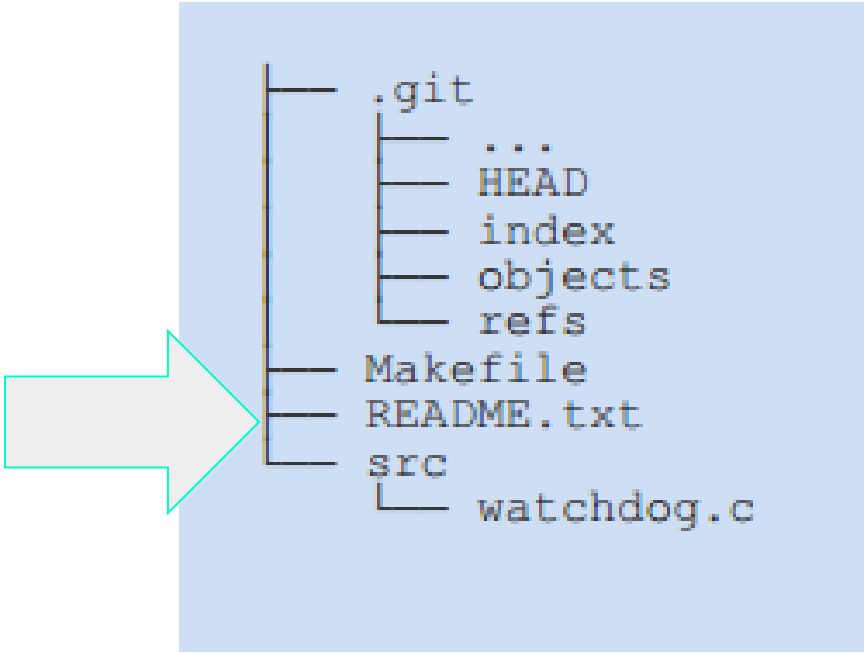
```
$ ls -al
total 11
drwxr-xr-x   7 sheta   Administ   4096 Dec  3 15:17 .
drwxr-xr-x   3 sheta   Administ   4096 Nov 30 11:26 ..
-rw-r--r--   1 sheta   Administ     23 Nov 30 11:09 HEAD
-rw-r--r--   1 sheta   Administ    363 Nov 30 11:46 config
-rw-r--r--   1 sheta   Administ     73 Nov 29 17:03 description
drwxr-xr-x   2 sheta   Administ   4096 Nov 29 17:03 hooks
-rw-r--r--   1 sheta   Administ     32 Nov 30 11:26 index
drwxr-xr-x   2 sheta   Administ      0 Nov 29 17:03 info
drwxr-xr-x   3 sheta   Administ      0 Nov 29 17:03 logs
drwxr-xr-x  25 sheta   Administ   4096 Nov 30 11:08 objects
-rw-r--r--   1 sheta   Administ     94 Nov 29 17:03 packed-refs
drwxr-xr-x   5 sheta   Administ      0 Nov 30 11:07 refs
```


```
.git
    ...
    HEAD          This file holds a reference to the branch (or commit) you currently have checked out
    config        This is the main Git configuration file. It keeps specific Git options for your project
    description   It will show when you have viewed your repository or the list of all versioned repositories.
    hooks         Directory contains shell scripts that are invoked after the corresponding Git commands
    index         The Git index is used as a staging area between your working directory and your repository
    info          Contains additional information about the repository.
    logs          Keeps records of changes made to refs.
    objects       In this directory the data of your Git objects are stored – all the contents of the files you have ever checked in
    packed-refs   The file consists of the tips of all branches and tags.
    refs          This directory normally contains the tips of branches and tags in  three subfolders – heads, remotes and tags.
Makefile
README.txt
src
    watchdog.c
```

The Working Tree

- The working tree has all files and folders as found in your HEAD, plus the changes you made since your last commit
- There is only ONE main working tree per repository (and only 1 .git folder as well)

The Working Tree



States of files in Working Tree

- **Untracked** – in the repository folder, git does not keep a version of it.
- **Modified** – tracked, modified since last stage or commit.
- **Staged** – a snapshot of the file, ready to be committed. Even if modified, git will still keep the snapshot.
- **Committed** – version of file saved in repository DB

Objects

Every object in GIT composed of those elements –

Type – “blob”, “tree”, “commit”, “refs” (branch/tag).

A "blob" is basically like a file – it is used to store the content of a source file.

A "tree" is basically like a directory - it references a group of other trees (subdirectories) and/or blobs (files).

A "commit" points to a single tree, marking it as what the project looked like at a certain point in time. Keeps changed files since the last commit, author of the changes, a reference to the parent commit(s), etc.

A “refs" is a way to mark a specific commit as special in some way. It is usually used to tag certain commits as specific releases or something along those lines.

Objects cont.

Almost all the GIT is built around manipulating this simple structure of four different object types. It is sort of it's own little file system that sits on top of your machine's file system.

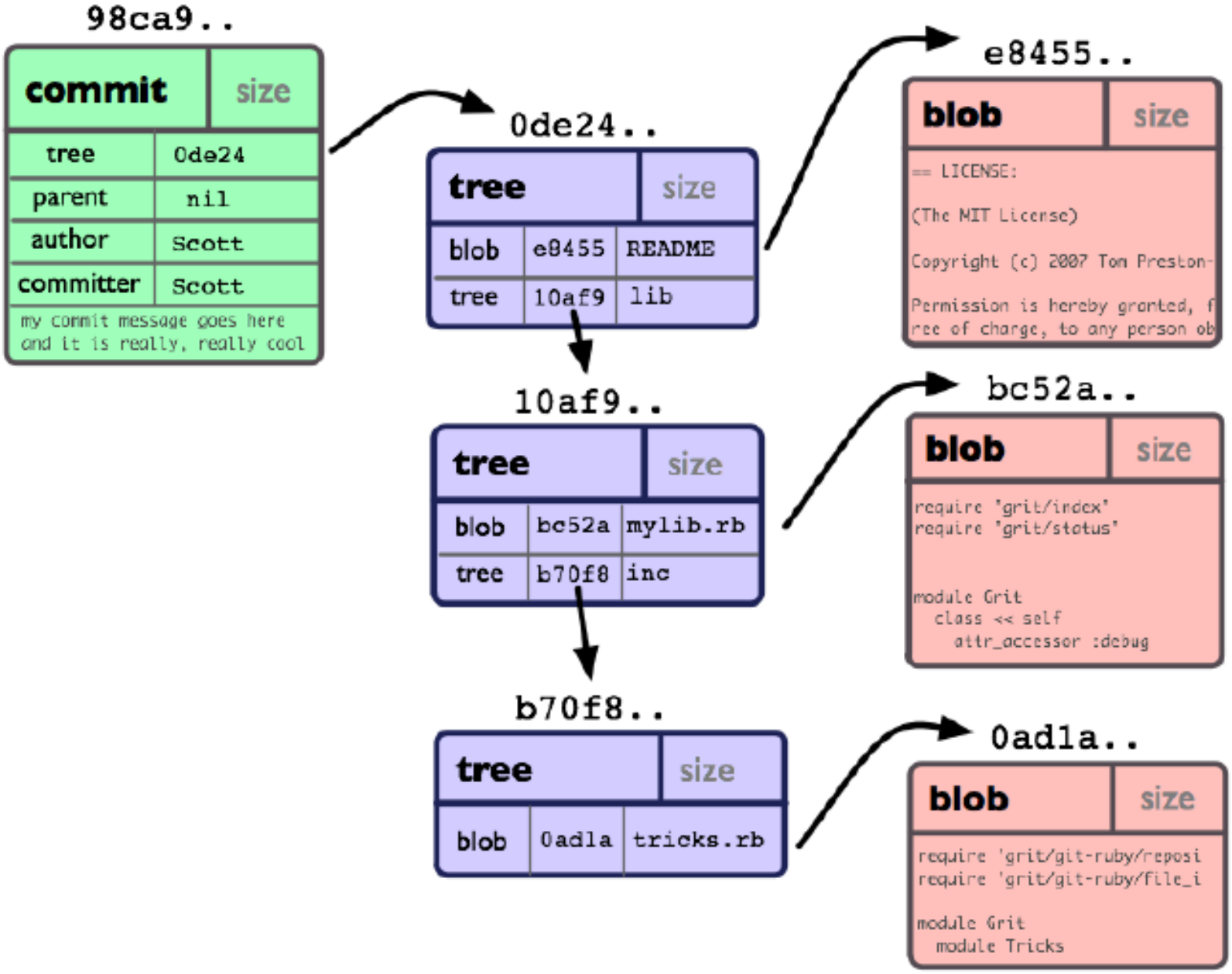
Let's say we have a small project that looks like this:

```
$>tree
.
|-- README
`-- lib
    |-- inc
    |   `-- tricks.rb
    `-- mylib.rb

2 directories, 3 files
```

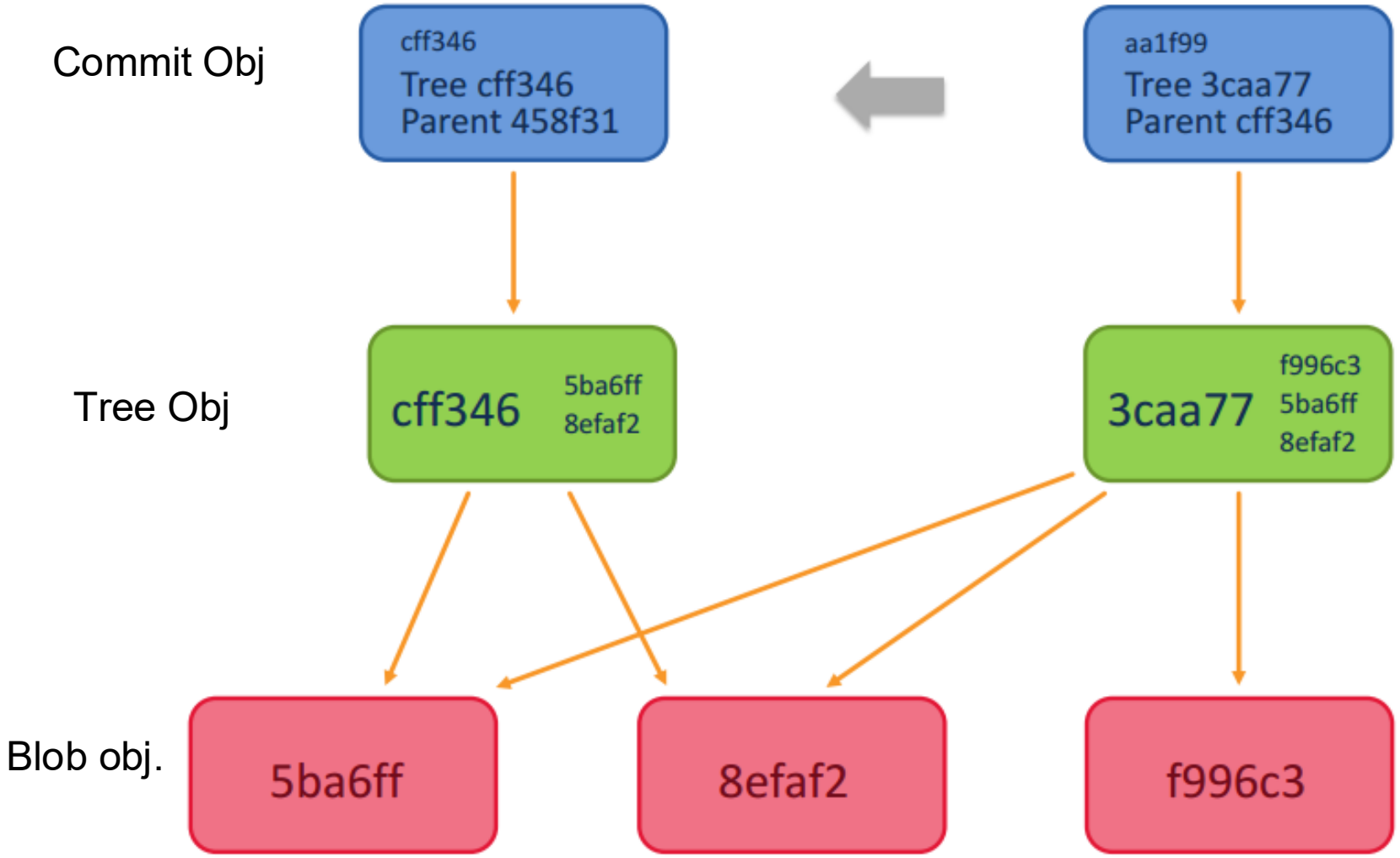
If we will commit this project to a GIT repository, it will be represented in GIT like this:

Commit Object



Commit object with its parent

- Links from tree object to common blobs



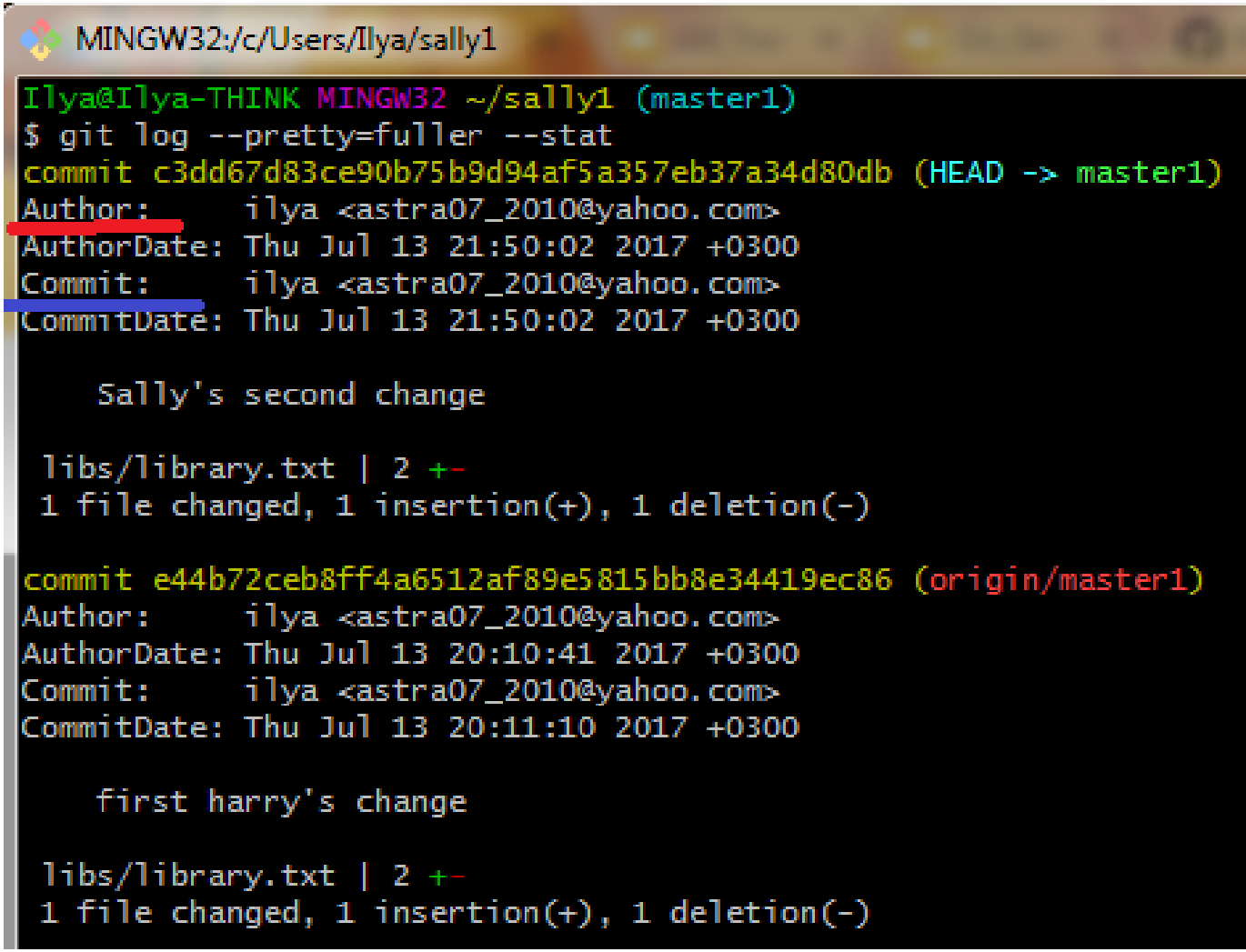
Secure Hash Algorithm – SHA1

- Each object in Git is represented by a 40-digit string that resembles this: **7bf68ebf3d8cff042bd3cb87e7592ddda9caa665**.
  This string is being calculated by taking the SHA1 hash of the contents of the object.

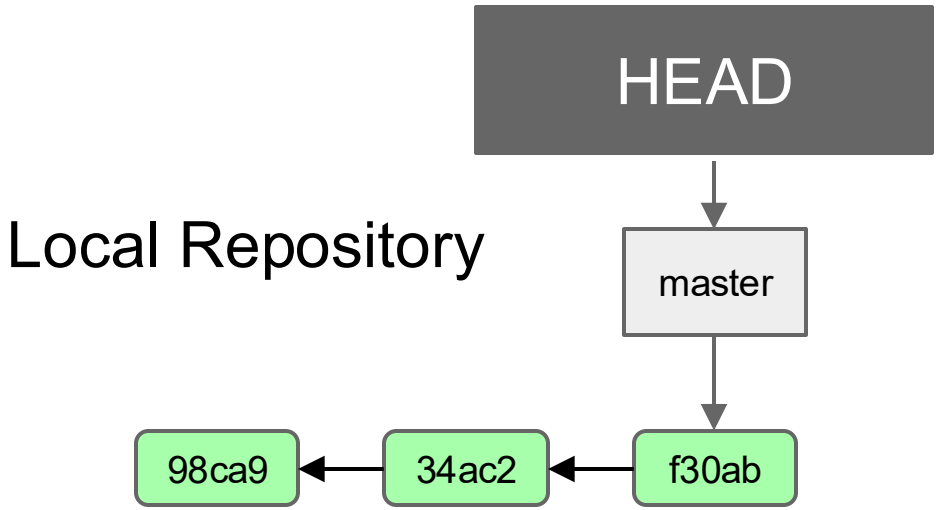
Each commit has **Author** and **Committer**

The Author is who really wrote the code, and committed

Committer, if not the same as the Author, took Original commit and reused it in his branch



The HEAD



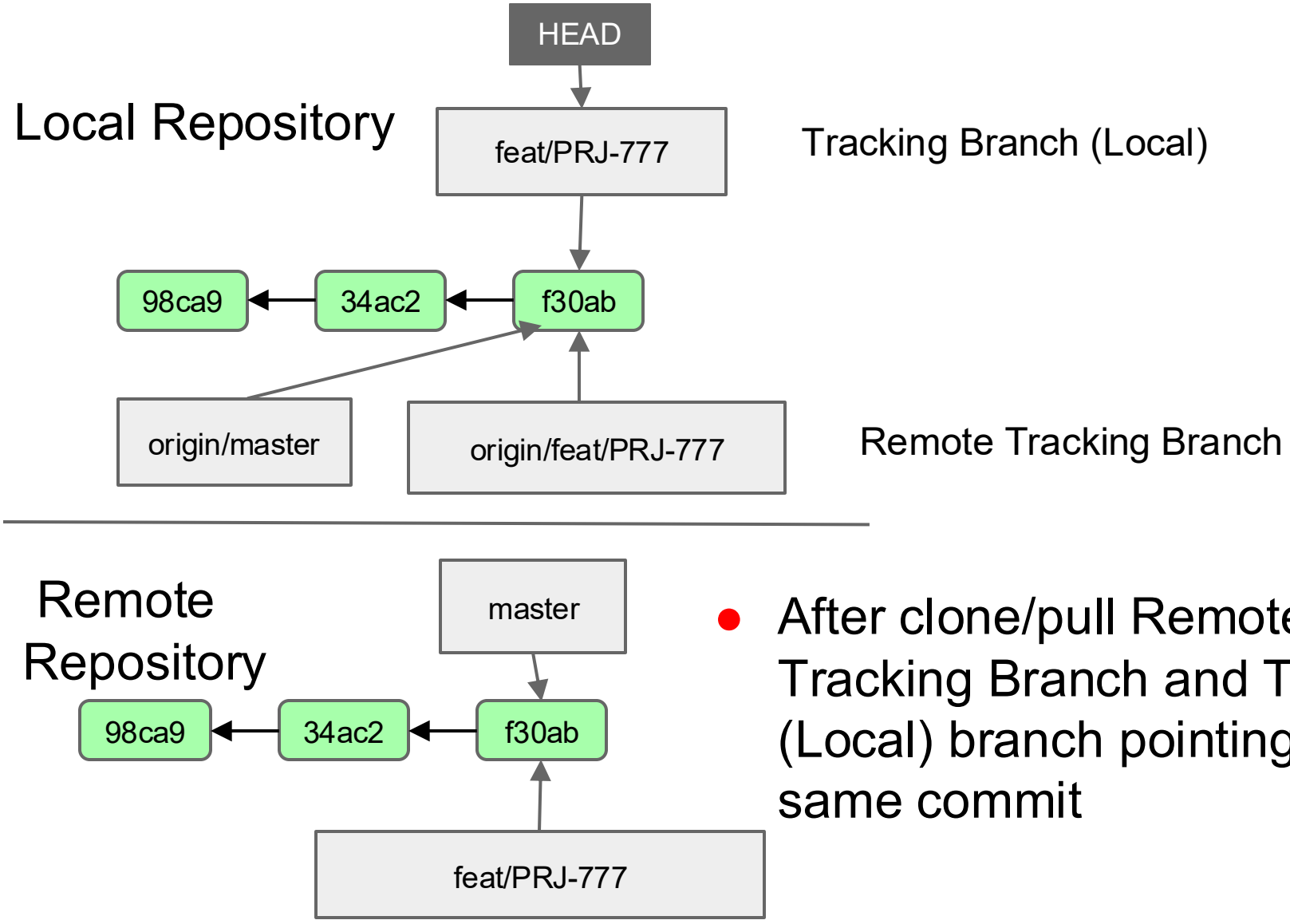
- `HEAD` is a 'pointer' to the tip of the currently checked out branch
  - In a *detached HEAD* state, `HEAD` points directly to a commit
- Only one `HEAD` per repository

Demo
Lab 3

Teamwork, parallel work:

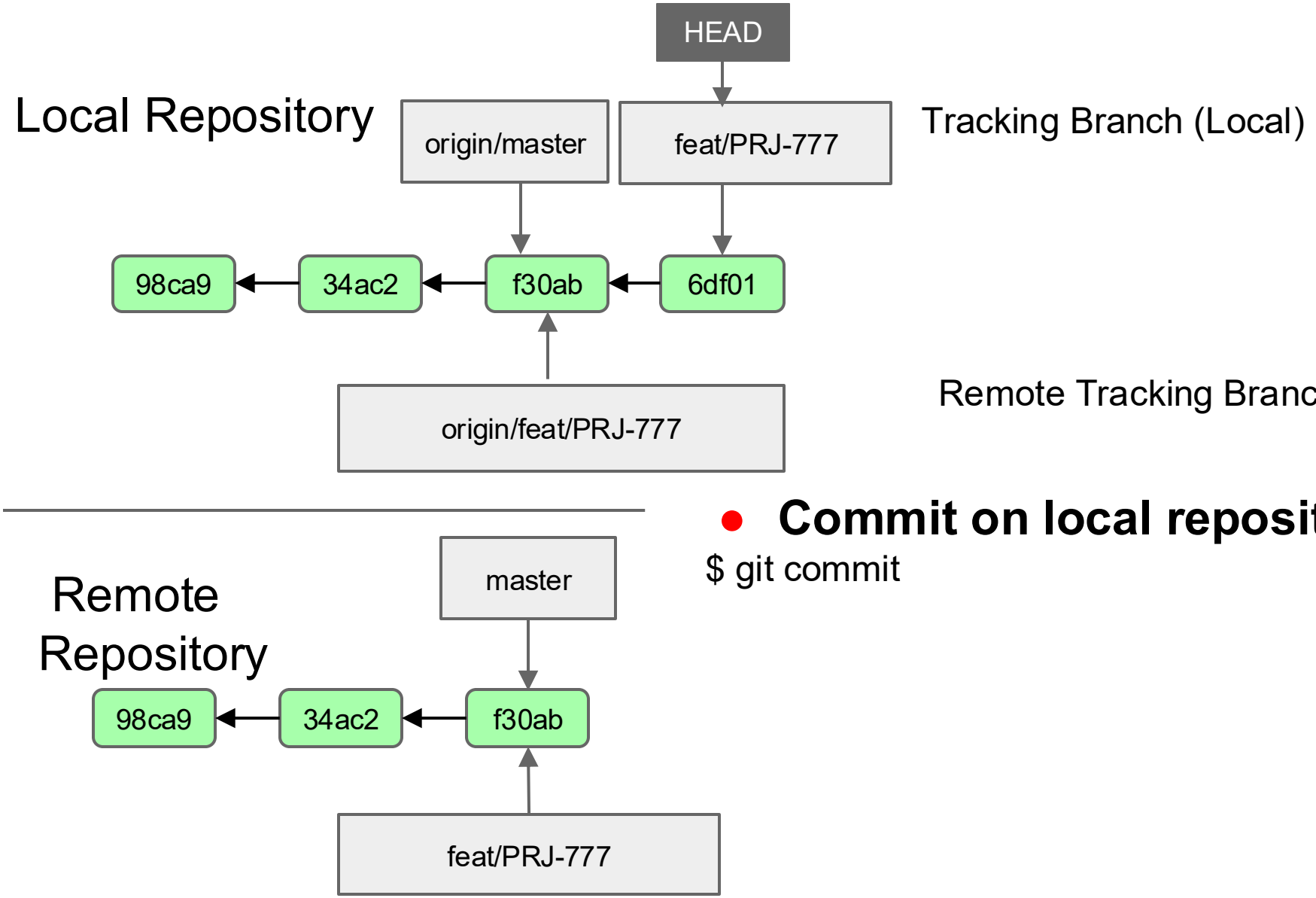
- We will clone the second work repository2
- We will commit changes in both repositories
- Push and pull with silent rebase to apply the commit of one repo into another
- Overview results

Rebasing Remote Tracking Branch



- After clone/pull Remote Tracking Branch and Tracking (Local) branch pointing to the same commit

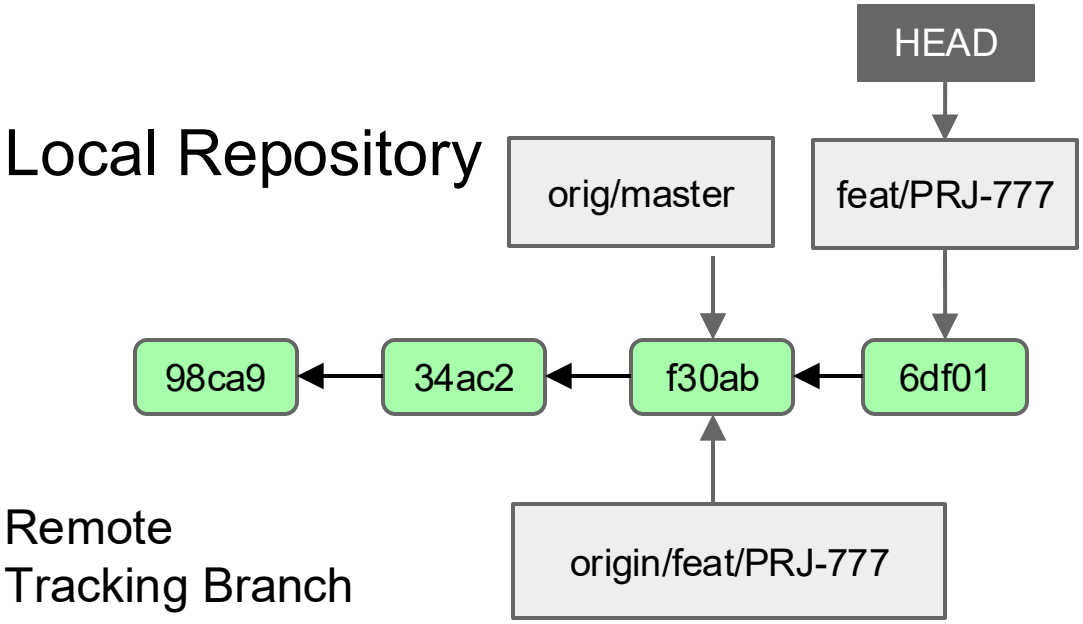
Rebasing Remote Tracking Branch

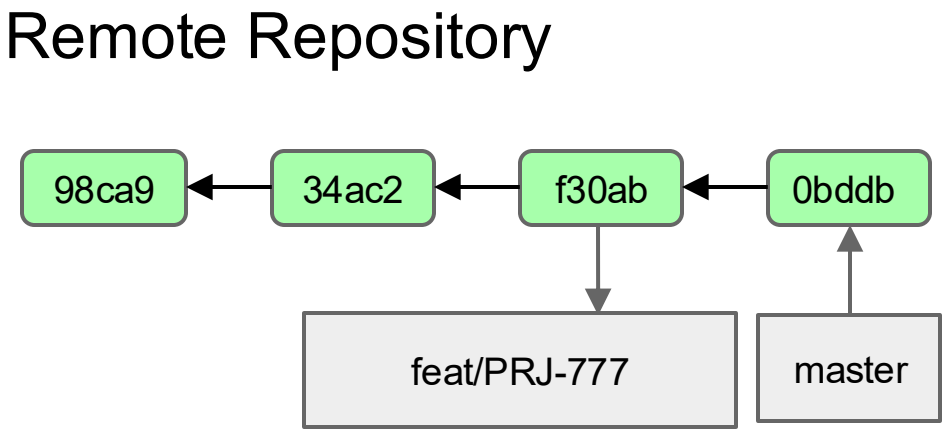


- **Commit on local repository**

$ git commit

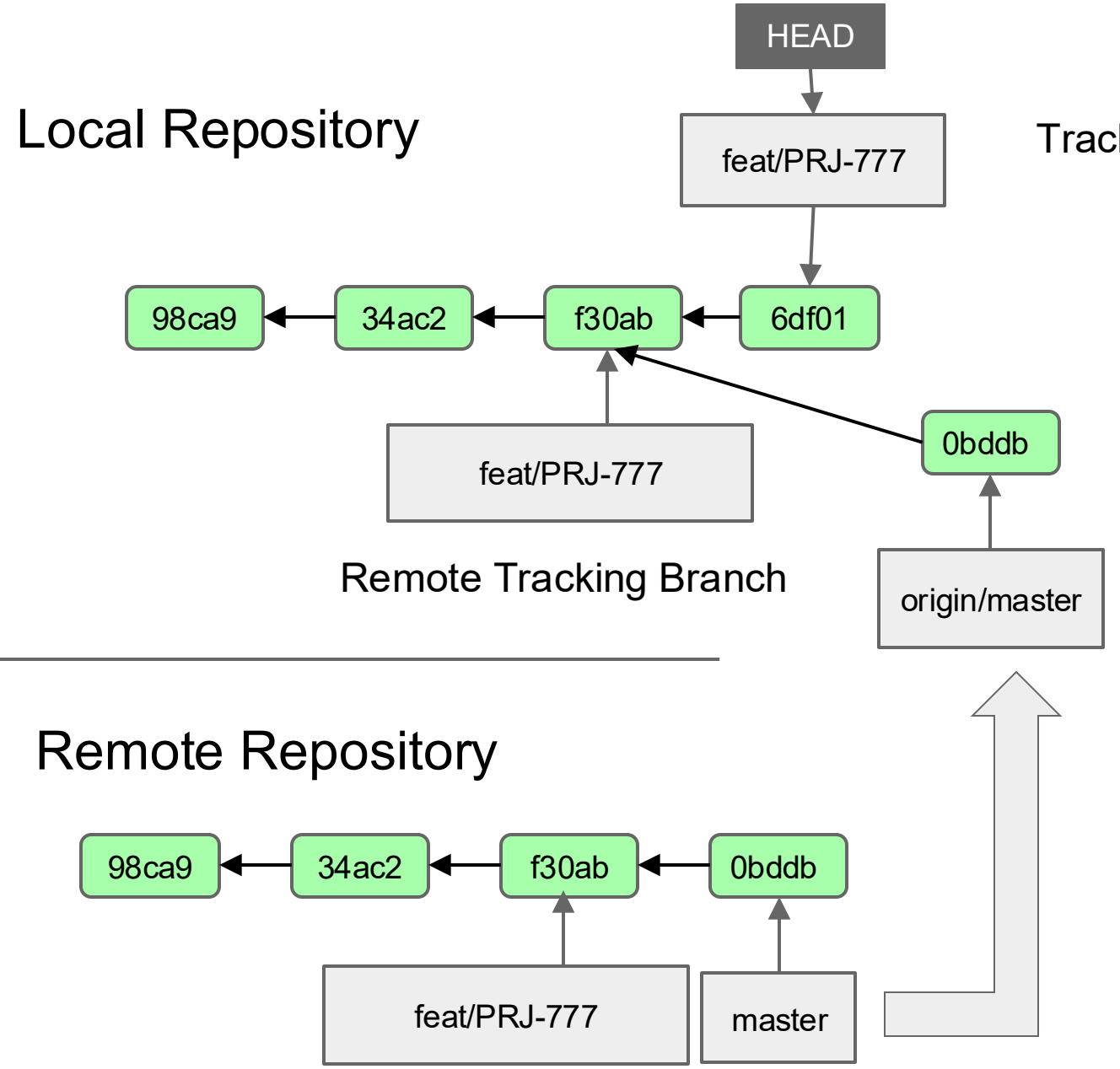
Rebasing Remote Tracking Branch





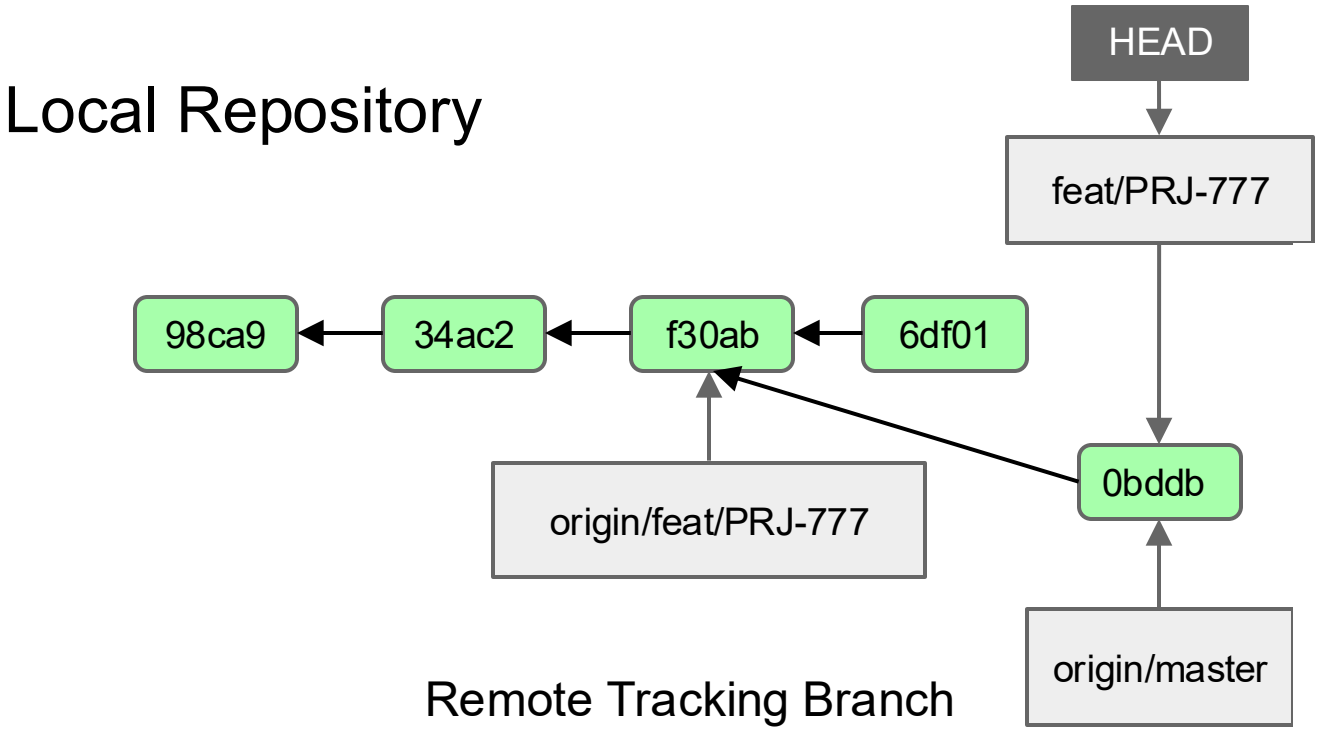
- **Merge another Commit to remote repository from another feature branch**

Rebasing Remote Tracking Branch



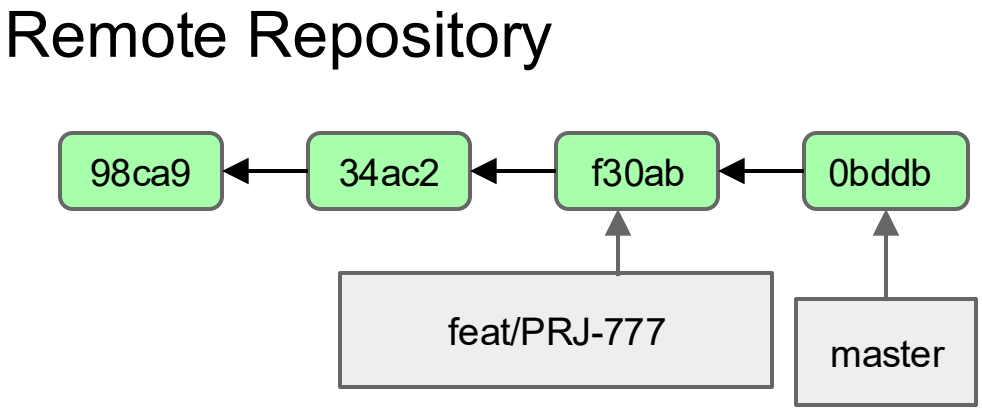
- **Fetch/Pull the Commit from remote repository to Remote Tracking Branch**

Rebasing origin/master



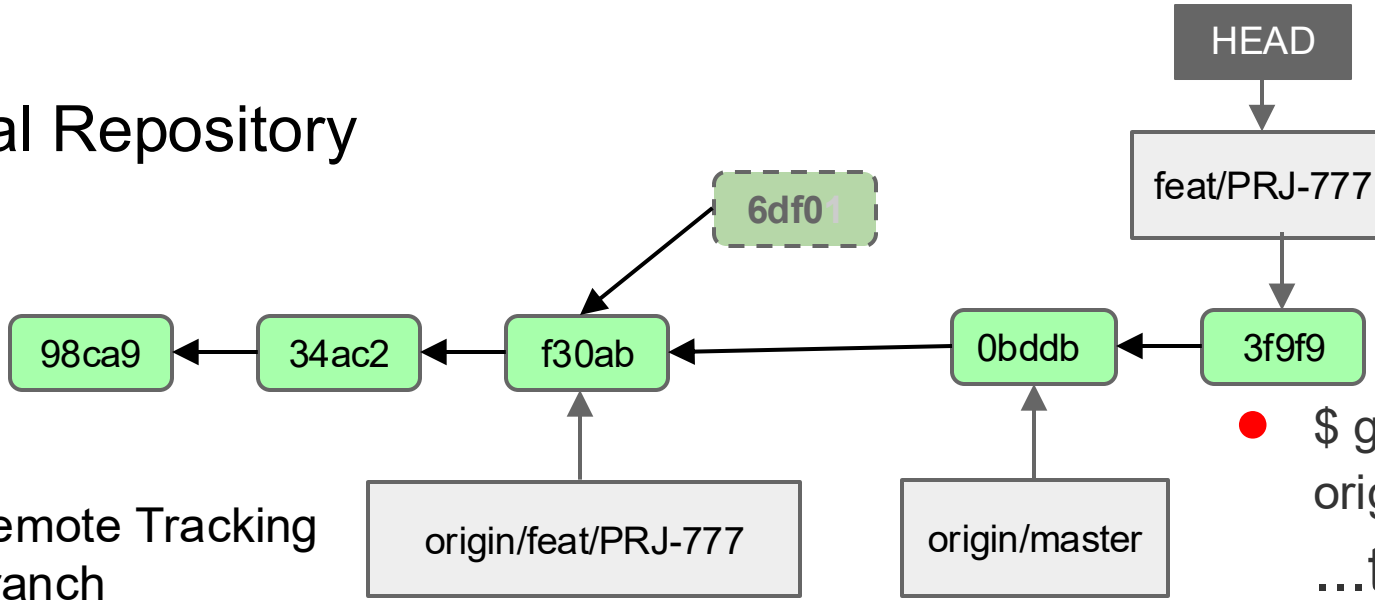
- $ git rebase origin/master

First, rewinding head to replay your work on top of it...



Rebasing Remote Tracking Branch

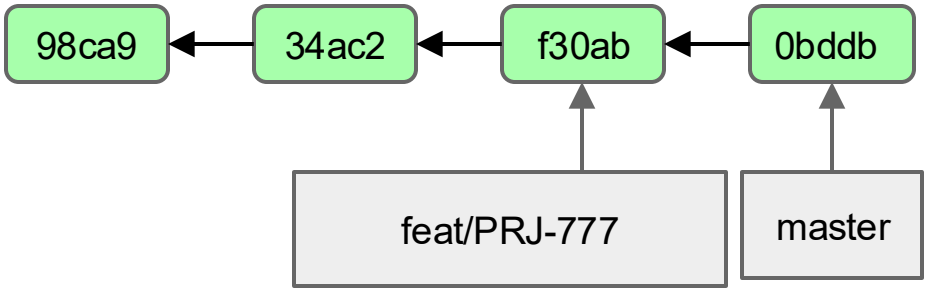
Local Repository



- $ git rebase origin/master

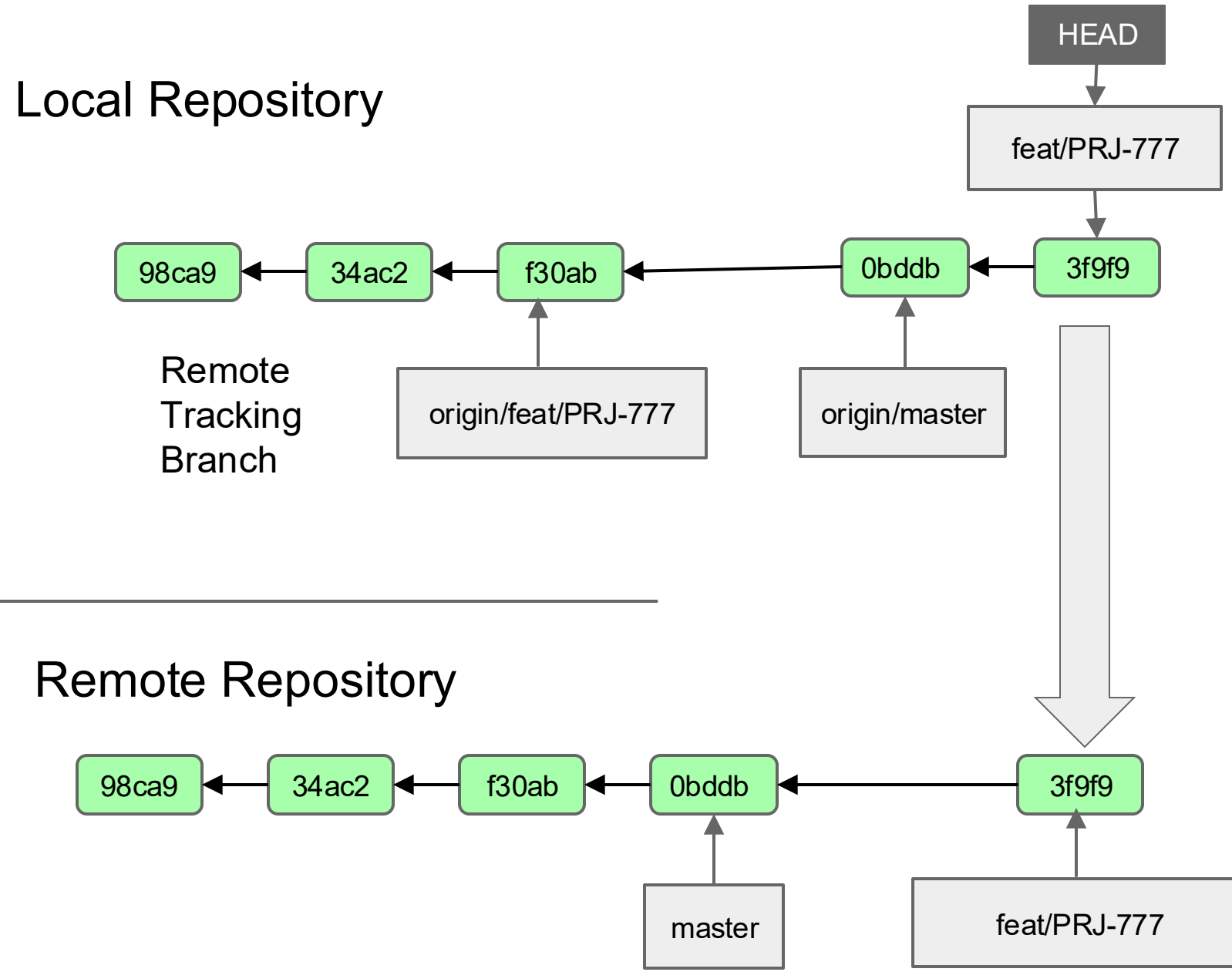
...to replay your work on top of it…

Remote Repository



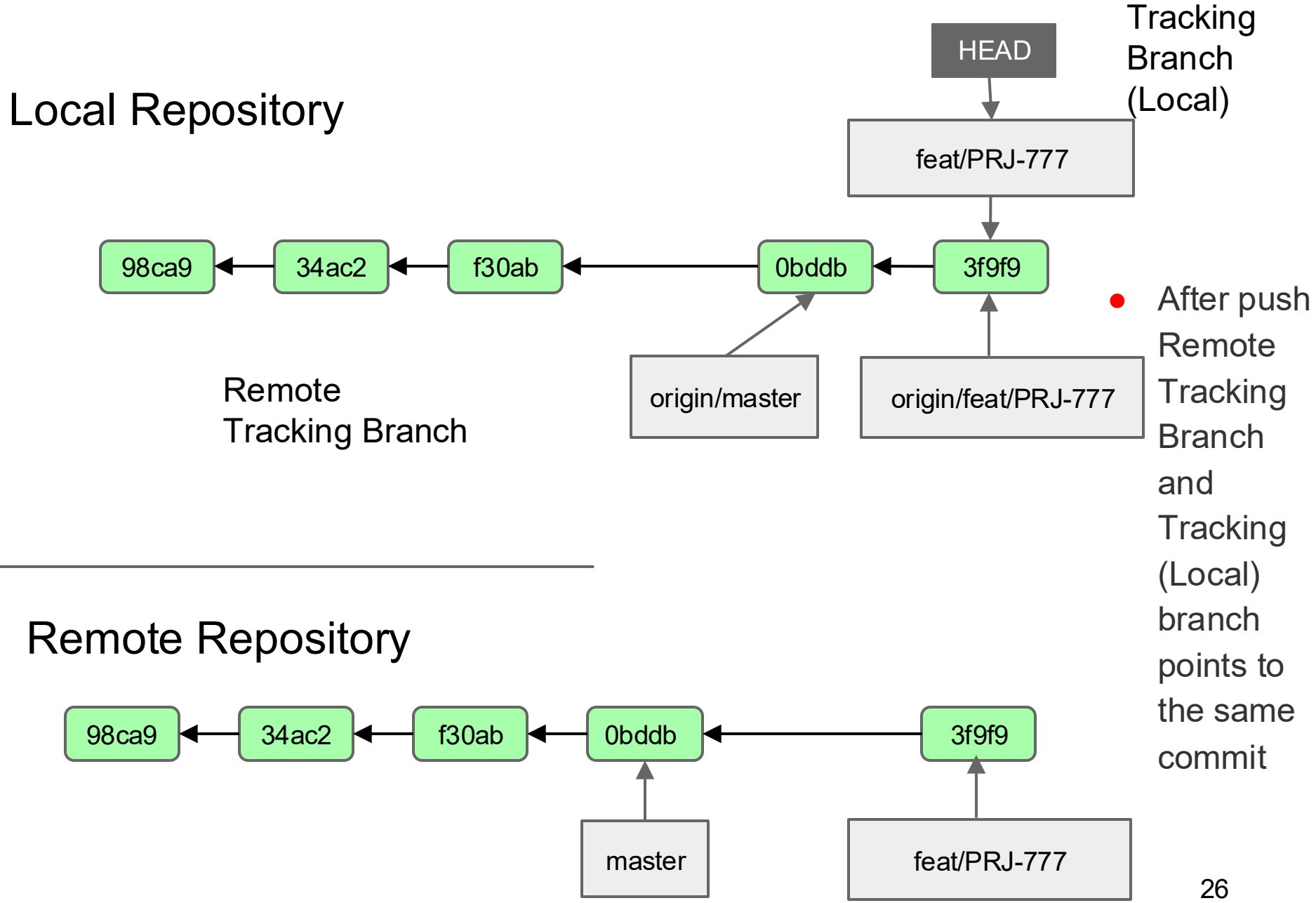
Applying: diff/patch of commit 6dfo on top of origin/master to create commit 3f9f9

Rebasing Remote Tracking Branch



- $ git push of commit 3f9f9

Rebasing Remote Tracking Branch



- After push Remote Tracking Branch and Tracking (Local) branch points to the same commit

Perils of rebase

- **Do not rebase commits that you have pushed to a public repository.**

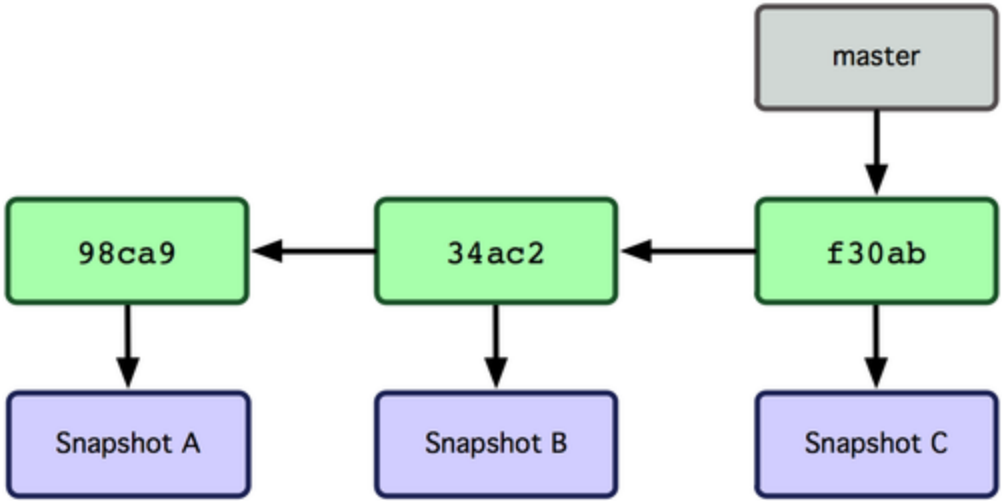
If you follow that guideline, you'll be fine. If you don't, people will hate you, and you'll be scorned by friends and family.

Demo
Lab 4

- Teamwork, parallel work with rebase, and conflict resolution:
- We will do commits in both repositories, with changes in the same line of the same file
- Pull with rebase, resolve conflicts, and save the resolution file
- Adding the file to the staging area means resolving the issue.
- Commit and push

Git branches

- Git branch is simply a movable pointer to a commit

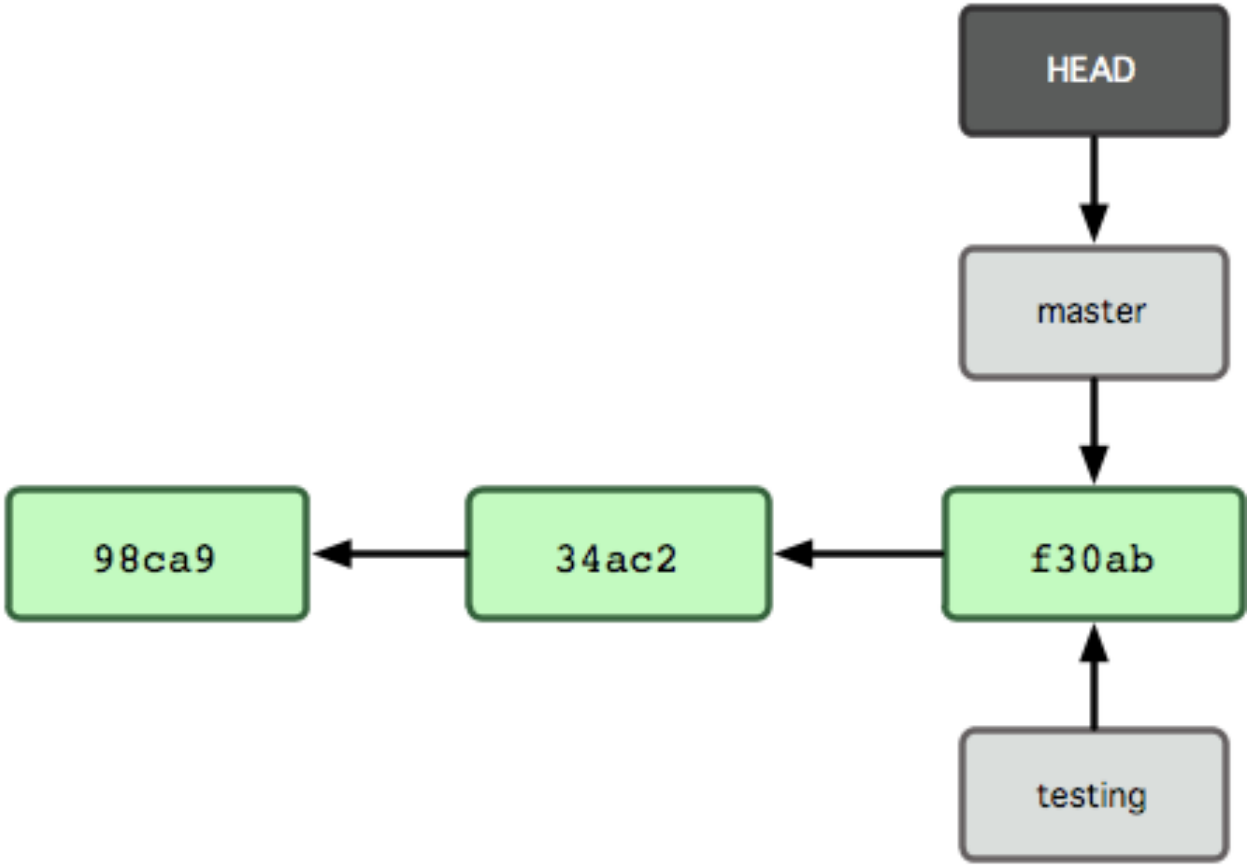


- Pointer moves forward automatically with each commit on a branch

Creating new branch

- New branch creates a new reference

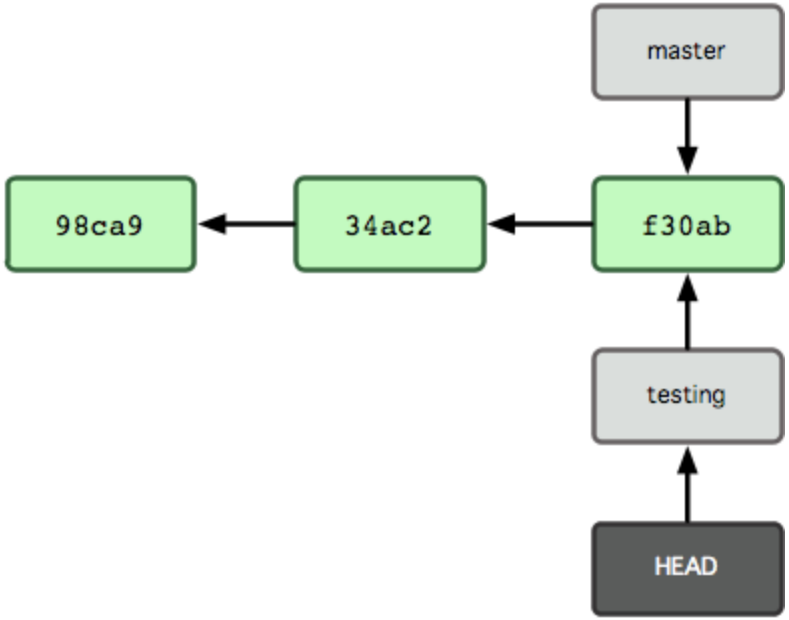
```
> git branch testing
```



Switching to a branch

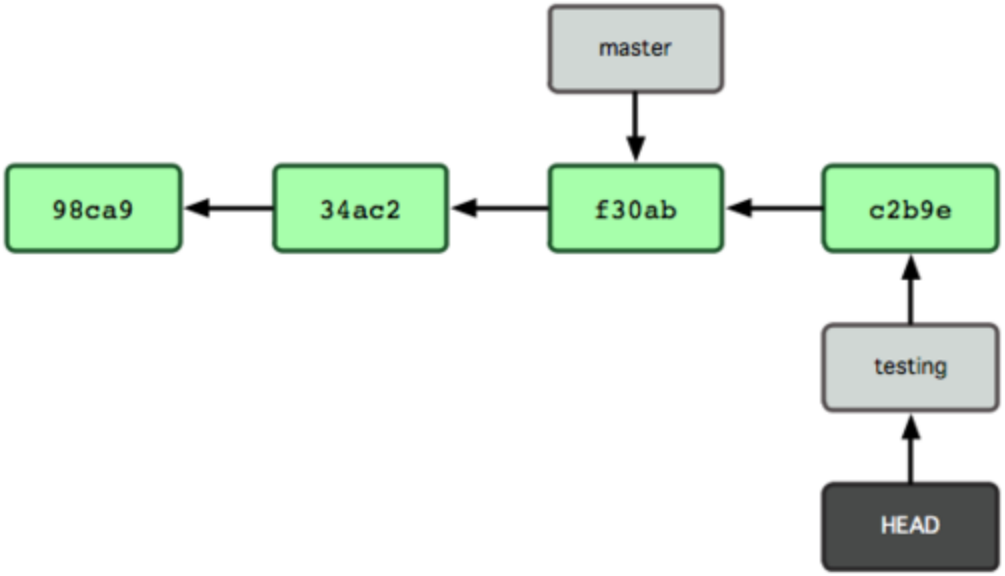
- Git checkout *branch-name* switches to an existing branch

```
> git checkout testing
```

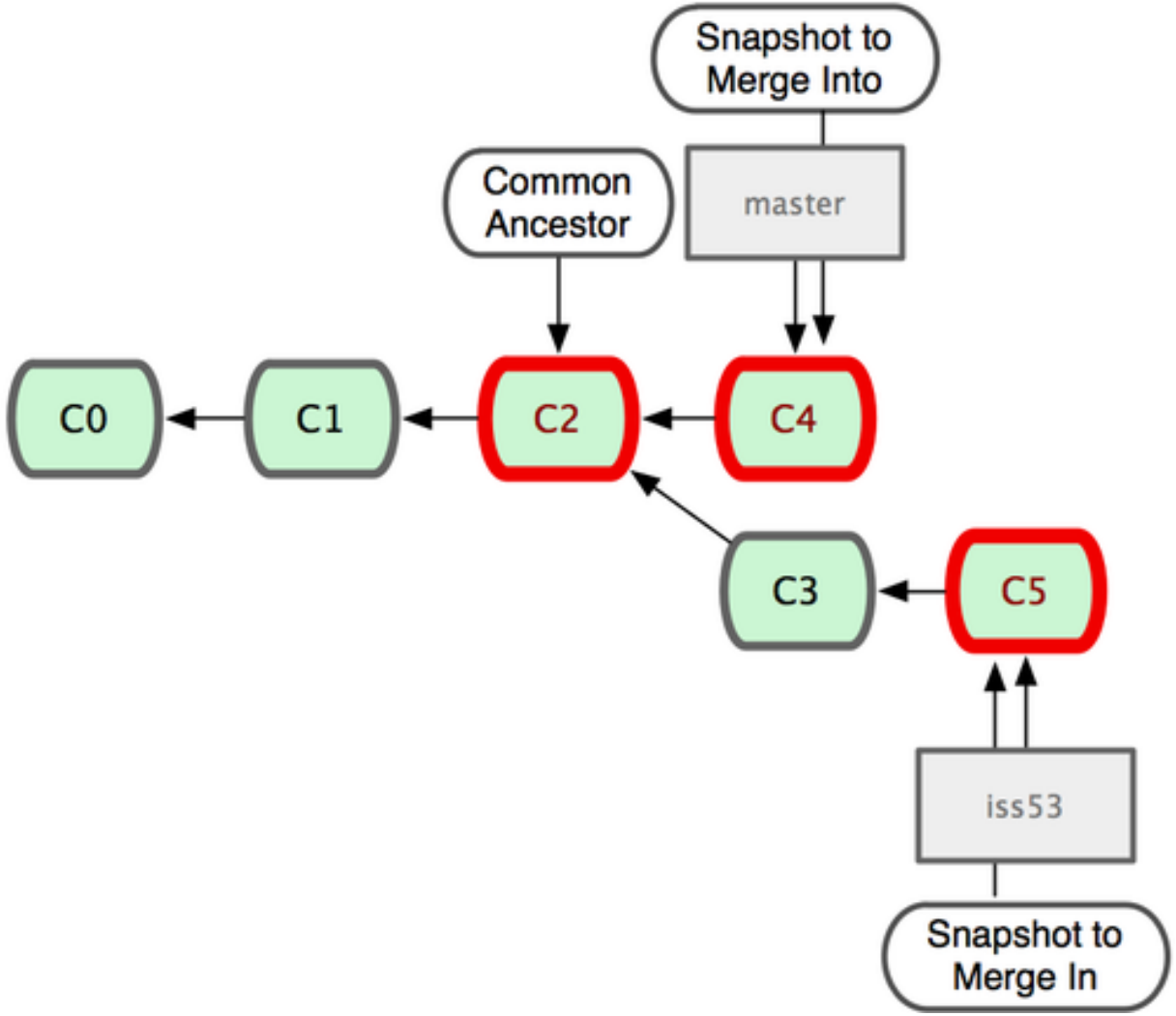


New commit moves current branch

```
> vi file04.txt
> git commit -a -m 'Commit message'
```

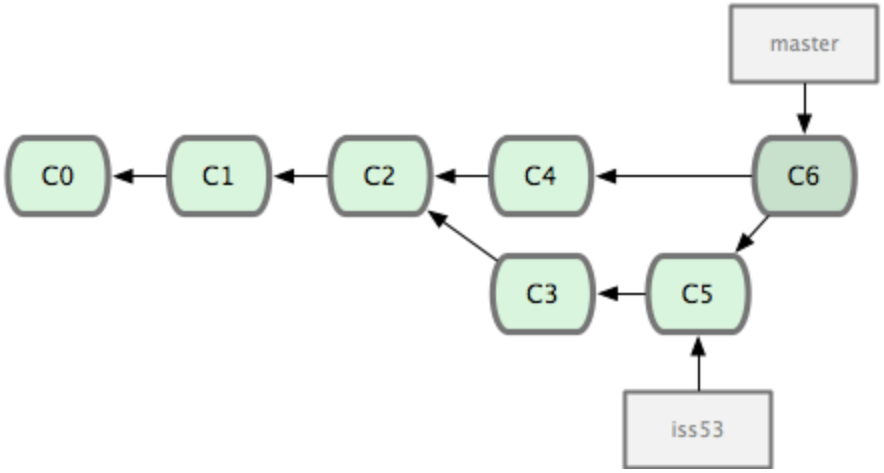


Merging branches



Merging branches (continued)

- As a result of merge Git creates a new commit, which has two parents:



Demo
Lab 5 - 7

- We will create a tag
- Create branch bugfix from the tag

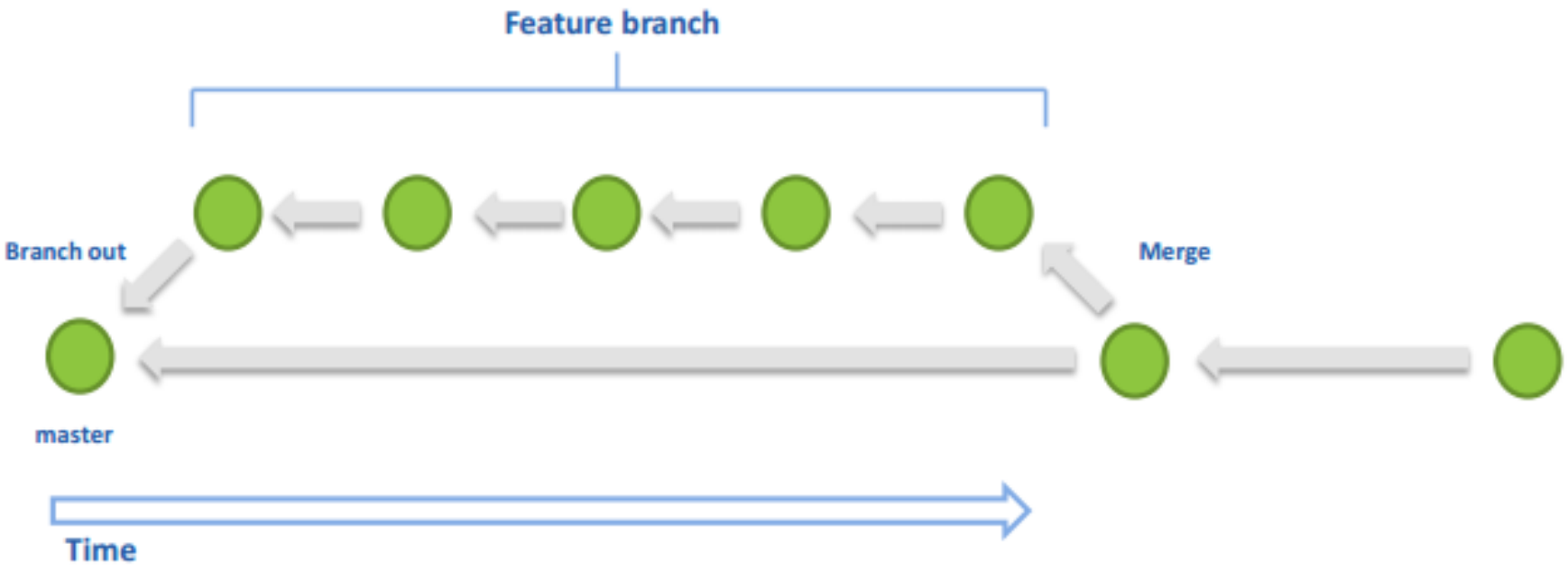
  I will do a bugfix commit in the bugfix branch

  Check out the master branch and commit a new change
- Merge bugfix branch

  Overview results

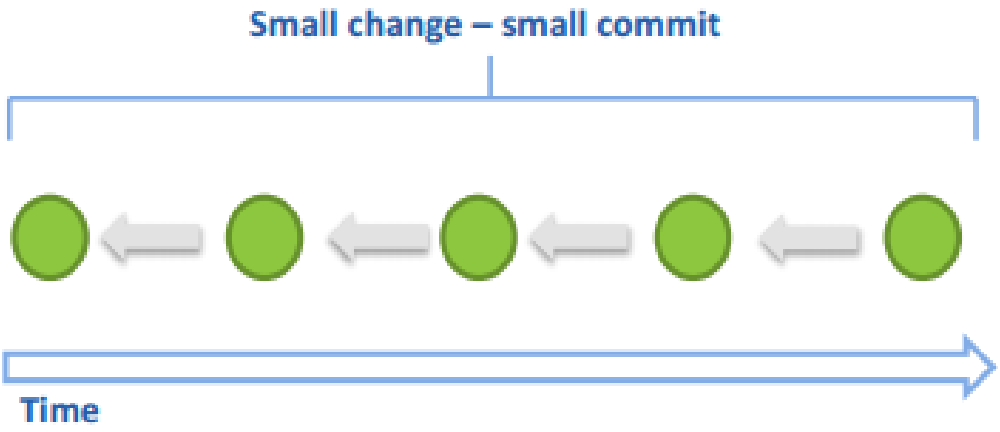
Best practices – local feature branches

Work on feature branches locally



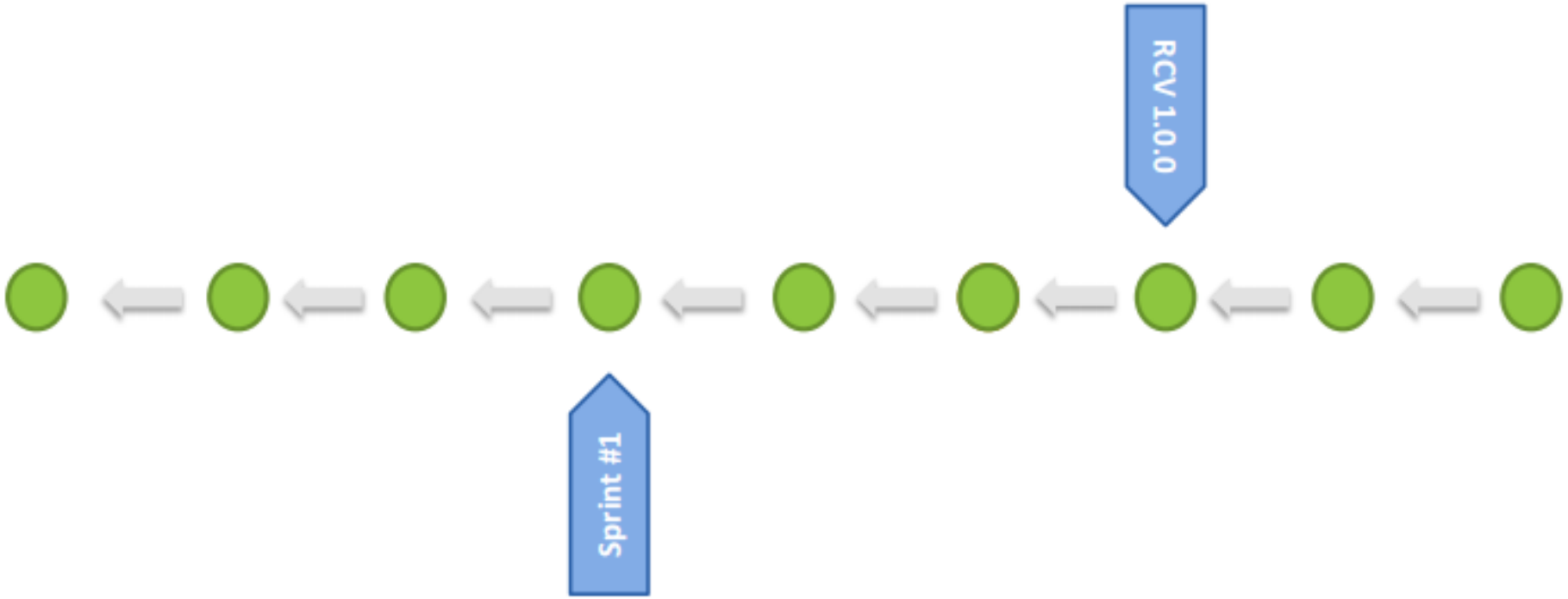
Best practices – commit

Keep changes small wherever possible and commit frequently



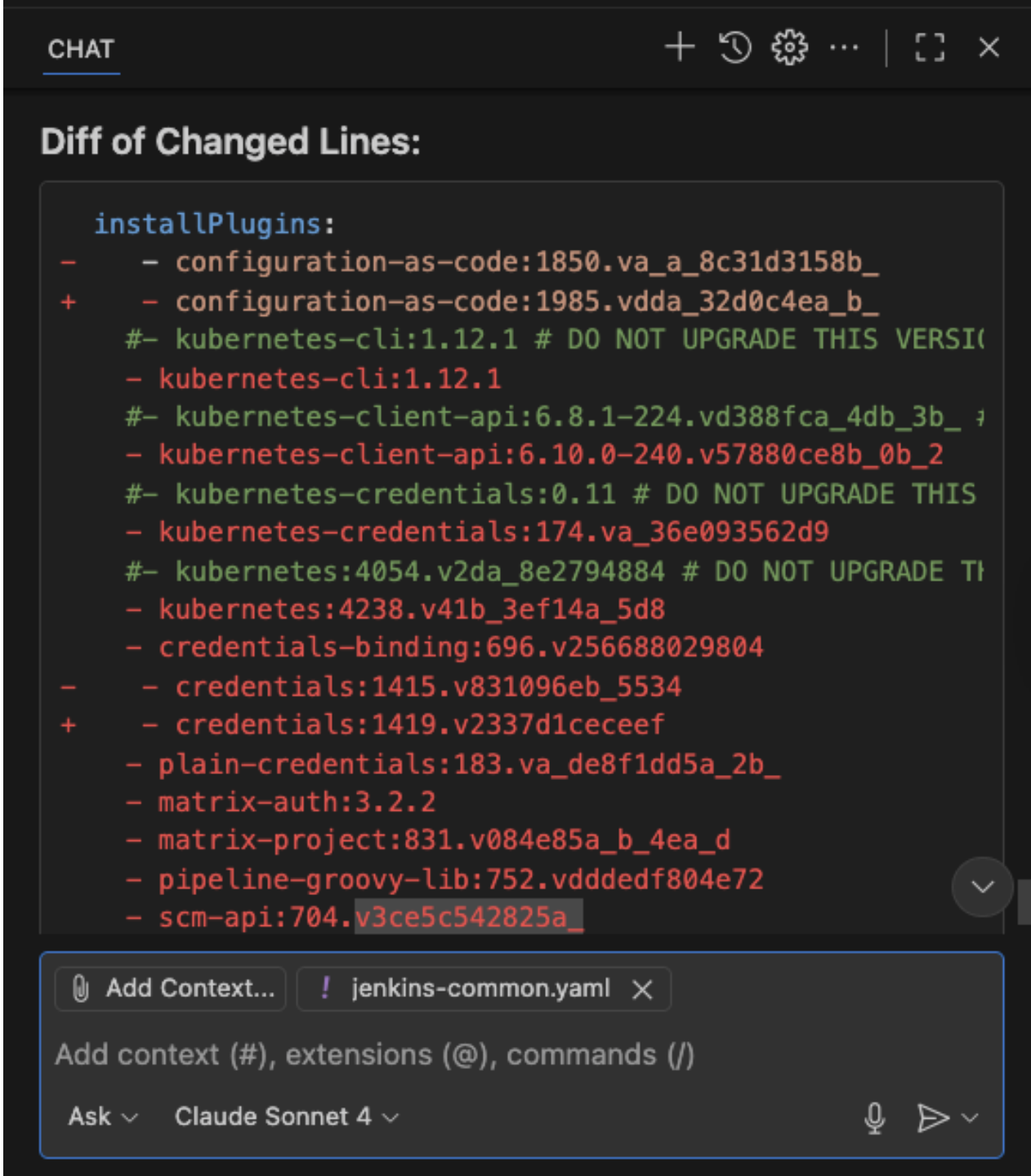
Best practices – tag milestones and important working changes!

Tag important milestones (for history and for accessibility)



How Git can manage AI changes

- VS Code Preferences: Open User Settings (JSON)
  "github.copilot.chat.codeGeneration.instructions": [
  {
  "text": "When suggesting code changes, must show only diff"
  }]
- Ask AI give you small changes



Control AI Changes with **Auto-Commit**-Push

VS Code Preferences: Open User Settings (JSON)

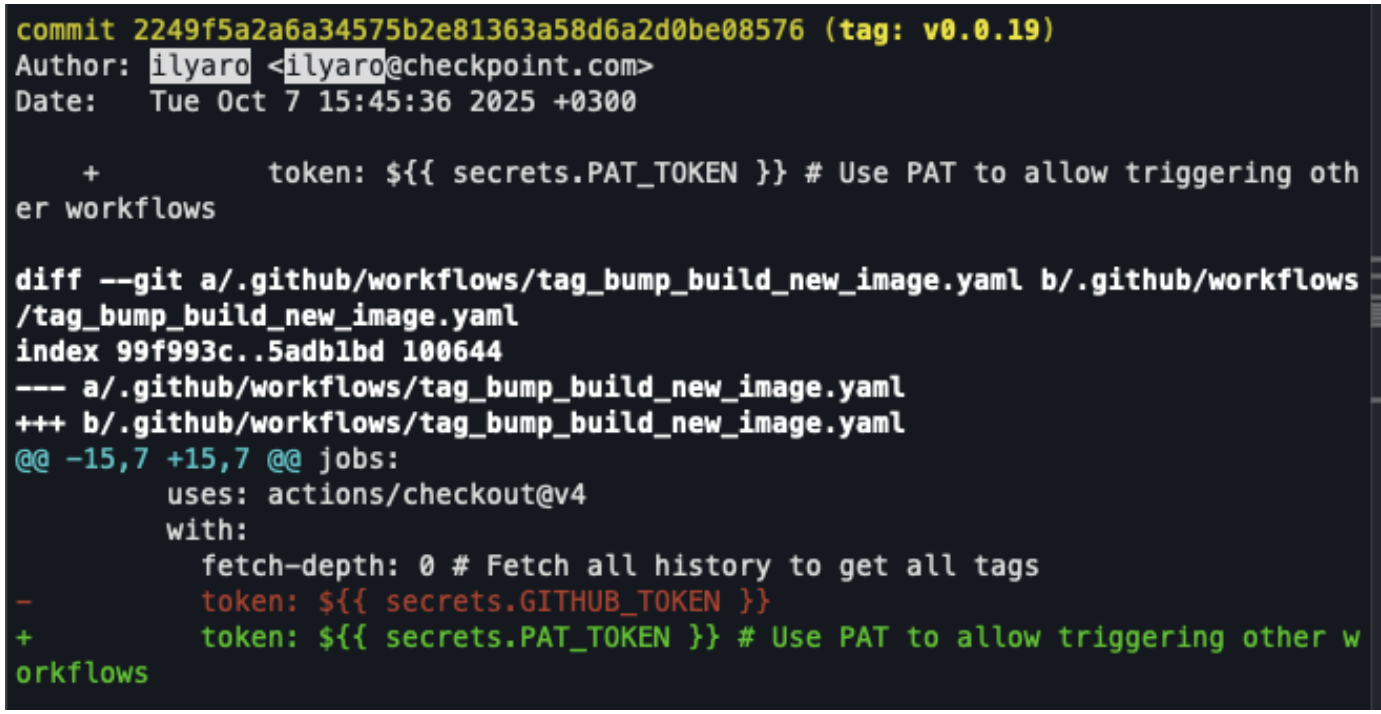
```
"emeraldwalk.runonsave": {
    "commands": [
    {
        "match": ".*",
        "cmd": "git rev-parse --is-inside-work-tree >/dev/null 2>&1 && git ls-files --error-unmatch ${file} && git commit -m \"`git diff -U0 ${file} | tail -1`\" ${file} && git symbolic-ref --short refs/remotes/origin/HEAD | grep -v $(git branch --show-current) > /dev/null 2>&1 && git push"
    }]},
```

Tag important working changes and push:

$ git tag v0.0.19 && git push –tags

- Save time for commits, keep them on the git server in a feature branch
- Reset to working tag when needed

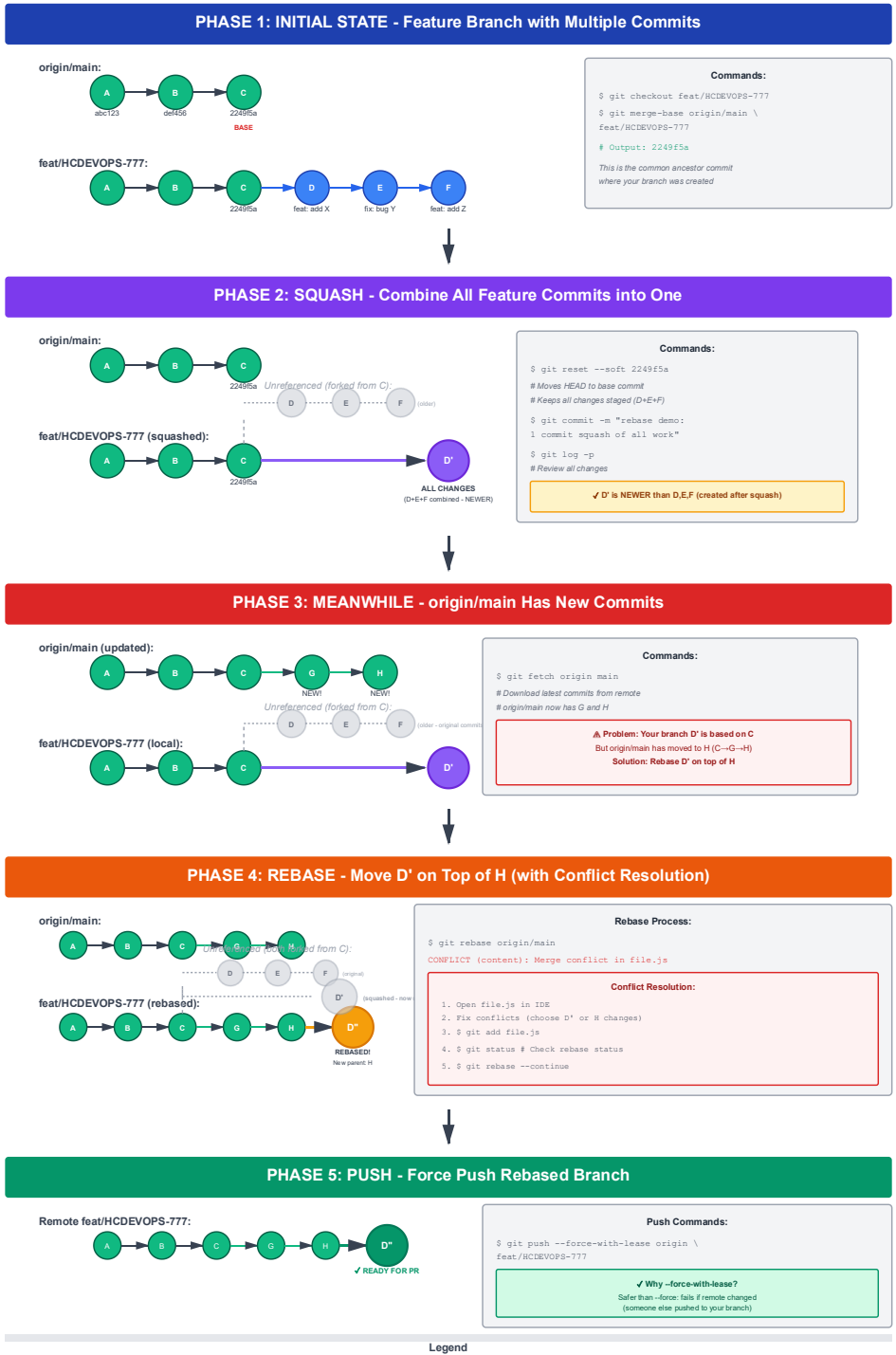
$ git reset –hard v0.0.19



Rebase on top of updated origin/main branch before pushing for code review.

Do it now on your current work branch.

https://github.com/ilyaro/git_best_practices_ppt/blob/master/Git_Rebase_Best_Practices.md



Best practices – squashing

Before pushing, squash related changes together to make for better understanding by others

Best practices - concise commit messages
For manual 1, commit before opening PR/MR

Limit commit message header to **60 characters** and add the meaningful details in the rest of the message

```
commit f6ce5cc010bf6665a8f2a701e7983e0c2ac8f144
Author: Shawn O. Pierce <sop@google.com>
Date:   Thu Nov 29 09:55:47 2012 -0800

    Sort comments before emailing them

    The order supplied by the caller can be random, ensure comments
    get sorted into a sane order before they are included into the email.

    Bug: issue 1692
    Change-Id: ibd85e514977545d022f936a5993f2a6ef6e52321
```

Best practices – Branch Layout

Branch layout

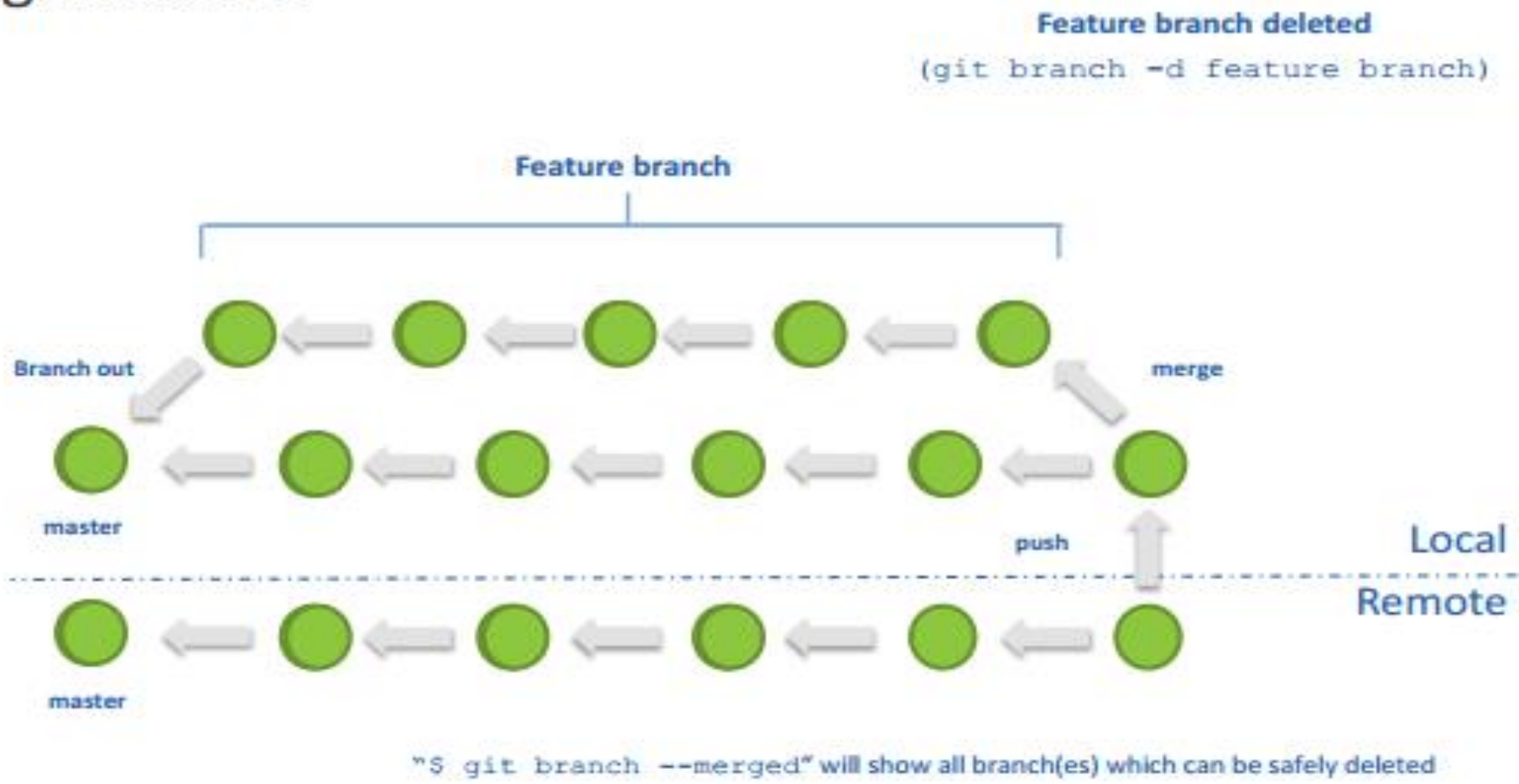
The branch layout is up to you, but there are some best practices though:

```
$ git branch # GOOD
  master
* devel
  feature/new-mailform
  fix/off-by-one
  fix/readme-grammar
```

```
$ git branch # BAD
  master
* devel
  new
  fix
  fix2
  t3rrible-br@nch-name
```

Best practices – clean up local branches

Clean up local branches once the code gets pushed to target branch



Lab 8 - 10

- We will create branch bugfix2, from Release_01
  Cherry-pick 1 commit from the master branch
- Undo modified file, undo staged file
  Undo the latest local commit, revert the pushed commit
- Stash meanwhile work aside, make a commit, return work from stash

Lab 11 - 13

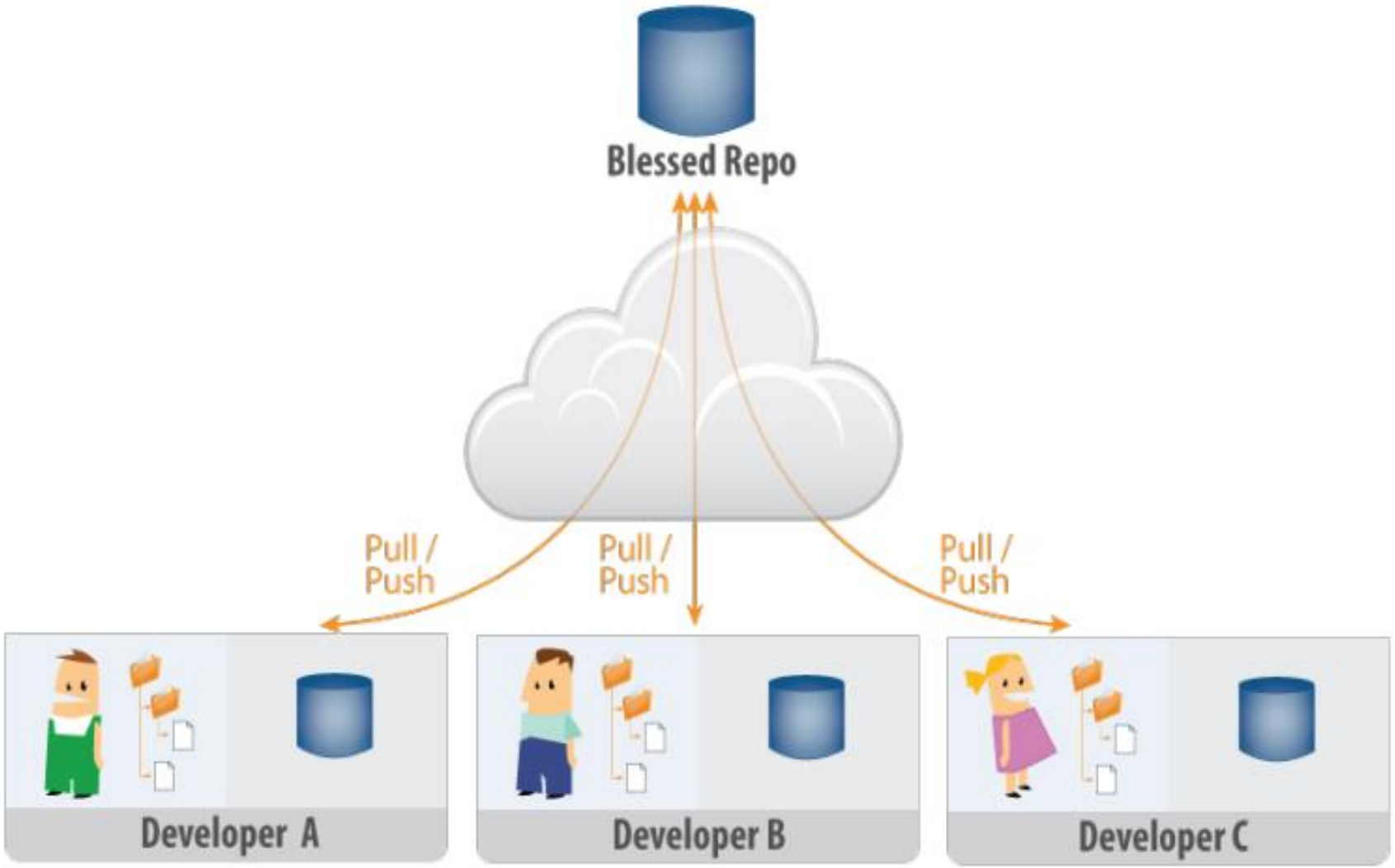
- Format patch in 1 repository, and apply it in another repository
- We will create 2 local commits squash them to 1 commit by interactive rebase, and push only 1 commit to the remote repository
- Create a commit in 1 repository and pull it from another repository, without pushing it to the origin repository

Question?

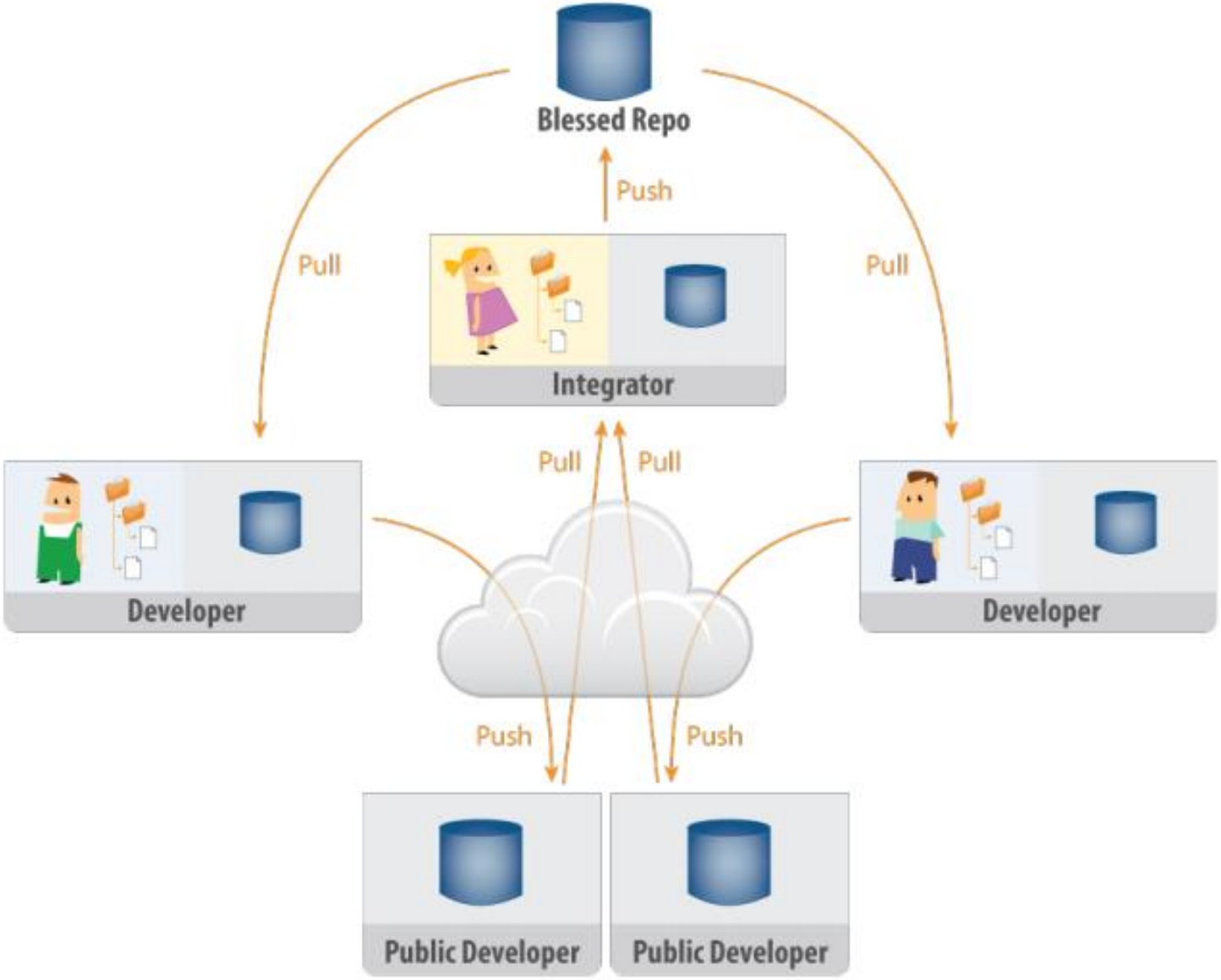
Thanks!

Backup slides

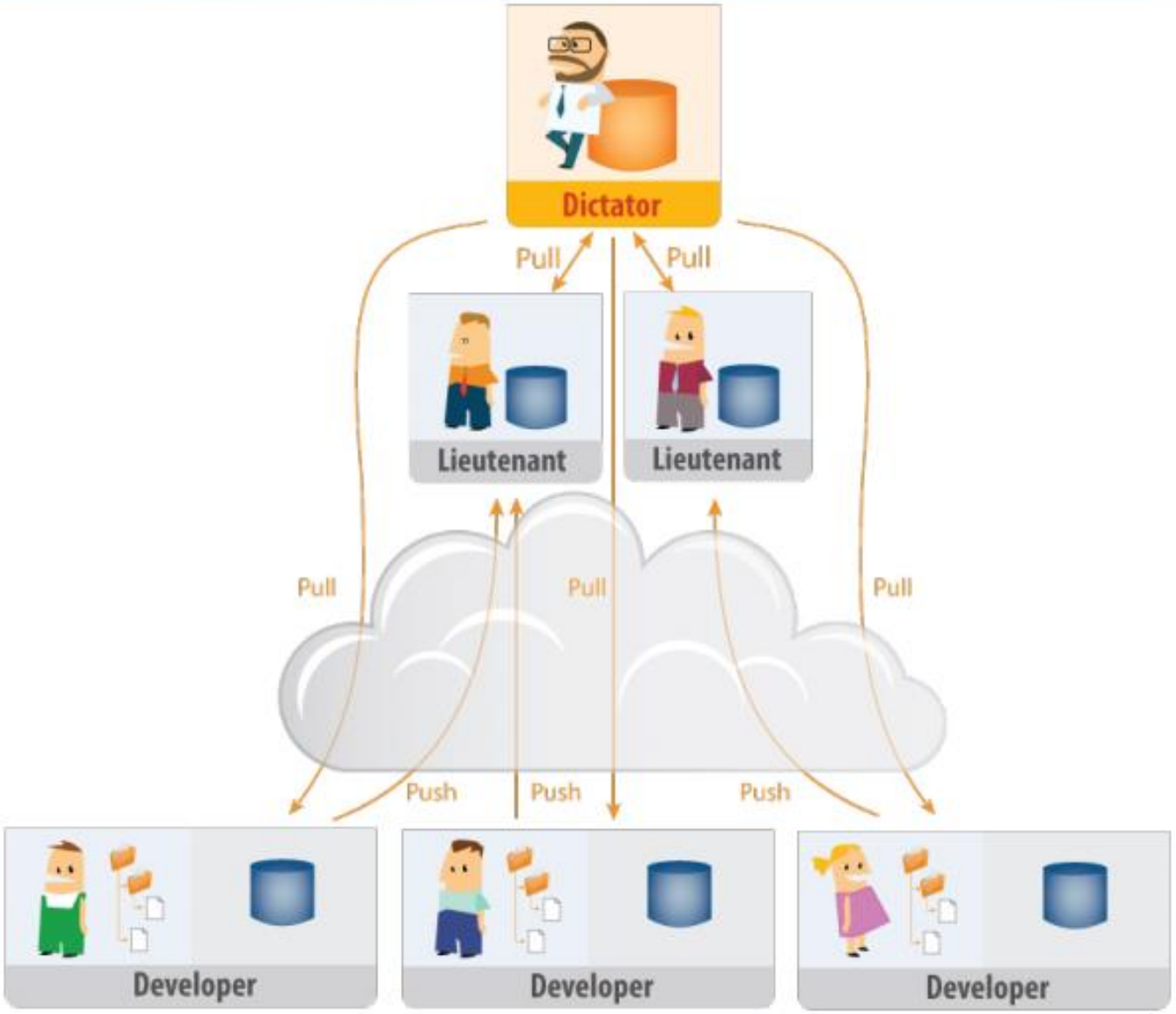
Centralized Workflow



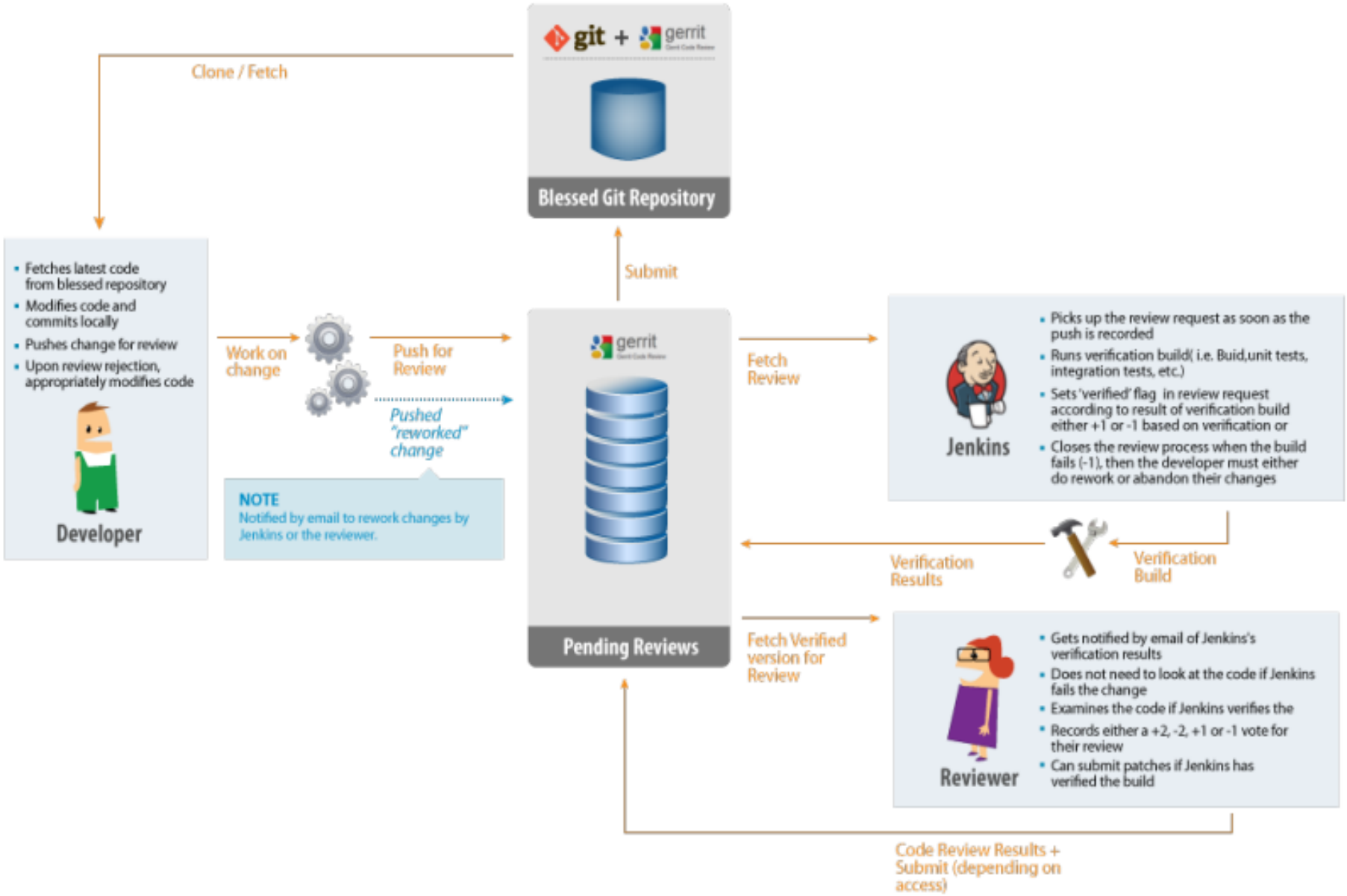
Integrators Workflow



Dictator / Lieutenants Workflow



Gerrit Code Review Workflow



Detached HEAD

If you checkout any commit SHA1, tag, or remote-tracking branch then you will end up having a “detached HEAD“:

```
$ git checkout 494e2cb73ed6424b27f9766bf8a2cb29770a1e7e
Note: checking out '494e2cb73ed6424b27f9766bf8a2cb29770a1e7e'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b new_branch_name

HEAD is now at 494e2cb... Added README file
```

Git stash

You may be in a state where you have some changes that are not ready for committing, but you need to change branches in order to work on something else.

```
sheta@SHETA-THINK ~/my-project (fix-off-by-one)
$ git status
# On branch fix-off-by-one
# Changes to be commited:
#    (use "git reset HEAD <file>…" to unstage)
#
#       modified:    README.txt

$ git checkout master
error: Your local changes to the following files would be overwritten by checkout

        README.txt
Please, commit your changes or stash them before you can switch branches.
Aborting
```

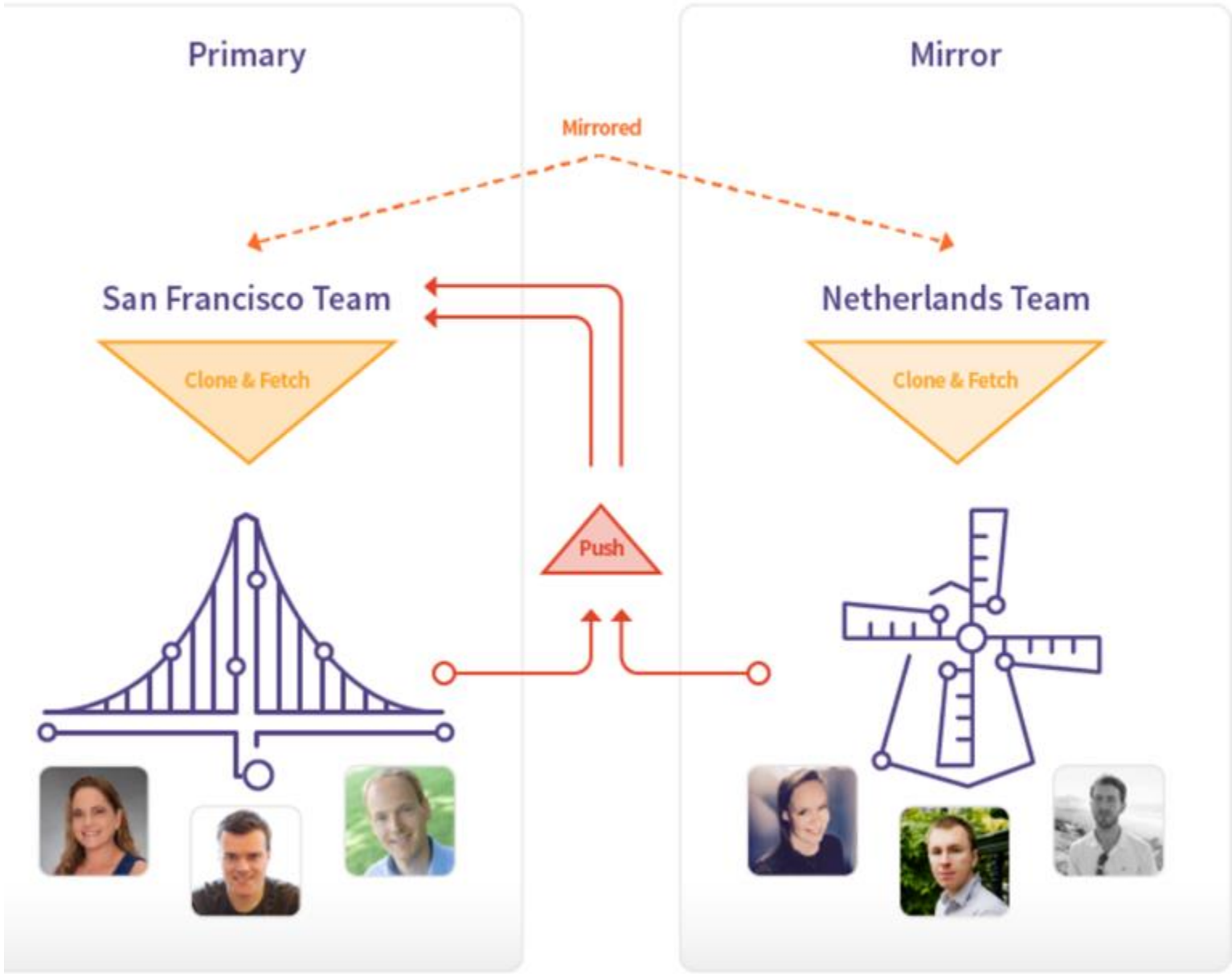
`git stash` takes current state of your working directory (what is staged, modified, etc.) and saves it as a stack of unfinished changes in **`refs/stash`**.

```
$ git stash save --all
Saved working directory and index state WIP on fix-off-by-one: ef2f6c3 Release r
e added
HEAD is now at ef2f6c3 Release note added
```

Later you can switch back to the previous branch and apply your saved changes to your working tree to have it exactly the way you had it prior to stashing your changes. You should

Git Master->Slave

Mirroring



What are tracking and remote-tracking branches?

- The combination of these branches defines a relationship between a local branch and one in the remote repository.
- When a repository is cloned, Git automatically creates **remote-tracking branches** (e.g., origin/master) for the remote branches and a **tracking branch** (e.g., master) to allow for local changes in relationship to the remote branch