GITHUB

Agenda - Day1

1. Internals of GIT - Folder Structure and Objects
2. Git Ignore
3. Branching and merging - Dealing with conflicts
4. Git rebase
5. Pushing repository to GITHUB
6. Synchronizing Remote branch with Local branch
7. Merging branch remotely through Pull Request
8. Forking a repository

Agenda - Day2

1. Tags
2. Two Factor Authentication
3. SSH setup
4. Branch Permissions
5. GITHUB Organization
6. Backup and Restore
7. Integrate with GUI tools
8. **Cherry Pick**
9. **git stash**

GIT Config

Global Level - All repositories for an account on a machine. ~/.gitconfig or

~/.config/git/config file: Specific to your user. You can make Git read and write to this file specifically by passing the --global option.

System Level - All repositories, all account on a machine. /etc/gitconfig file:

Contains values for every user on the system and all their repositories. If you pass the option --system to git config, it reads and writes from this file specifically.

Repository Level - For a particular repository. config file in the Git directory (that

is, .git/config) of whatever repository you're currently using: Specific to that single repository.

GIT Folder Structure

CONFIG - Contains settings for this repository.

HEAD - The current ref that you're looking at. In most cases it's probably refs/heads/master

HOOKS - This contains scripts that are executed at certain times when working with Git, such as after a commit or before a rebase.

INDEX - The staging area with meta-data such as timestamps, file names and also SHAs of the files that are already wrapped up by Git.

GIT Folder Structure

INFO - Relatively uninteresting except for the exclude file that lives inside of it.

LOGS - Contains history for different branches.

OBJECTS - Git's internal warehouse of blobs, all indexed by SHAs.

ORIG_HEAD - When doing a merge, this is the SHA of the branch you're merging into.

REFS - the master copy of all refs that live in your repository, be they for stashes, tags, remote tracking branches, or local branches.

DESCRIPTION - If you're using gitweb or firing up git instaweb, this will show up when you view your repository or the list of all versioned repositories.

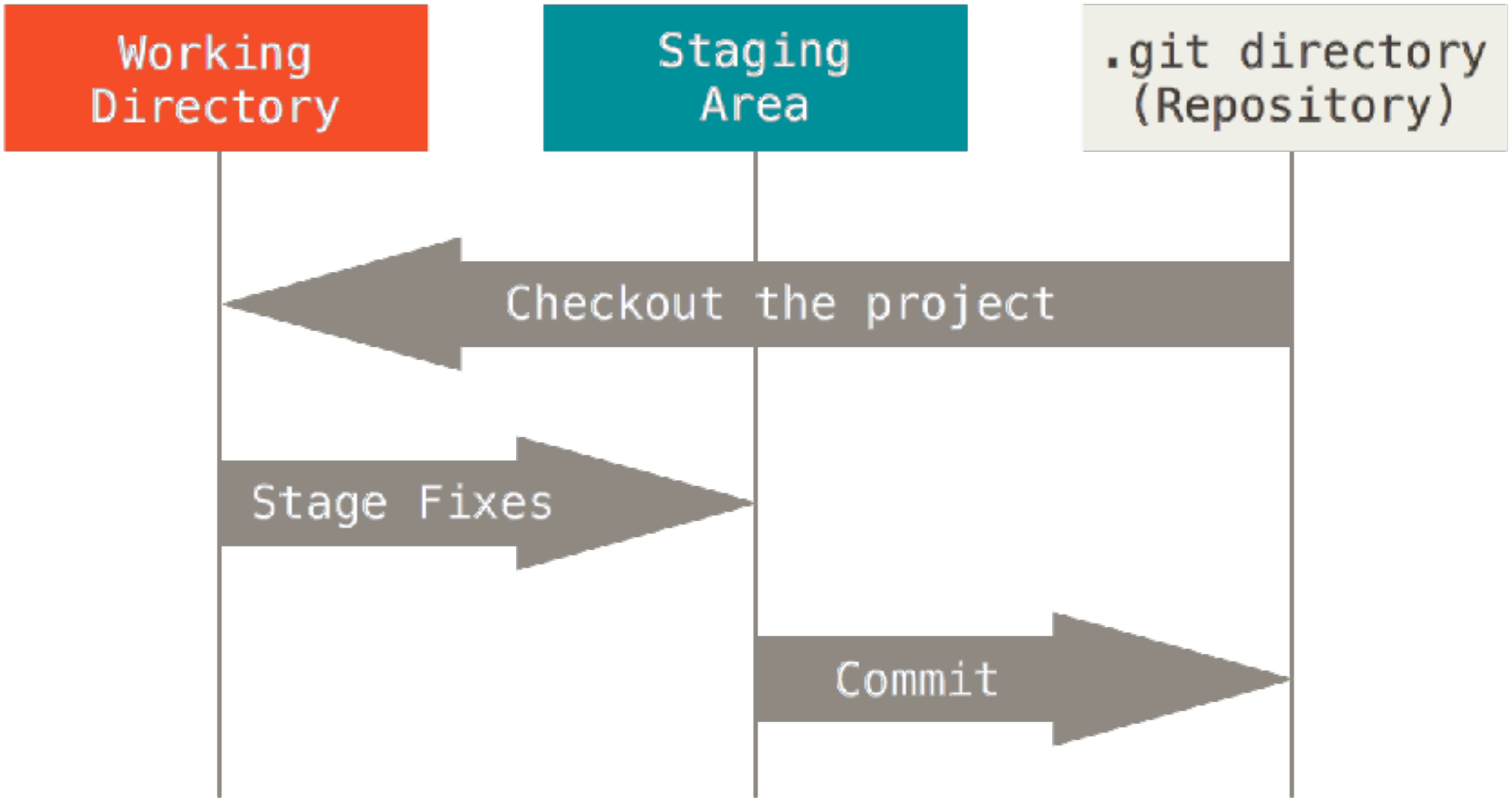
GIT Objects

BLOB - A blob generally stores the contents of a file.

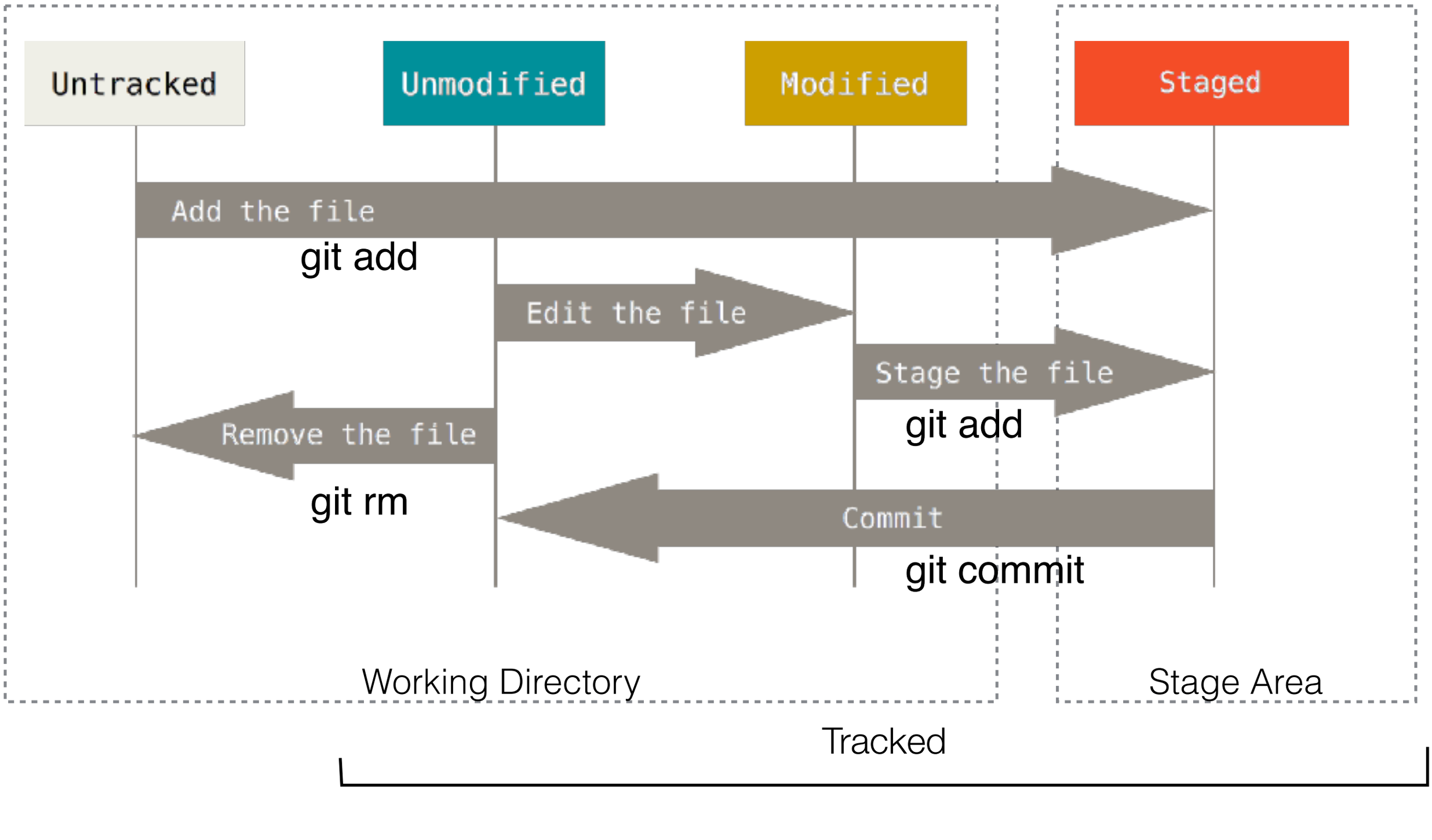
TREE - is basically like a directory - it references a bunch of other trees and/or blobs

COMMIT - Points to a single tree, marking it as what the project looked like at a certain point in time. It contains meta-information about that point in time, such as a timestamp, the author of the changes since the last commit, a pointer to the previous commit(s), etc.

Git Local Repository



GIT File Status



Exercise

1. Open Git bash
2. Configure - git config —global user.name ‘parvez’
3. Configure - git config —global user.email ‘parvez@abc.com’
4. Create folder - mkdir repo1 and cd inside it
5. Create a file - touch index.txt
6. Create repo - git init
7. Check the commit status - git status
8. stage(add) a file - git add index.txt
9. Look at .git/object folder
10. Commit the file - git commit -m ‘commit message’
11. Look at .git/object folder
12. Check the commit status - git status
13. Modify the file again and repeat the above steps

Digging Into your Commits

1. git show
    1. git show <commitID>:filename - This will show the content of a particular file at a commit
    2. git show <commitID> - Shows commit and patch details
    3. git show --stat --oneline HEAD
    4. git show --name-only --oneline HEAD
    5. git log --name-status --oneline

2. List all changed files between two commits :
    1. git show --pretty="format:" --name-only 7e00b7a..5dcb195 | sort | uniq
    2. git diff --name-only 7e00b7a..5dcb195
    3. git diff --name-status 7e00b7a..5dcb195

3. List a file that is changed in a particular commit:
    1. git diff-tree --no-commit-id --name-only -r bd61ad98
    2. git show --pretty="" --name-only 194b644
4. This will list all files at a particular commit
    1. git ls-tree --name-only -r <commit-ish>

5. git blame <commitID> <filename>

Unstaging

1. Revert to last commit: git reset —hard HEAD
2. Remove all added changes from index - git rest —mixed HEAD
3. Remove some files added to index - git reset HEAD <filename>

Gitignore

Gitignore - To ignore files to be tracked

- git check-ignore -v * - To list ignored files

Steps

- Create .gitignore file
- Add contents to be ignored
- Stage and Commit the file

git diff

- **git diff –staged** - Shows difference between staged and last commit
- **git diff**: Show differences between your working directory and the index.
- **git diff –cached**: Show differences between the index and the most recent commit.
- **git diff HEAD:** Show the differences between your working directory and the most recent commit.

Git branching workflow

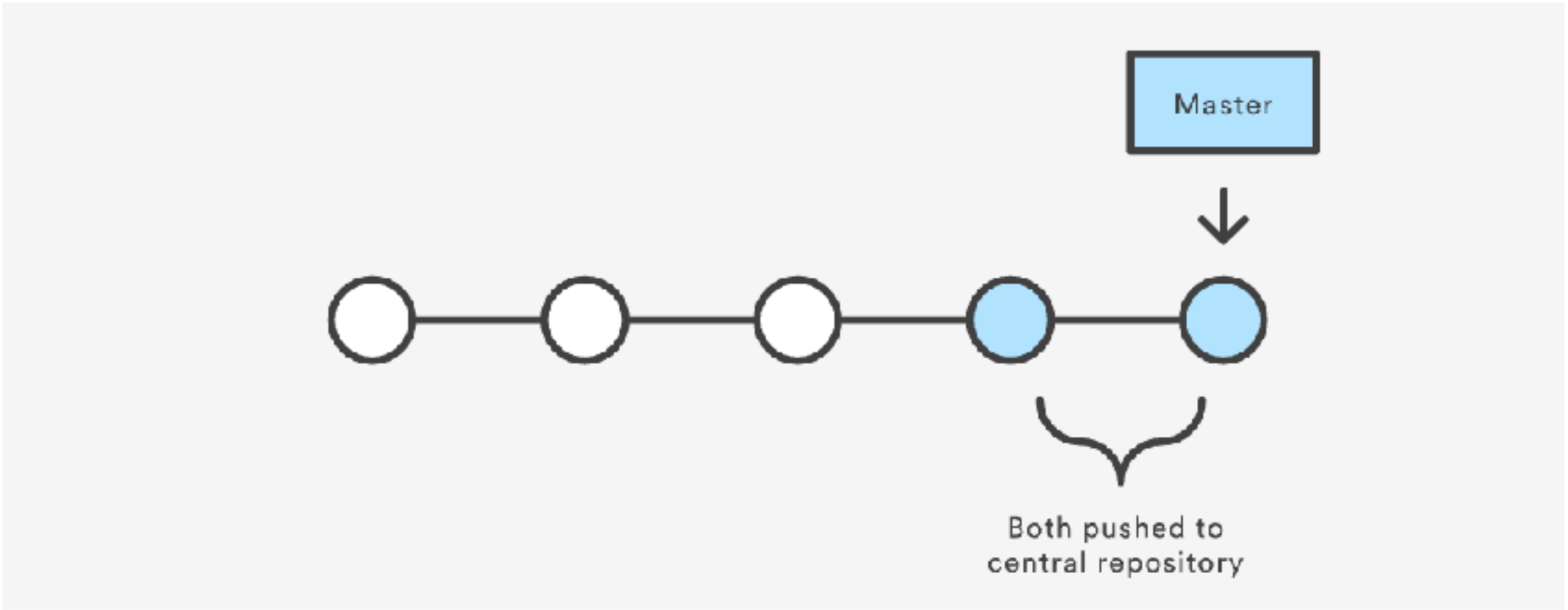
Centralized Workflow

Feature Branch Workflow

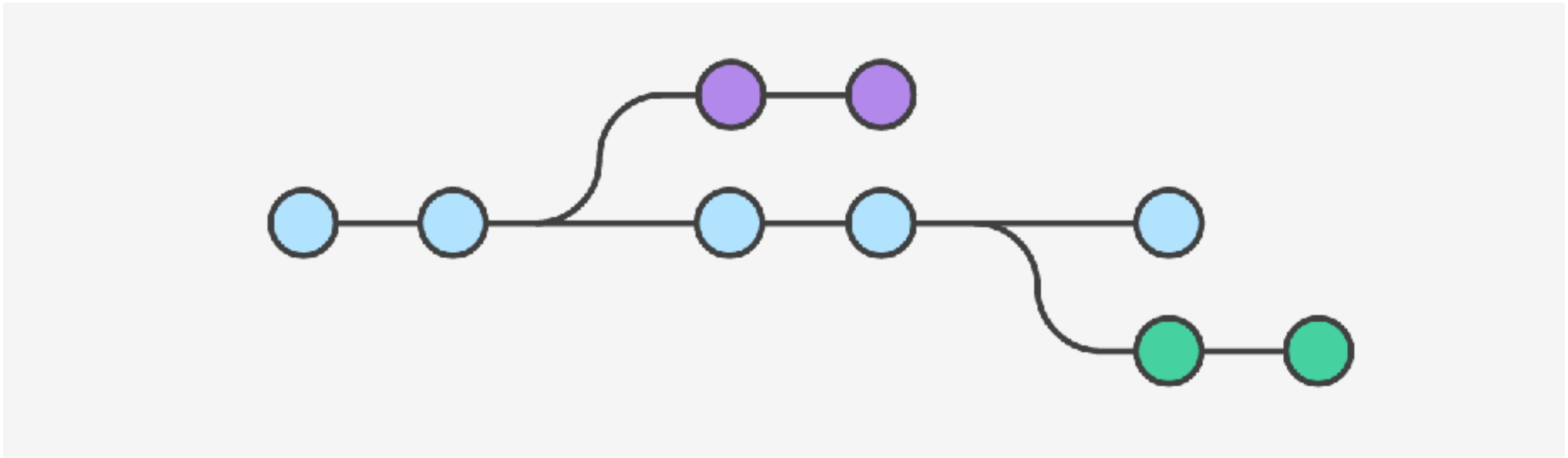
Gitflow workflow

Forking workflow

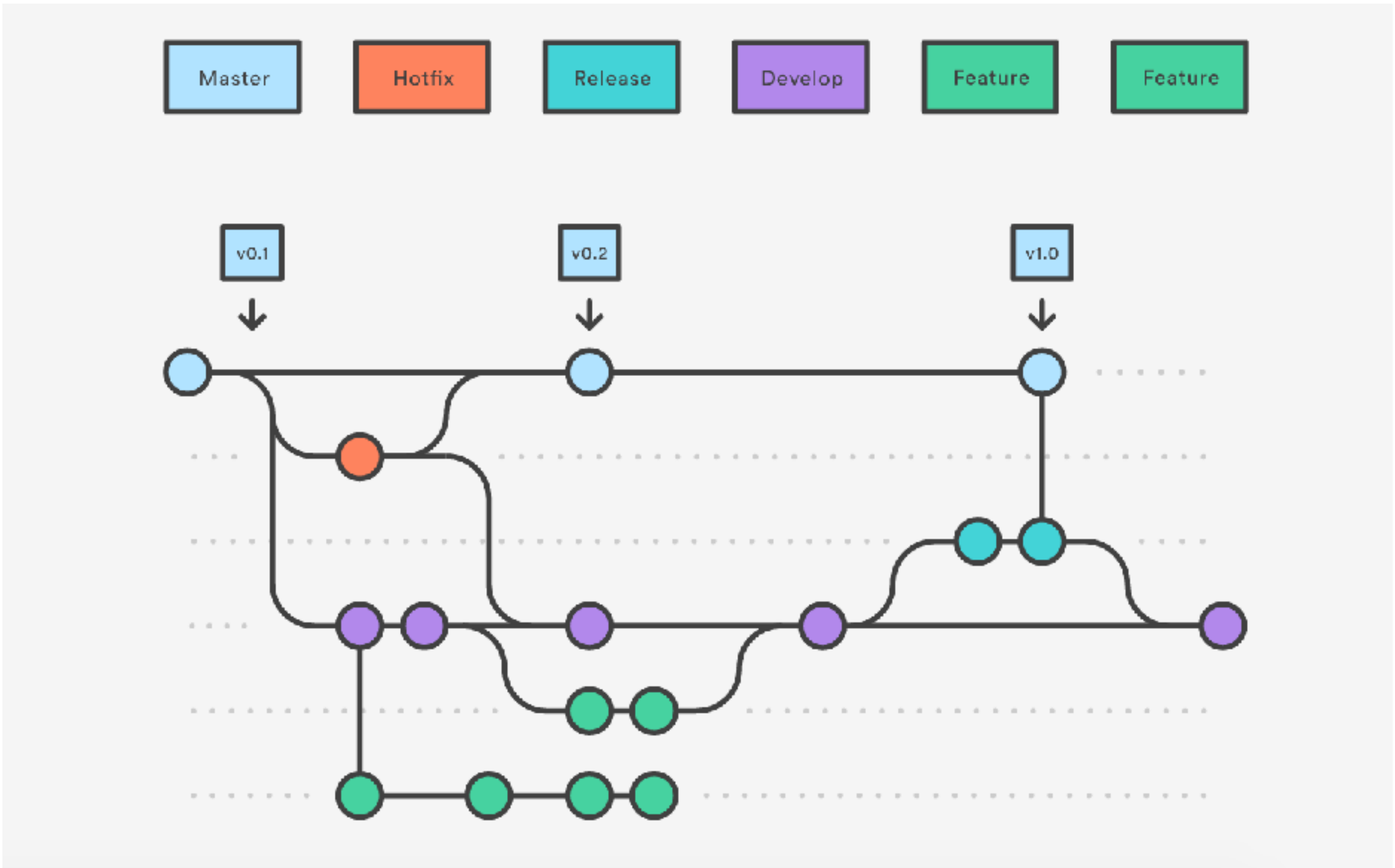
centralized Workflow



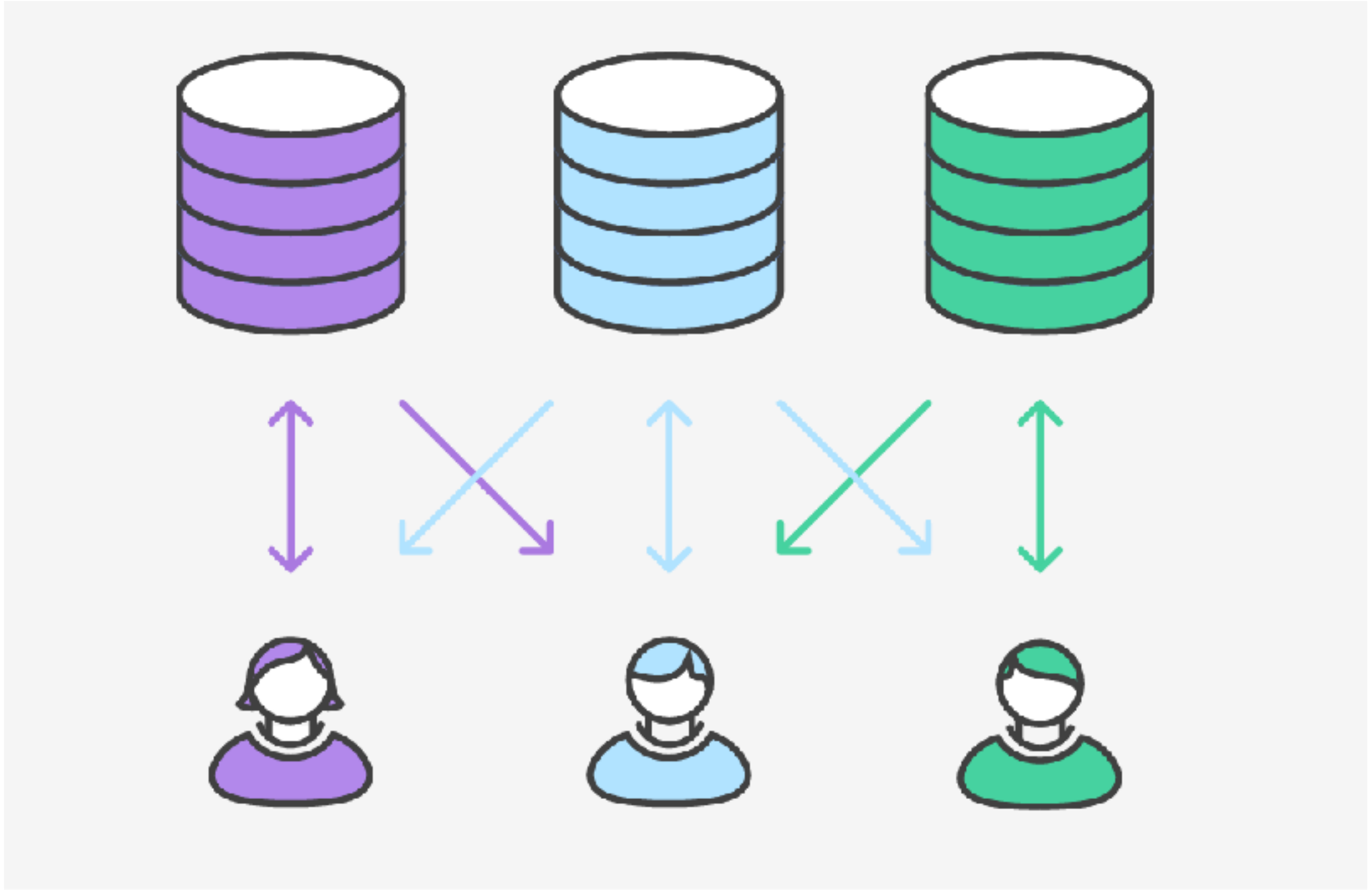
Feature Workflow



Git flow workflow



Forking Workflow



Tags

- Creating Tags:
  - Light weight tags: git tag v1.0-lw
  - Annotated Tags: git tag -a v1.1 -m "my version 1.1"
- git show v1.0
- git tag
- git tag -l "v1.8.5*"
- git log --tags --show-notes --simplify-by-decoration --pretty="format:%h %d %s"

GITHUB

User Account

Organization

Teams & Permissions

Collaborators

Pricing and Installation

Organization

- Helps centralizes your organization’s code
- Contains members and teams
- Simplifies management of group owned repositories
- Company specific dashboard
- Can add Read Only Collaborators
- Someone else can be given admin rights

Teams

- Teams give organizations the ability to create groups of members and control access to repositories.
- Teams can map to physical teams within your company, but they can also represent areas of interest or expertise.
- Team members can be granted read, write, or admin permissions to specific repositories.
- @mentions - to notify appropriate parties that you'd like to request their input or attention

Team Maintainer

- An organization owner can promote any member of the organization to team maintainer for a team, giving them a subset of privileges available to organization owners.

  - Change the team's name, description, and visibility
  - Add organization members to the team
  - Remove organization members from the team
  - Promote an existing team member to team maintainer

Contributors

Contribution is counted for

- **Issues and pull requests**
  - Done within last one year
  - For standalone repository and not on forked repository
- **Commits**
  - When made within an year
  - Email address used for commit is associated with github account
  - Standalone repo
  - On default branch (usually master)
  - In the gh-pages branch (for repositories with project pages site)

Milestones & Labels

- Can be created for Issues and Pull Requests
- Used to track the progress of similar issues and pull requests as they're opened and closed over time.

Labels can be applied to issues and pull requests to signify priority, category, or any other information that you find useful.

Notification

- Participating Notification
  - Someone mentions you
  - You are assigned to an issue or pull request
  - Someone makes a comment in a conversion you have subscribed to
- Watching Notification
  - Opened issues and their comments
  - Opened pull requests and their comments
  - Comments on any commits
  - Published releases

Outside Collaborators

Consultant or a temporary employee

Repository permissions - Read, write and Admin can be set for outside collaborators

@mentions

Get notified in email for Issues and pull request when someone mentions you as ‘@<username>

Organization Permission level

Organization Action	Owner	Members	Billing Managers
Invite people to join the organization	Yes		Yes
Add and remove people from all teams	Yes		
Promote organization members to team maintainer	Yes		
Add Collaborators to all repositories	Yes		
Access the organization audit log	Yes		
Delete all teams	Yes		
Delete the organization account, including all repositories	Yes		
Create teams	Yes	Yes	
See all organization members and teams	Yes	Yes	
@mention any visible teams	Yes	Yes	
Can be made a team maintainer	Yes	Yes	

Repository Permission Level for Organization

	Read	Write	Admin	Owner
Pull, push and clone all repositories in the organization				Yes
Promote organization members to team maintainer				Yes
Convert organization members to outside collaborators				Yes
Create repositories	Yes		Yes	Yes
Delete repositories			Yes	Yes
Change repository settings			Yes	Yes
Transfer repository into and out of the organization account			Yes	Yes
Add a repository to a team			Yes	Yes
Add outside collaborators			Yes	Yes
Pull from the teams assigned	Yes	Yes	Yes	Yes
Push to the team's assigned		Yes	Yes	Yes
Fork the team's assigned repositories	Yes	Yes	Yes	Yes
Send pull requests from forks of the team's assigned repositories	Yes	Yes	Yes	Yes
Merge and close pull requests		Yes	Yes	Yes

Repository Permission Level for Organization

	Read	Write	Admin	Owner
Open issues	Yes	Yes	Yes	Yes
Close, reopen, and assign issues		Yes	Yes	Yes
Apply labels and milestones		Yes	Yes	Yes
Create and edit releases		Yes	Yes	Yes
View draft releases		Yes	Yes	Yes
View published releases	Yes	Yes	Yes	Yes
Edit and delete their own comments on commits, pull requests, and issues	Yes	Yes	Yes	Yes
Edit and delete anyone's comments on commits, pull requests, and issues		Yes	Yes	Yes
Edit wikis	Yes	Yes	Yes	Yes

LDAP

- Instances using LDAP for user authentication can use LDAP Sync to manage a team's members by mapping a team to an LDAP group on your LDAP server by setting the groups Distinguished Name (DN) in the LDAP group field.

- LDAP Sync only manages the team's member list. You must manage the team's repositories and permissions from within GitHub Enterprise.

- If an LDAP group mapping to a DN is removed, such as if the LDAP group is deleted, then every member is removed from the synced GitHub Enterprise team. To fix this, map the team to a new DN, add the team members back, and manually sync the mapping.

Forking a repository

- A Fork is a copy of a repository
- It allows you to freely experiment with changes without affecting the original repository
- Forks are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea.

Two factor Authentication

- Require 2 authentications - One is the password and second the code sent to your phone
- Configuring through OTP mobile
- Configuring via text message
- Configuring via FIDO U2F

github Search

- When the issue or pull request was Closed
    - silly in:body closed:<YYYY-MM-DD - Matches issues and pull requests with the word "silly" in the body that were closed before October 2012.
    - weird in:body updated:>=2013-02-01 - Matches issues with the word weird in the body that were updated after February 2013.
    - author:puzrin created:2012-11-01..2012-12-01 - Matches issues created by @puzrin created between November and December of 2012.
- Search by number of comments
    - state:closed comments:>100 - Matches closed issues with more than 100 comments.
- Search by commit SHAs within pull request
    - e1109ab - Matches pull requests with a commit SHA that starts with `e1109ab`.
- Search by type
    - cat type:pr - Matches pull requests with the word cat.
    - github commenter:defunkt type:issue - Matches issues that contain the word github, and have a comment by @defunkt.

github Search

In qualifer

- warning in:title: Matches issues with warning in their title.
- error in:title,body: Matches issues with error in their title or body.
- shipit in:comment: Matches issues mentioning :shipit: in their comments.

SSH for windows

- Checking existing SSH key : ls -al ~/.ssh
- Generate SSH Key: ssh-keygen -t rsa -b 4096 -C “your_email@example.com"
- Ensure ssh-agent is enabled: eval "$(ssh-agent -s)”
- Add your SSH key to the ssh-agent: ssh-add ~/.ssh/id_rsa
- clip < ~/.ssh/id_rsa.pub (mac: pbcopy < ~/.ssh/id_rsa.pub)

SSH Error - permission denied

- Check that you are connecting to the right server: ssh -vT git@github.com
  - Note IP address. This should be GITHUB IP address port 22
- Always use GIT user
  - ssh -T git@github.com (message:Hi username! You've successfully authenticated…)
- Make sure you have key being used
  - Open terminal
  - Verify you have private key generated and loaded into SSH.
    - eval "$(ssh-agent -s)”
    - ssh-add -l (ssh-add -l -E md5 for open ssh 6.8 onwards)
- check that the key is being used by trying to connect to git@github.com: ssh -vT git@github.com
- Verify the public key is attached to your account
  - Start SSH agent in the background. - eval "$(ssh-agent -s)”
  - Find and take a note of your public key fingerprint - ssh-add -l

Backup

- https://docs.haskellstack.org/en/stable/README/
- git clone https://git.joeyh.name/git/github-backup.git
- cd github-backup
- stack install
- RUN: github-backup pmisarwala mybackup
- github-backup BooleanMinds