Ask Git

Version control system with GIT

Daniel Kwaro, KEMRI-CGHR

25th June 2019

Version control the 'usual way'

- There are many ways to track history of files

- The most basic is to add systematic suffixes to file names:
    - main_original.do
    - main_revised_1.do
    - main_revised_2.do
    - and so on...

Version control the 'git way'

1. Make changes
2. Take a snapshot of changes
3. GOTO line 1

- Adds ability to :
    - move to any snapshot in the history of changes
    - back-up history of changes
    - collaborate or work remotely

Let's create a story

#create folder to store the story file

#use notebook++ to write a story

- Once upon a time_____, and every day_____, until one day_____, and because of that_____ and because of that_____until finally______ and ever since then______.
  - This is a story about <character>
  - Once upon a time there lived .....
  - Everyday,<character did this and that>
  - One day, <event happened>
  - And then....
  - And then....
  - Finally.......
  - Ever since

Working alone

Download and configure git application

#download and install git:

- https://git-scm.com/downloads

#download and install git GUI client

- https://desktop.github.com
- https://tortoisegit.org/
- or, interact with git through command line

#configure git

- git config --global user.name "My Name"
- git config --global user.email myEmail@example.com

You might want or be required to use an SSH key:

tutorial by DigitalOcean: https://docs.gitlab.com/ce/ssh/README.html

Create local repository

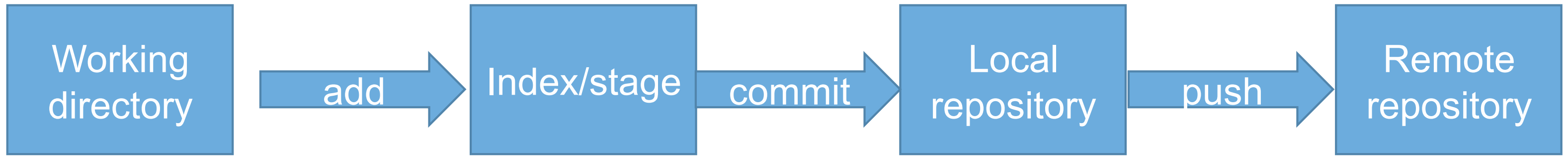
# open the workspace/directory/folder you want to keep track of

```
pwd  #print/display the current working directory
cd path/to/workspace
```

# ask git to initialize/create a repository in the current workspace/working directory

```
git init
```

#git maintains three local “trees” and synchronizes with remote “tree”



Save most up-to-date version of project

# ask git to mark files you would like to register, i.e. add them to the “index/list of staged files”

```
git add <filename>
```

# ask git to record/register/save/commit/log the marked/index files into the repository

```
git commit – m “<message describing the initial snapshot>
```

#ask git to display/show details of a specific commit/snapshot

```
git show
```

***commit refers to both the verb/action of making a snapshot/revising and the noun/snapshot/ revision**

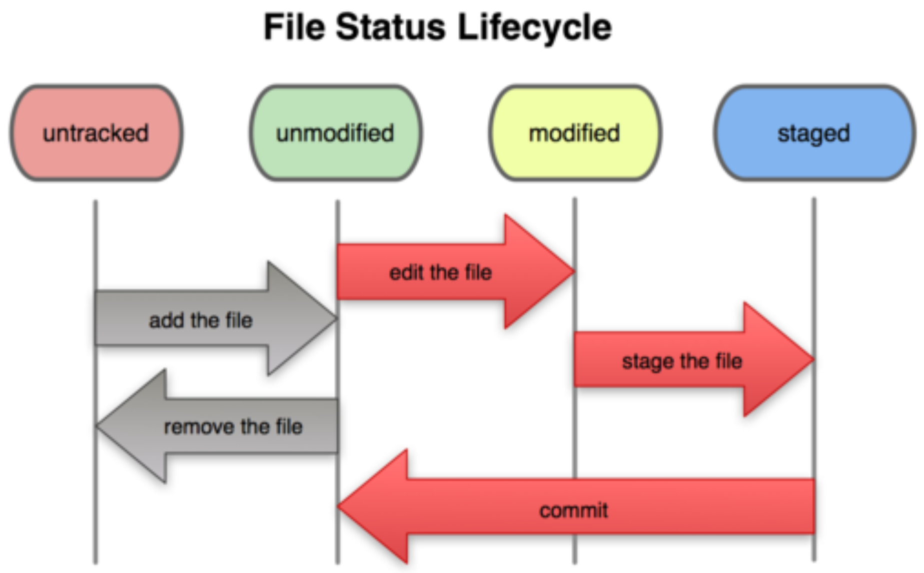
Make changes to the workspace

#change code in working directory:

- Create new file
- Modify existing file
  - change contents
  - change name/ rename
  - change location/move
- Delete existing file

#ask git to display status of files and folders in workspace

git status



Save new changes made to the workspace

# ask git to mark files you would like to add to stage or list of changes to register later

git add

# ask git to register/commit marked files

git commit

#ask git to display status of all files in the working directory

git status

#ask git to display/show details of the specific commit

git show <commit>

#ask git to display the differences between commits

git diff <commit-from>..<commit-to>

Make and record more changes to the workspace

#change code in working directory:

- Create new file
- Modify existing file
  - change contents
  - change name/ rename
  - change location/move
- Delete existing file

#ask git to record the changes

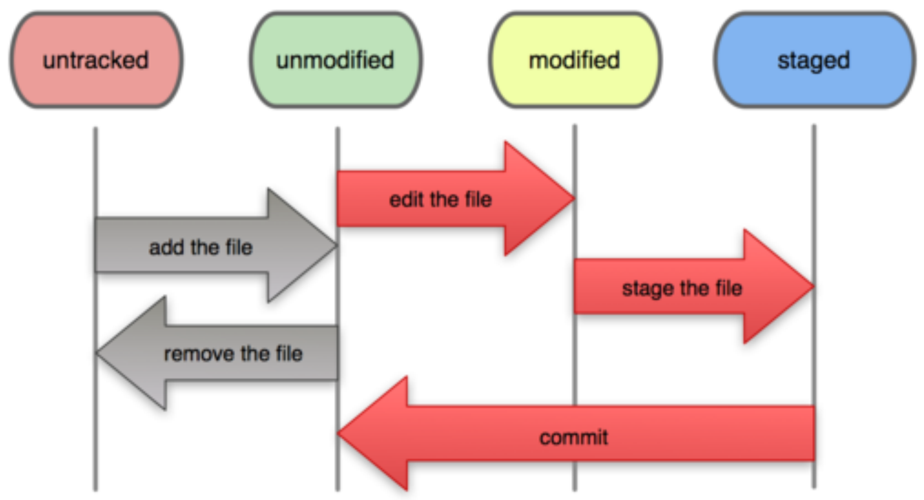
git add

git commit

#ask git to display status of files and folders in workspace

git status

File Status Lifecycle



Check history of changes made to the workspace

#ask git to display the history/log of registered changes

git log

git log --pretty=oneline *//see a very compressed log with one line for every commit*

git log --graph --oneline --decorate –all //see ASCII art tree of all branches

git log –name-status *//see only files which have changed*

Inspecting a repository

#to view status of files in working directory and staging area:

git status or git status –s (short version)

#to see what is modified but unstaged:

git diff

#to see a list of staged changes:

git diff --cached

#to see a log of all changes in your local repo:

git log or git log --oneline (shorter version)

git log -2 (to show only the 2 most recent updates)

Create a branch to track “experimental” changes to the workspace

# ask git to create a new feature branch out of the master branch

git branch <feature_x>

#ask git to display the list of branches and indicate current/active branch

git branch

# ask git to switch/checkout to the new branch

git checkout

Make and record more changes in the experimental/feature branch

#change code in working directory:

- Create new file
- Modify existing file
  - change contents
  - change name/ rename
  - change location/move
- Delete existing file

#ask git to record the changes

git add

git commit

#ask git to display status of files and folders in workspace

git status

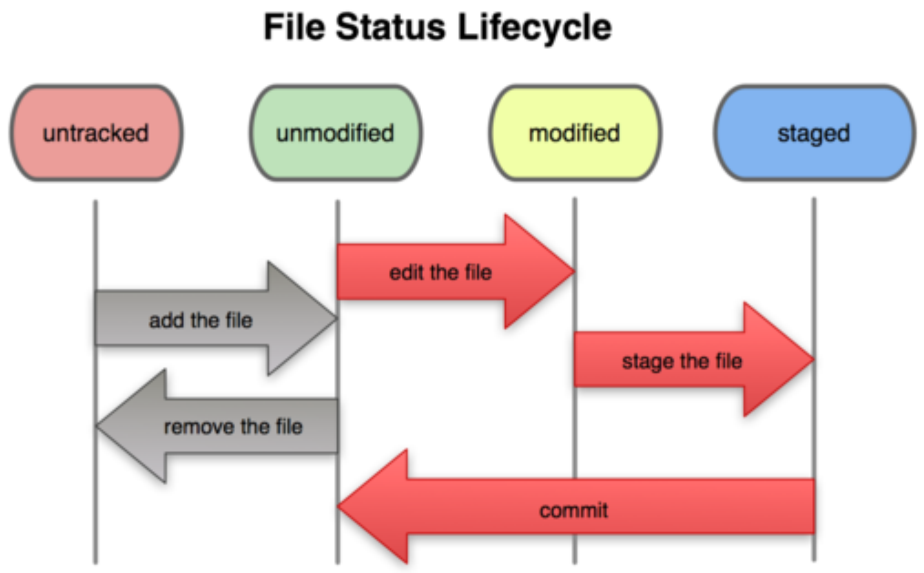
#compare the two branches

#checkout main

#list files in working directory

#checkout <feature_x>

#list files in working directory



Undo changes on a file

#ask git to display commit history

git log --oneline in your terminal window.

#Copy the commit hash for the second commit in the log: 52f823c then press q to exit the log.

#ask git to reset the specific commit

 git revert 52f823c

#ask git to display status of your files

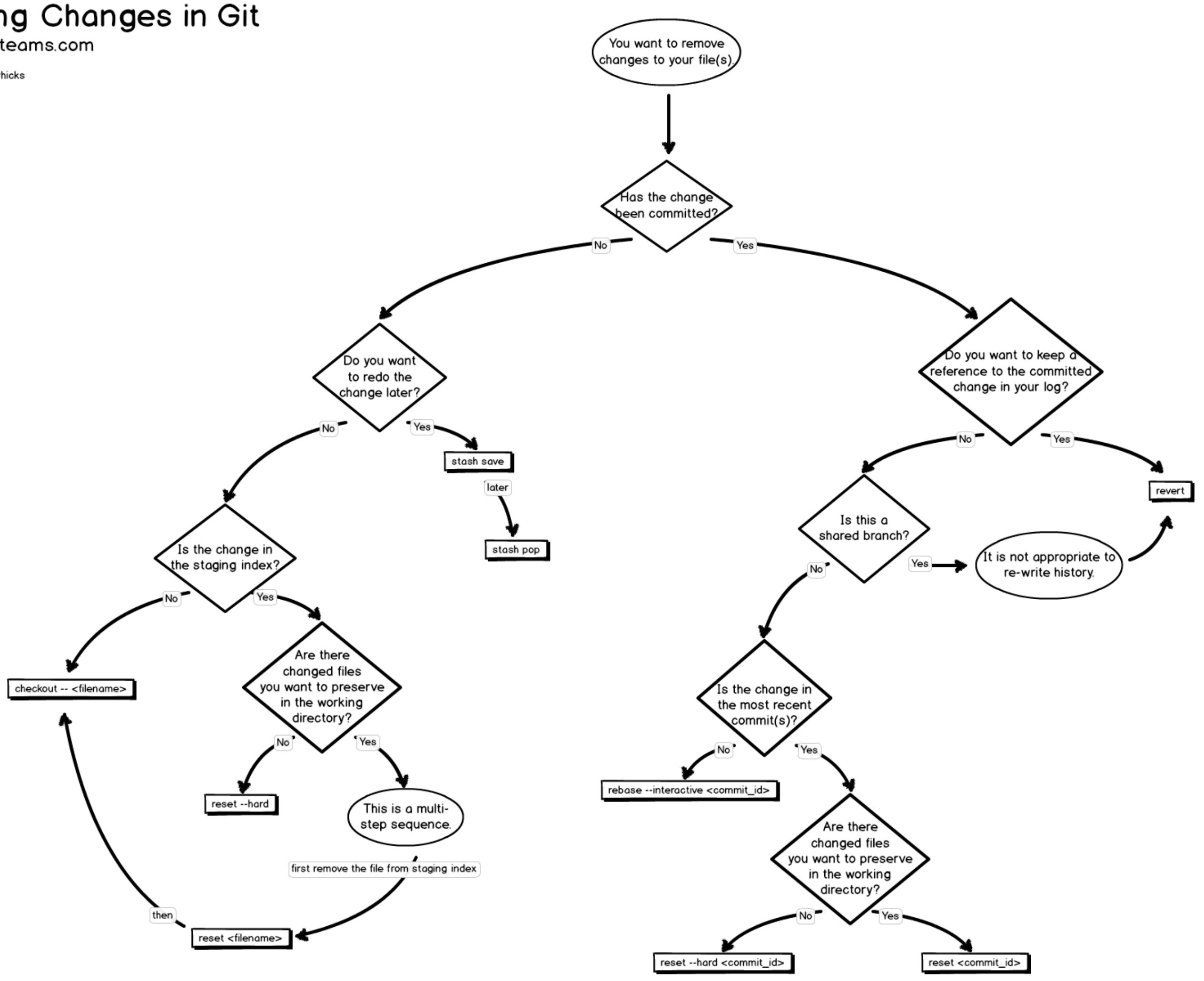
git log //a new commit was created that undoes the changes

git status //the commit was undone and is now an uncommitted change

Undoing Changes in Git

www.gitforteams.com

Author: emmajane
Contributors: jameyhicks


You want to remove changes to your file(s).
Has the change been committed?
No
Yes
Do you want to redo the change later?
No
Yes
stash save
later
stash pop
Is the change in the staging index?
No
Yes
checkout -- <filename>
Are there changed files you want to preserve in the working directory?
No
Yes
reset --hard
This is a multi-step sequence.
first remove the file from staging index
reset <filename>
then
Do you want to keep a reference to the committed change in your log?
No
Yes
revert
Is this a shared branch?
Yes
It is not appropriate to re-write history.
No
Is the change in the most recent commit(s)?
No
rebase --interactive <commit_id>
Yes
Are there changed files you want to preserve in the working directory?
No
reset --hard <commit_id>
Yes
reset <commit_id>

Merge feature branch to the master branch

#ask git to switch/checkout to master branch: git checkout master

#ask git to check whether current/active branch is master: git branch

#check current list of files in master: ls

#preview updates before merging: **git diff <source_branch> <target_branch>**

#merge feature branch **into** master branch: git merge <feature_x>

#check current list of files in master branch: ls

#display branching structure

~delete experiemental branch if stale then display branching structure

***never commit to master because master will be the branch that you pull upstream changes from**

*Use git pull –rebase to move all of your commits to the tip of the history.

Create remote repo for back up

- Introduce yourself to GitHub

#Register on GitHub:

https://github.com/

#Authenticating to GitHub Desktop

https://help.github.com/desktop/guides/gettingstarted/authenticating-to-github/

#Configuring Git for GitHub Desktop

https://help.github.com/desktop/guides/gettingstarted/configuring-git-for-github-desktop/

create online repository to back up your history and work remotely

#configure the remote repo

git remote add origin <link of your central repository>

#Push your local changes to the remote repo.

#**Pull** from remote repo to get most recent changes.

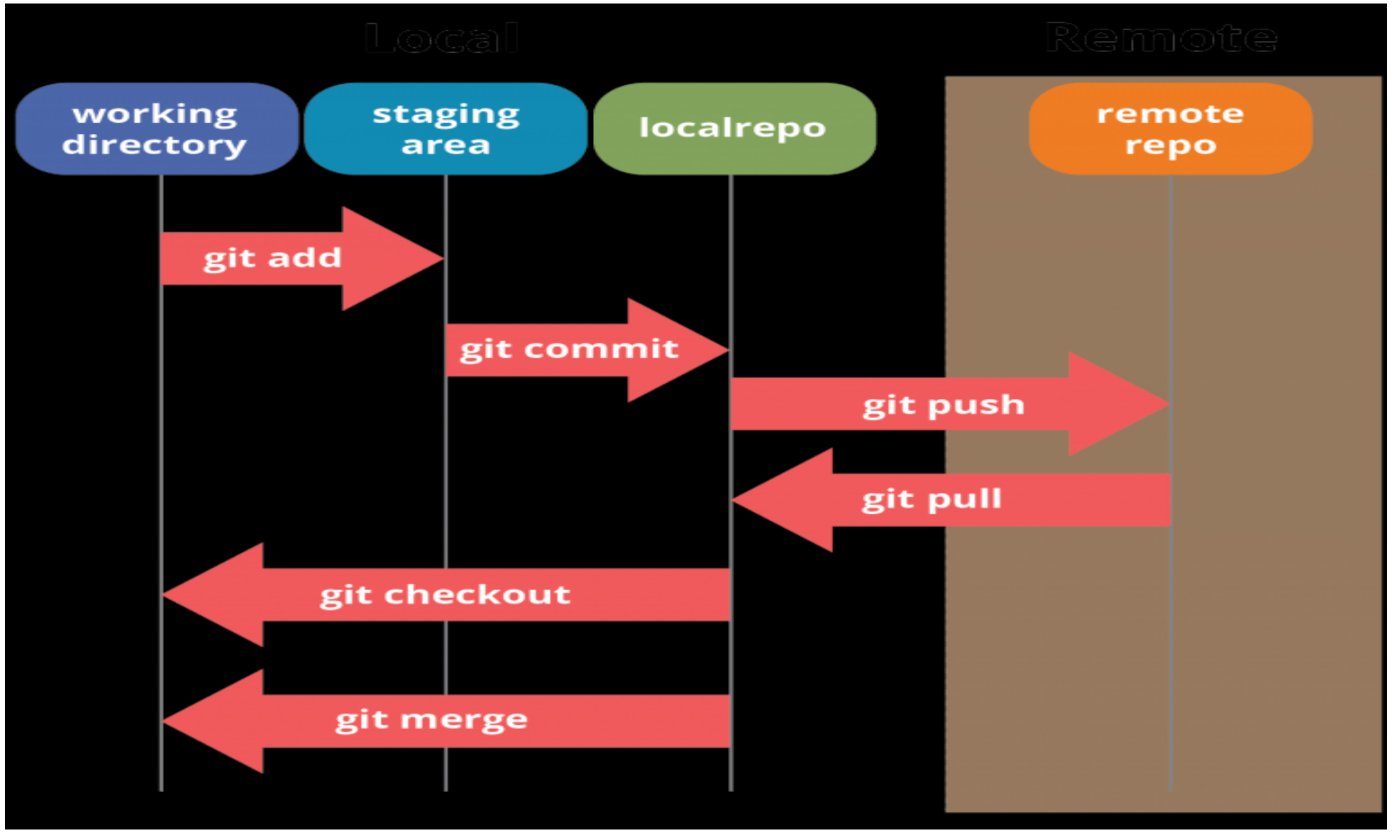
– (fix conflicts if necessary, add/commit them to your local repo)

#To fetch the most recent updates from the remote repo into your local repo, and put them into your working directory:

– git pull origin master

#To put your changes from your local repo in the remote repo:

– git push origin master



source: https://www.edureka.co/blog/git-tutorial/

Summary of individual work –initialize. ((code-stage-&-commit)$^+$. **pull** rebase.**push**)$^+$

- Create/initialize local repo to trace history of your work (i.e. git init)
- Start from master branch
- Stage and commit current uptodate files if available(i.e. git add, git commit)
- Branch out from master to feature branch - for registering new changes (i.e. git branch, git checkout)
- Make, stage, and commit changes in feature branch (i.e. git add, git commit).
- Merge feature branch into the master branch (i.e. git merge)
- Delete stale feature branch
- Sync with online repo (i.e. git pull, git push)
  - Git pull: fetch from a remote repo and try to merge into the current branch
  - Git push : push your new branches and data to a remote repository
- Pick a local repo→take snapshot of current work →make changes → record changes→ synchronize with remote repository

Working in a team

Create online copy of main repository then download

#fork existing central online repository

Forked is nothing more than a server side git clone

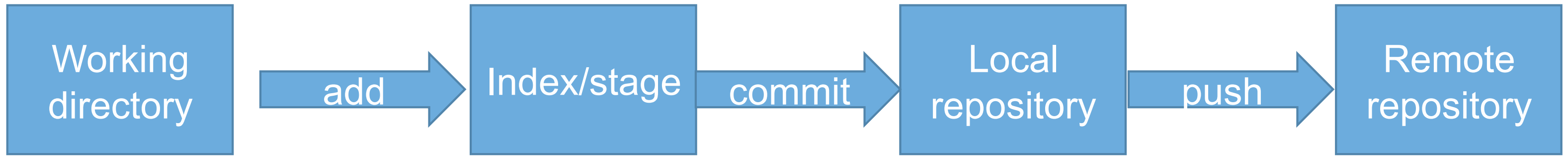
#download forked repository to local machine

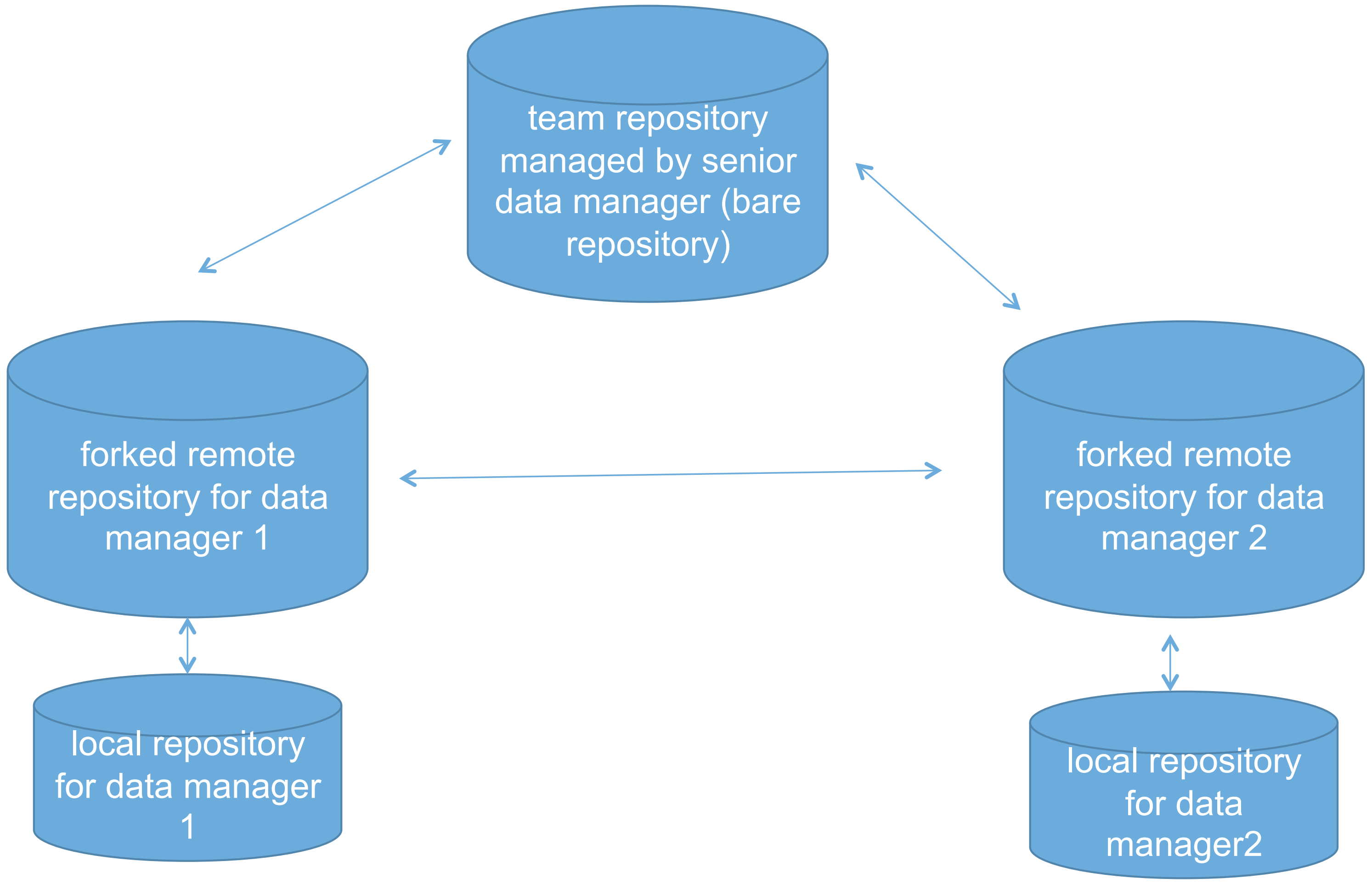
pwd #print/display the current working directory

cd path/to/workspace

git clone /path/to/forked-repository

#git maintains three local “trees” and synchronizes with remote “tree”




team repository managed by senior data manager (bare repository)
forked remote repository for data manager 1
forked remote repository for data manager 2
local repository for data manager 1
local repository for data manager2

Create a branch to track “experimental” changes to the workspace

# ask git to create a new feature branch out of the master branch

git branch <feature_x>

#ask git to display the list of branches and indicate current/active branch

git branch

# ask git to switch/checkout to the new branch

git checkout

Make and record more changes in the experimental/feature branch

#change code in working directory:

- Create new file
- Modify existing file
  - change contents
  - change name/ rename
  - change location/move
- Delete existing file

#ask git to record the changes

git add

git commit

#ask git to display status of files and folders in workspace

git status

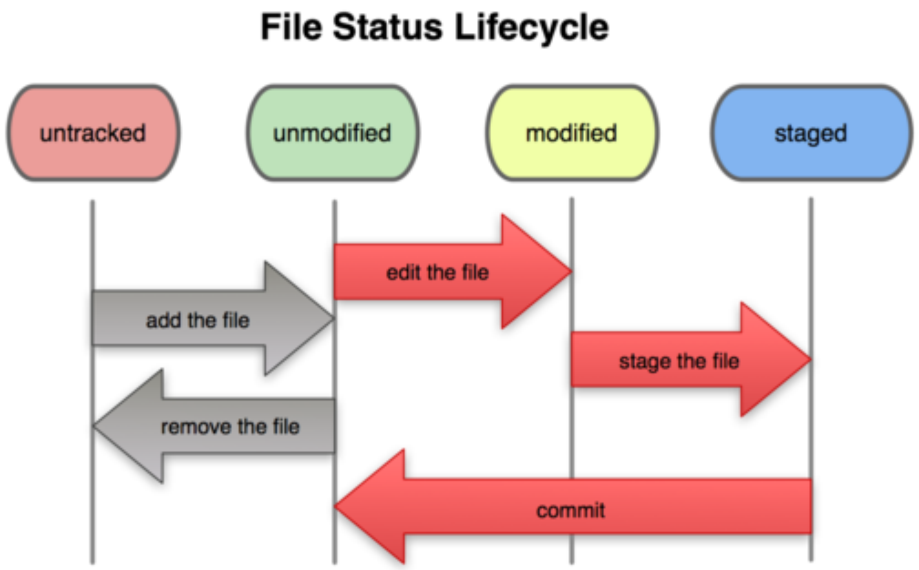
#compare the two branches

git checkout main

#list files in working directory

git checkout <feature_x>

#list files in working directory



Merge feature branch to the master branch

#ask git to switch/checkout to master branch: git checkout master

#ask git to check whether current/active branch is master: git branch

#check current list of files in master: ls

#preview updates before merging: **git diff <source_branch> <target_branch>**

#merge feature branch **into** master branch: git merge <feature_x>

#check current list of files in master branch: ls

#display branching structure

~delete experiemental branch if stale then display branching structure

***never commit to master because master will be the branch that you pull upstream changes from**

*Use git pull –rebase to move all of your commits to the tip of the history.

Asking git for information

#to view status of files in working directory and staging area:

git status or git status –s (short version)

#to see what is modified but unstaged:

git diff

#to see a list of staged changes:

git diff --cached

#to see a log of all changes in your local repo:

git log or git log --oneline (shorter version)

git log -2 (to show only the 2 most recent updates)

Synchronize remote and local repository

#Push your local changes to the remote repo.

#Pull from remote repo to get most recent changes.

– (fix conflicts if necessary, add/commit them to your local repo)

#To fetch the most recent updates from the remote repo into your local repo, and put them into your working directory:

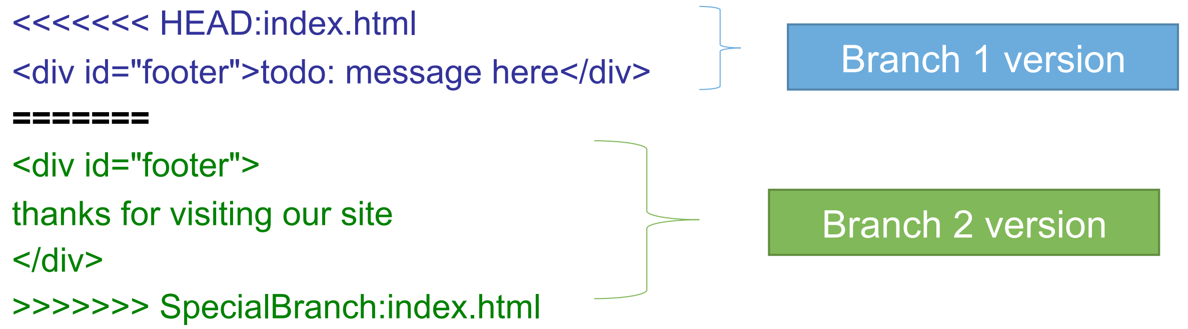
– git pull origin master

#To put your changes from your local repo in the remote repo:

– git push origin master

Handle conflicts during merging

- The conflicting file will contain <<< and >>> sections to indicate where git was unable to resolve a conflict:



```
<<<<<<< HEAD:index.html
<div id="footer">todo: message here</div>
=======
<div id="footer">
thanks for visiting our site
</div>
>>>>>>> SpecialBranch:index.html
```

- Find all such sections, and edit them to the proper state (whichever of the two versions is newer / better / more correct)
- When done, stage then commit before merging then uploading

Handle conflicts during merging

```
#ask git to fetch updates from remote repository and display downloaded branches
git fetch origin
#ask git to display log of commits in remote master branch
git log --oneline master..origin/master
#ask git to switch to master branch
git checkout master
#ask git to merge changes from remote master
git merge origin/master
#git fetch then git rebase is better than git fetch then git merge
```

Request senior data manager to review and accept your work

Pull Requests initiate discussion about your commits. Pull Requests also provide a way to notify project maintainers about the changes you'd like them to consider

1. On GitHub, go to the repository from which you would like to propose changes...team repository
2. In the “Branch” menu, choose the branch that contains your commits
3. To the left of the “Branch” menu, click the green Compare and Review button
4. The Compare page will automatically select the base and compare branches; to change these, click edit
5. Click Create pull request. Title and describe your pull request. Create pull request
6. Proposed changes will be merged from the head branch to the base branch that was specified in the pull request
   - https://help.github.com/en/articles/creating-a-pull-request-from-a-fork
   - https://help.github.com/en/articles/merging-a-pull-request

Summary of teamwork –fork. clone. ((code-stage-&-commit)$^+$. rebase. push.pull request)$^+$

- Create online repo to back up your work
- Clone the repo (i.e. git clone)
- Start from master branch
- Stage and commit current uptodate files if available(i.e. git add, git commit)
- Branch out from master to feature branch - for registering new changes (i.e. git branch, git checkout)
- Make, stage, and commit changes. (i.e. git add, git commit).
- Merge feature branch into the master branch (i.e. git merge)
- Delete stale feature branch
- Sync with online repo (i.e. git pull – rebase origin master, git push origin master)
- forking gives a "clean" environment
- pull requests give a nice way to discuss code

Asking git for help

#get help info about a particular command git help ***[command]***

```
git help stash
```

Questions

Thank You!