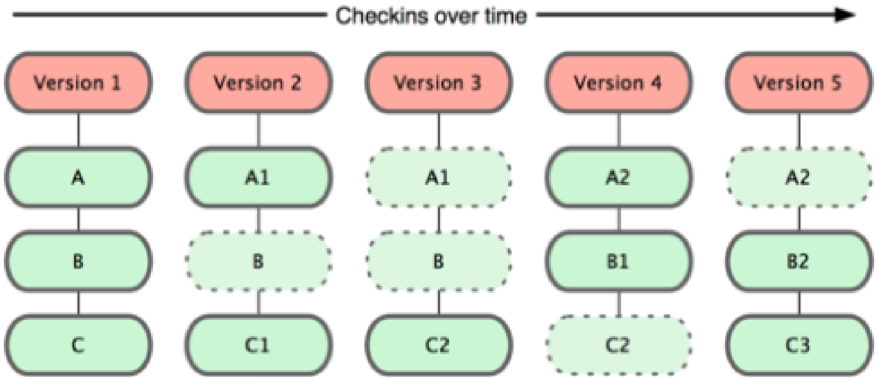
A distributed version control system

GIT

Version control systems

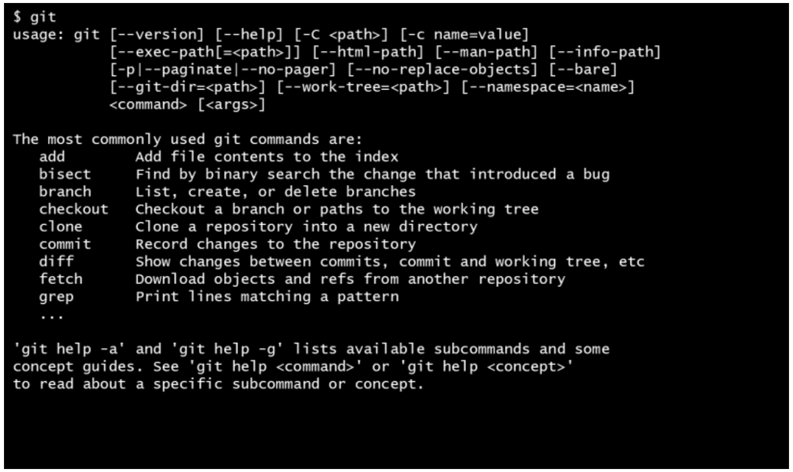
- Version control:
  - Managing multiple versions of documents, programs, etc.
  - For working by yourself:
    - "time machine" for going back to earlier versions
    - Support for different versions (standalone, web app, etc.) of the same basic project
  - For working with others:
    - Greatly simplifies concurrent work, merging changes
  - For getting an internship or job:
    - Any company with a clue uses some kind of version control
    - Companies without a clue are bad places to work

Git



Why Git?

- Git was created by Linus Torvalds in 2005
- You don't "checkout“ from a central repo:
  - you "clone" it and "pull" changes from it
- Your local repo is a complete copy of everything on the remote server
  - yours is "just as good" as theirs
- Many operations are local:
  - check in/out from local repo
  - commit changes to local repo
  - local repo keeps version history
- When you're ready, you can "push" changes back to server



```
$ git
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p|--paginate|--no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

The most commonly used git commands are:
   add        Add file contents to the index
   bisect     Find by binary search the change that introduced a bug
   branch     List, create, or delete branches
   checkout   Checkout a branch or paths to the working tree
   clone      Clone a repository into a new directory
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   fetch      Download objects and refs from another repository
   grep       Print lines matching a pattern
   ...

'git help -a' and 'git help -g' lists available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
```

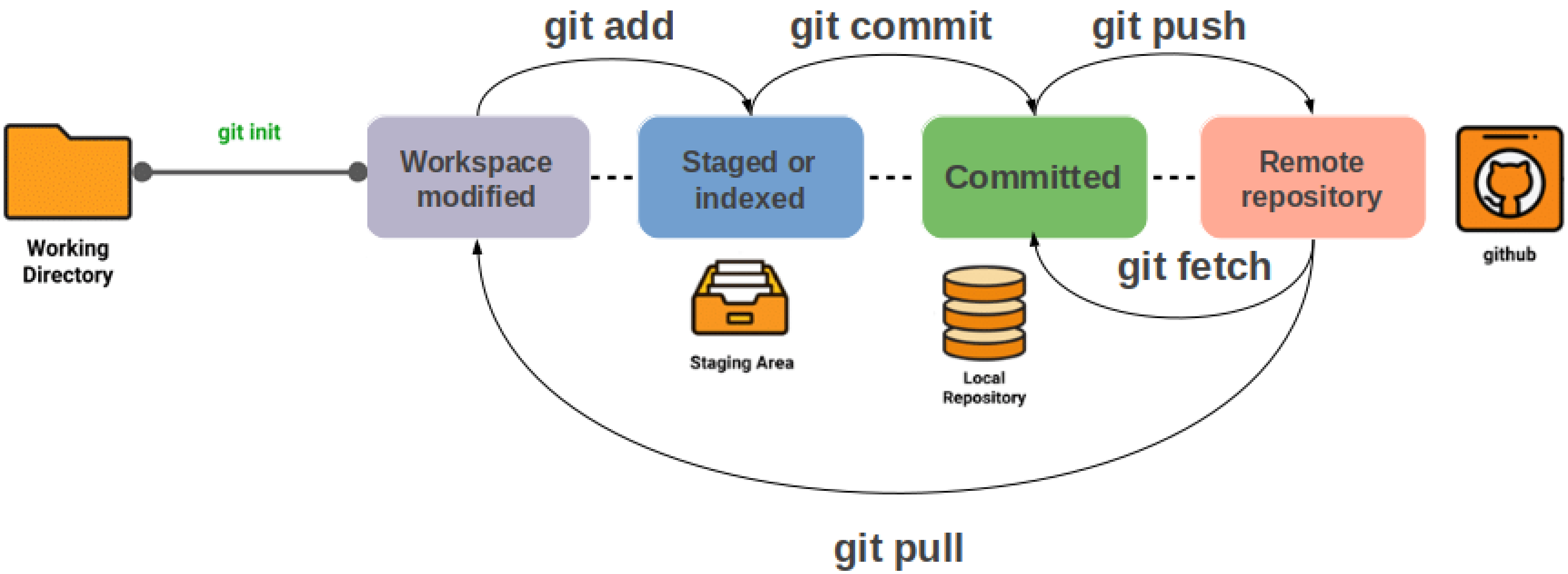
- Git is primarily a command-line tool
- The Git GUIs are more trouble than they are worth

Download and install Git

- Online materials
  - Mac OS X
    - http://sourceforge.net/projects/git-osx-installer
    - brew install git
  - Linux
    - apt-get install git (debian/ubuntu)
    - yum install git (fedora/redhat)
  - Standard one:
    - http://git-scm.com/downloads
  - SackExchange:
    - http://stackoverflow.com/questions/315911/git-for-beginners-the-definitive-practical-guide#323764

Git cycle life

- Cycle life



Let’s create your first GIT

- Step 1 : Configure git

Global configuration

- git config -l
- git config --global user.name “Ismail Berrada“
- git config --global user.email ismail.berrada@um6p.ma
- git config --global core.editor nano

Working project configuration

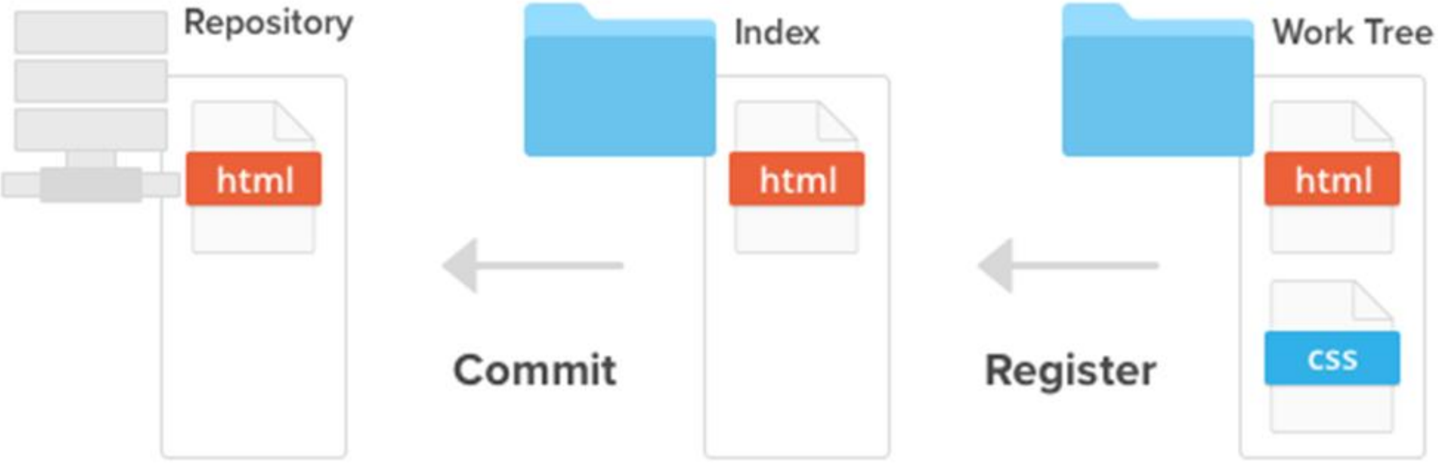
- cd to the project directory
- Leave out the –global

git config --list to verify the config

- Step 2 : Create a local repository
  - **cd to the project directory**
  - **Type in git init**
    - Create the repository .git containing various files (a “hidden” directory)
    - You do *not* work directly with the contents of that directory; various git commands do that for you
    - You *do* need a basic understanding of what is in the repository

Let’s create your first GIT

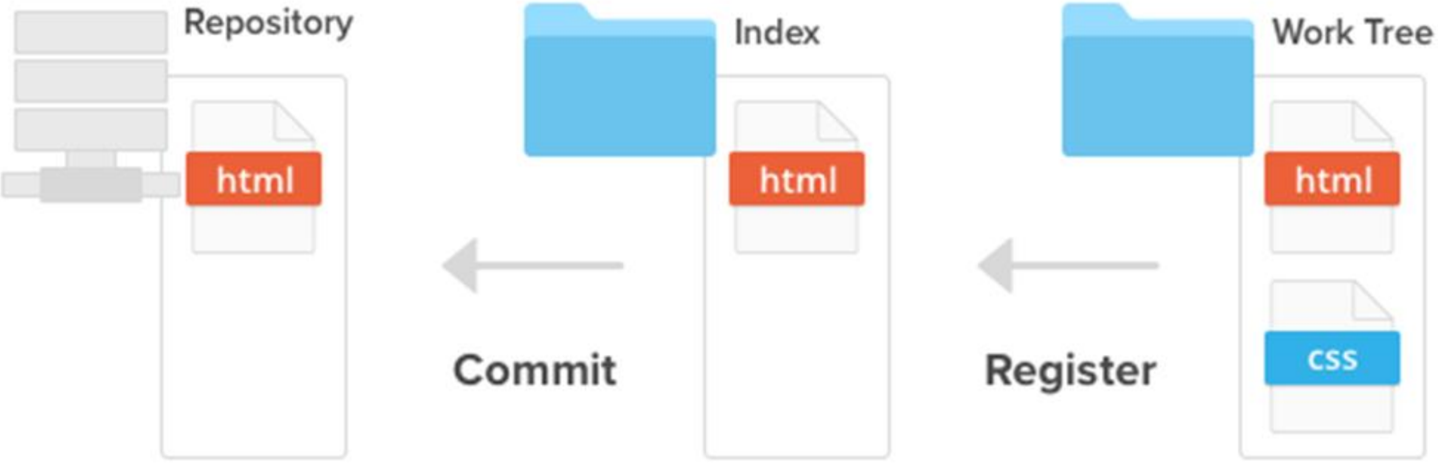
- Step 3: Fill the local Index
  - Do the work as usual coding. If you create new files and/or folders, they are not tracked by Git unless you ask it to do so
    - **git add newFile1 newFolder1 newFolder2 newFile2**
    - **Or type in git add .**
      - dot means “this directory”: track all your current files.
      - We said that the files are staged.



Files not registered in the index cannot be committed.

Let’s create your first GIT

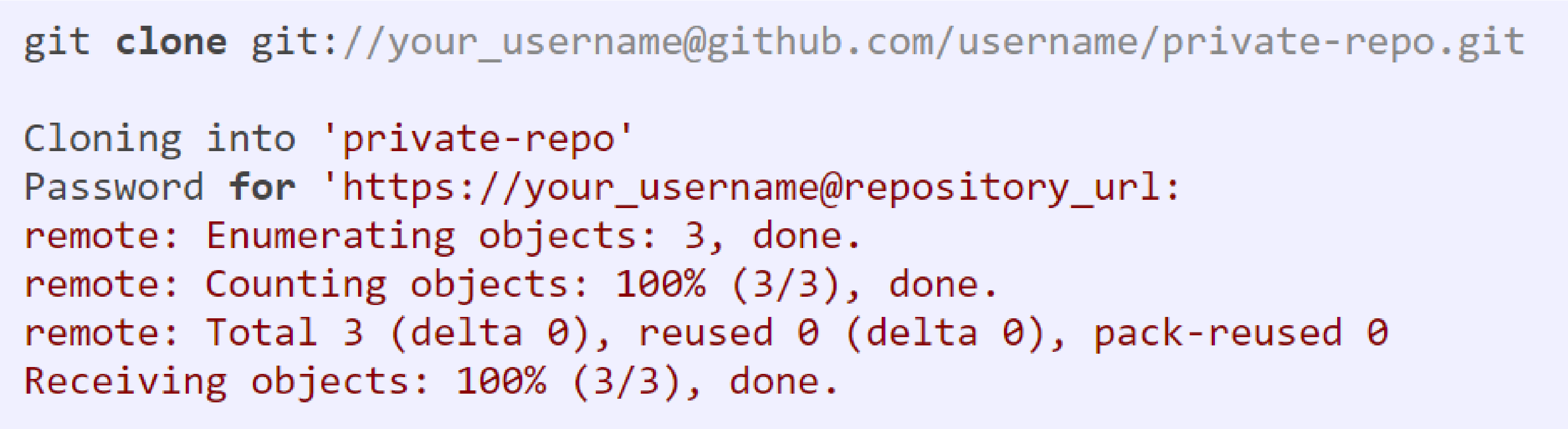
- Step 4 : Commit to the local repository
  - Make a “snapshot” of everything being tracked into your repository. A message telling what you have done is required
    - **Type in git commit –m "Initial commit“**
    - **Or type git commit**
      - It opens an editor, enter the message, save and quit.
    - In git, “Commits are cheap”, do them often.



Files not registered in the index cannot be committed.

Let’s create your first GIT

- Step 5 : Clone a remote repository
  - You can clone a remote repo to your current directory:
    - **git clone https:// github.com/username/mygit.git**
      - This will create the given local directory, containing a working copy of the files from the repo, and a .git directory (used to hold the staging area and your actual local repo)
    - **git clone git://your_username@github.com/username/private-repo.git**



```
git clone git://your_username@github.com/username/private-repo.git

Cloning into 'private-repo'
Password for 'https://your_username@repository_url:
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

Recapitulation

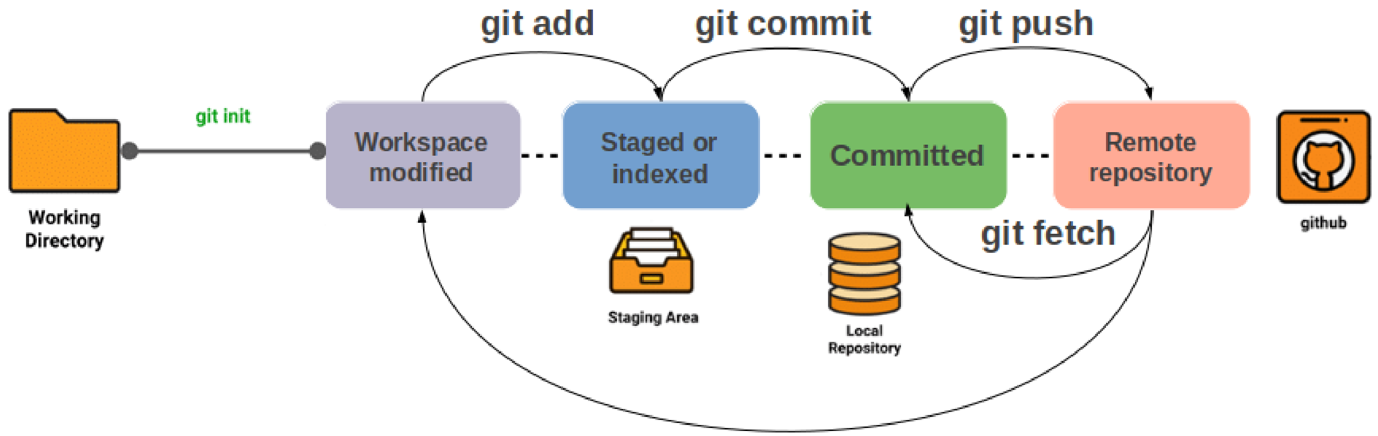
- Cycle life

Top-level **working directory:**

- One of these subdirectories, named .git, is your **repository**: a (key = object) database

“snapshot” of your project directory, and put it in your repository is called a commit object that contains

- (1) a set of files, (2) references to the “parents” of the commit object, and (3) a unique “SHA1” name



Work as much as you like, but the working repository isn’t updated until you commit

release is the distribution of a given changest of repository.. version of release corresponding to a given commit

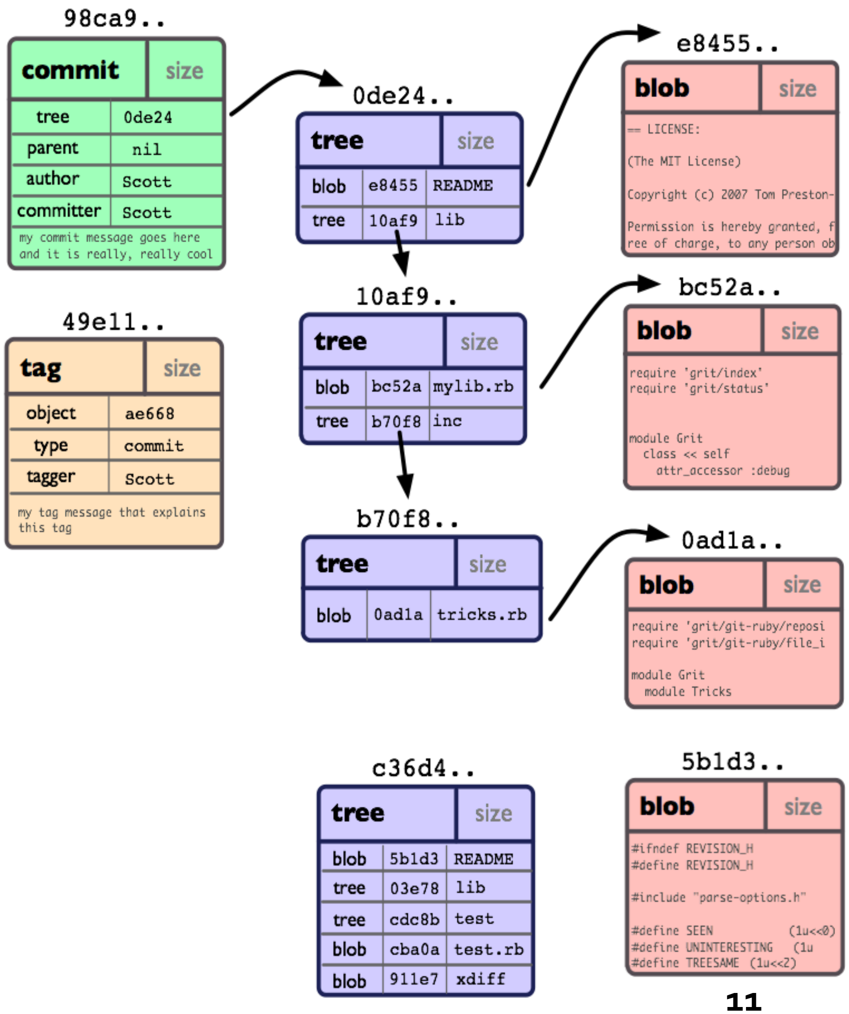
Git internal storage

- Object model

All the information needed to represent the history of a project is stored in files referenced by a SHA1 40-digit: **"object name"**

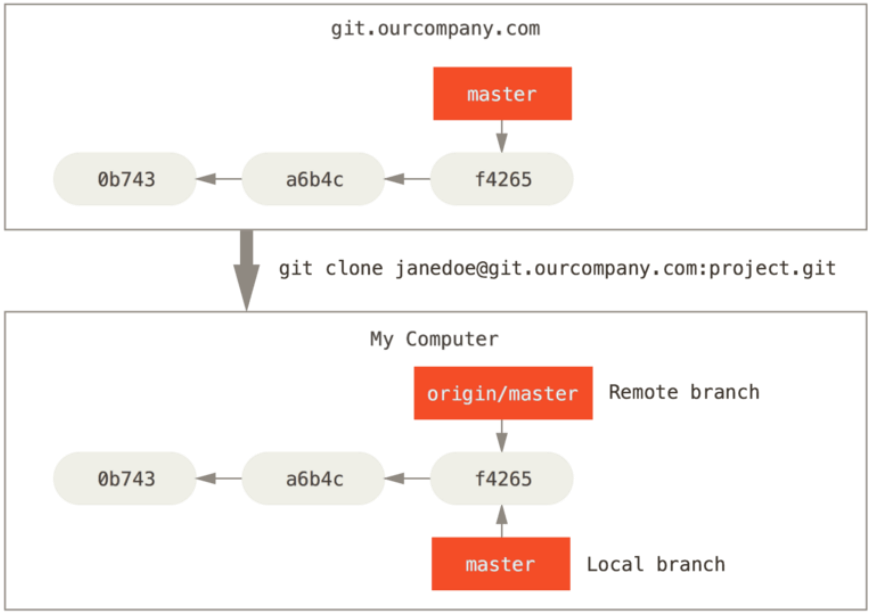
- Object: a **type**, a **size** and **content**.
  - **"blob"**: to store file data
  - **"tree"**: like a directory - it references a bunch of other trees and/or blobs (i.e. files and sub-directories)
  - **"commit"**: a single tree, the project looked like at a certain point in time.
  - **"tag"**: mark a specific commit as special in some way (special version, ..).

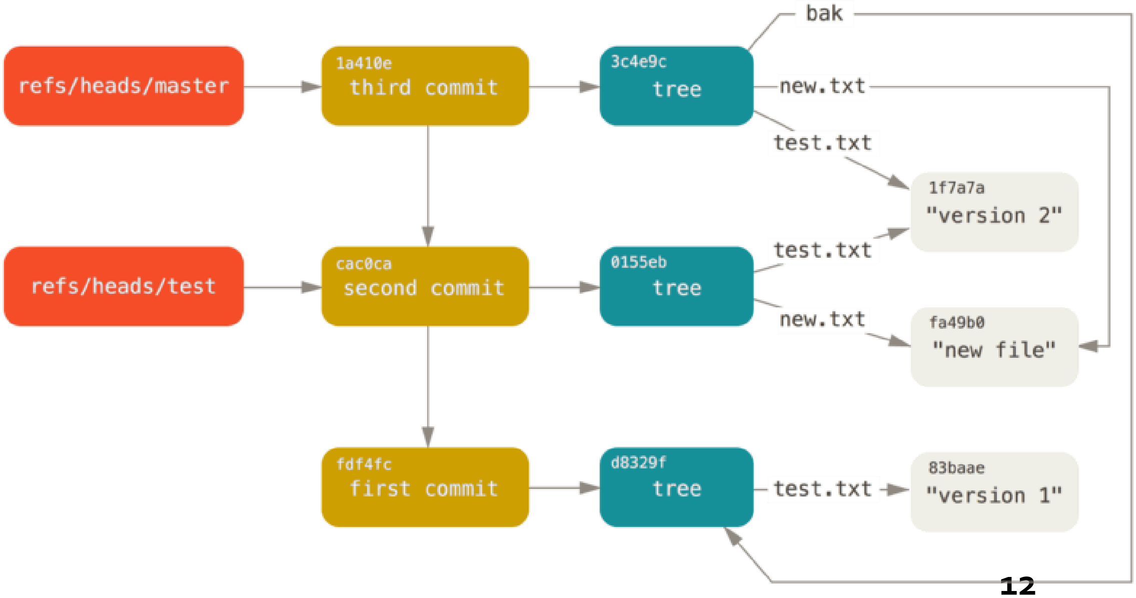
A commit object represents the complete state of the project



Git internal storage

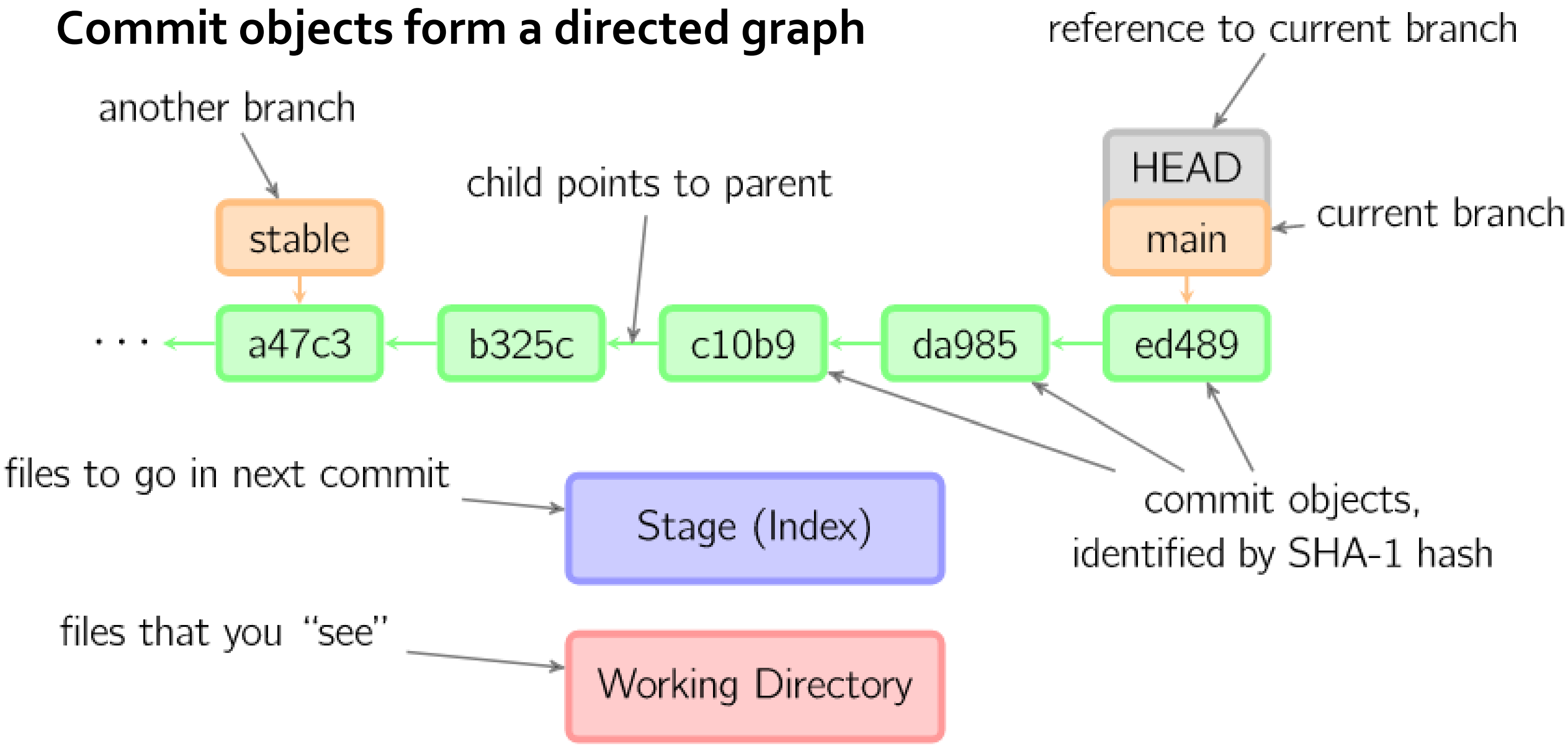
- Git references
  - Git keeps the history of your repository reachable from commit, in files called “references” or “refs”.
  - You can find the files that contain those SHA-1 values (40 character string of hex digits) in the .git/refs directory. Often we only see the first 7 characters: 0e52da7 Initial commit





Local git area

- Git commit graph



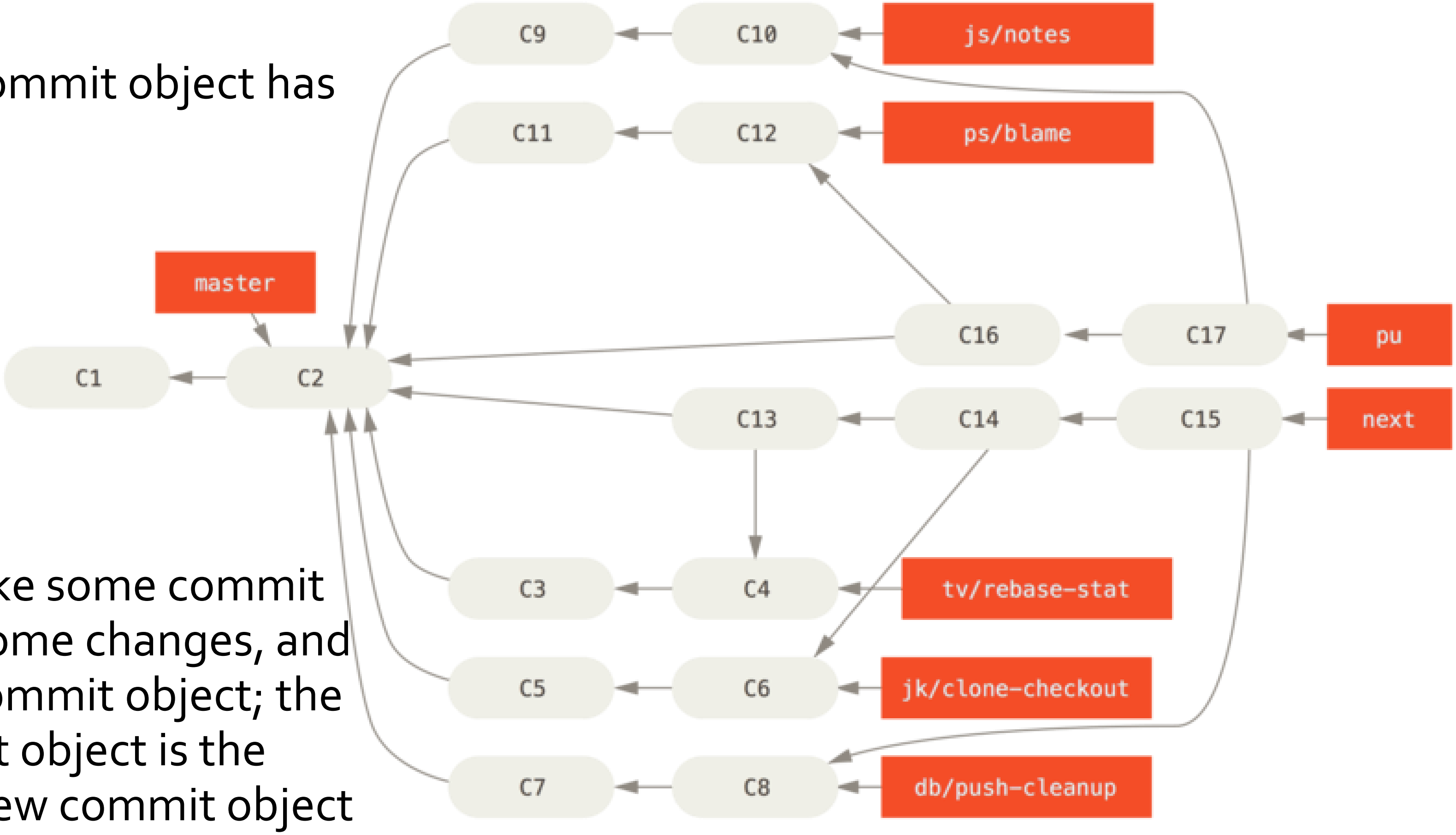
- A head is a reference to a commit object. The "current head" is called HEAD. Usually, you will take HEAD (the current commit object), make some changes to it, and commit the changes, creating a new current commit object.
- Each user has their own copy of the repo, and commits changes to their local copy of the repo before pushing to the server.

Local git area

- Git commit graph (more complex)

- The *very first* commit object has no "parents"

- Usually, you take some commit object, make some changes, and create a new commit object; the original commit object is the parent of the new commit object



You can also merge two commit objects to form a new one

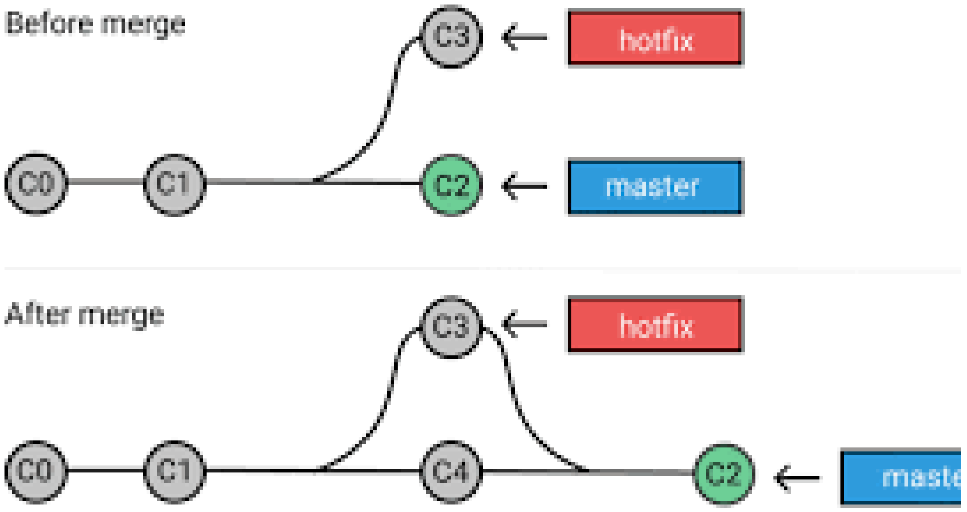
- The new commit object has two parents

Local git area

- merge

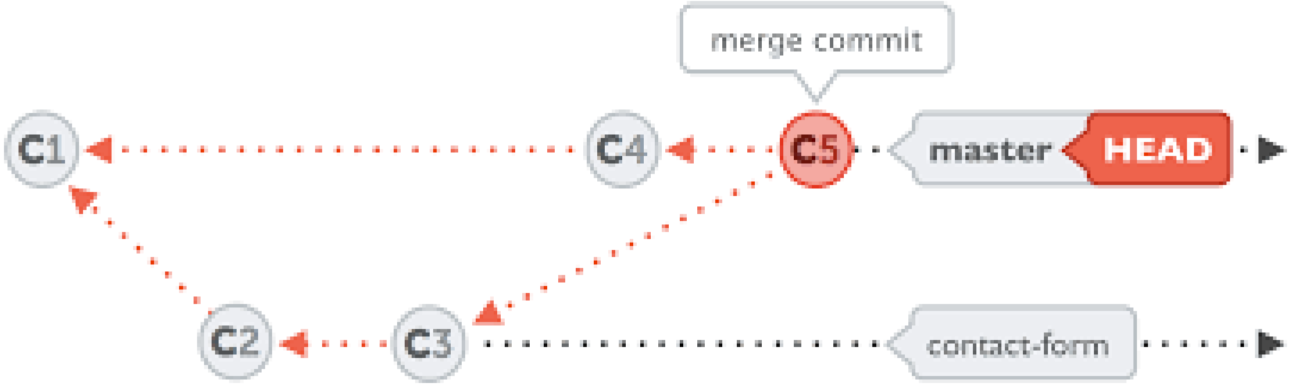
You can also take any previous commit object, make changes to it, and commit those changes

- This creates a branch in the graph of commit objects



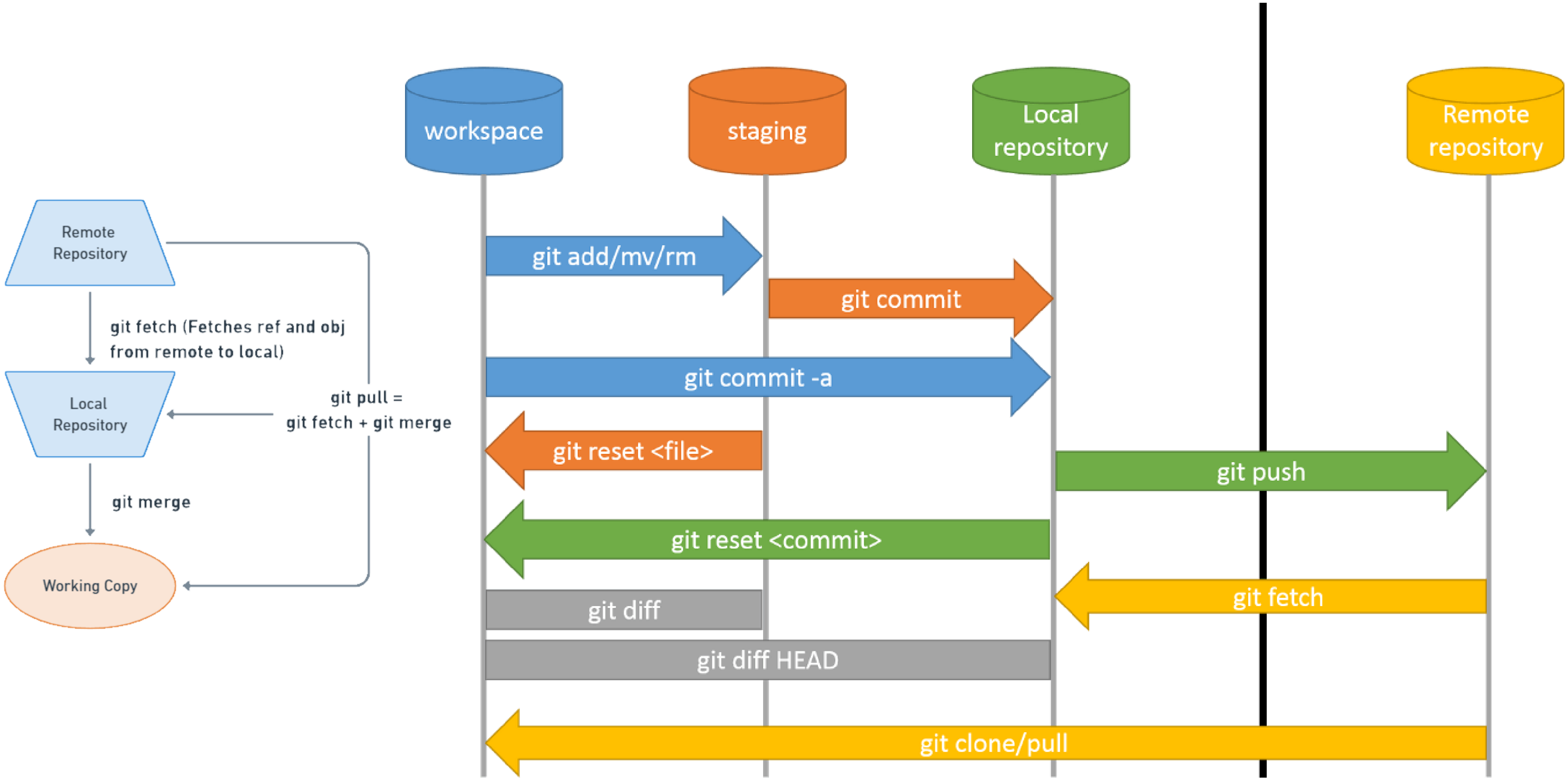
You can merge any previous commit objects

- This joins branches in the commit graph



Git workflow

- Local vs remote



Remote Repository

git fetch (Fetches ref and obj from remote to local)

Local Repository

git pull = git fetch + git merge

git merge

Working Copy

workspace

staging

Local repository

Remote repository

git add/mv/rm

git commit

git commit -a

git reset <file>

git push

git reset <commit>

git fetch

git diff

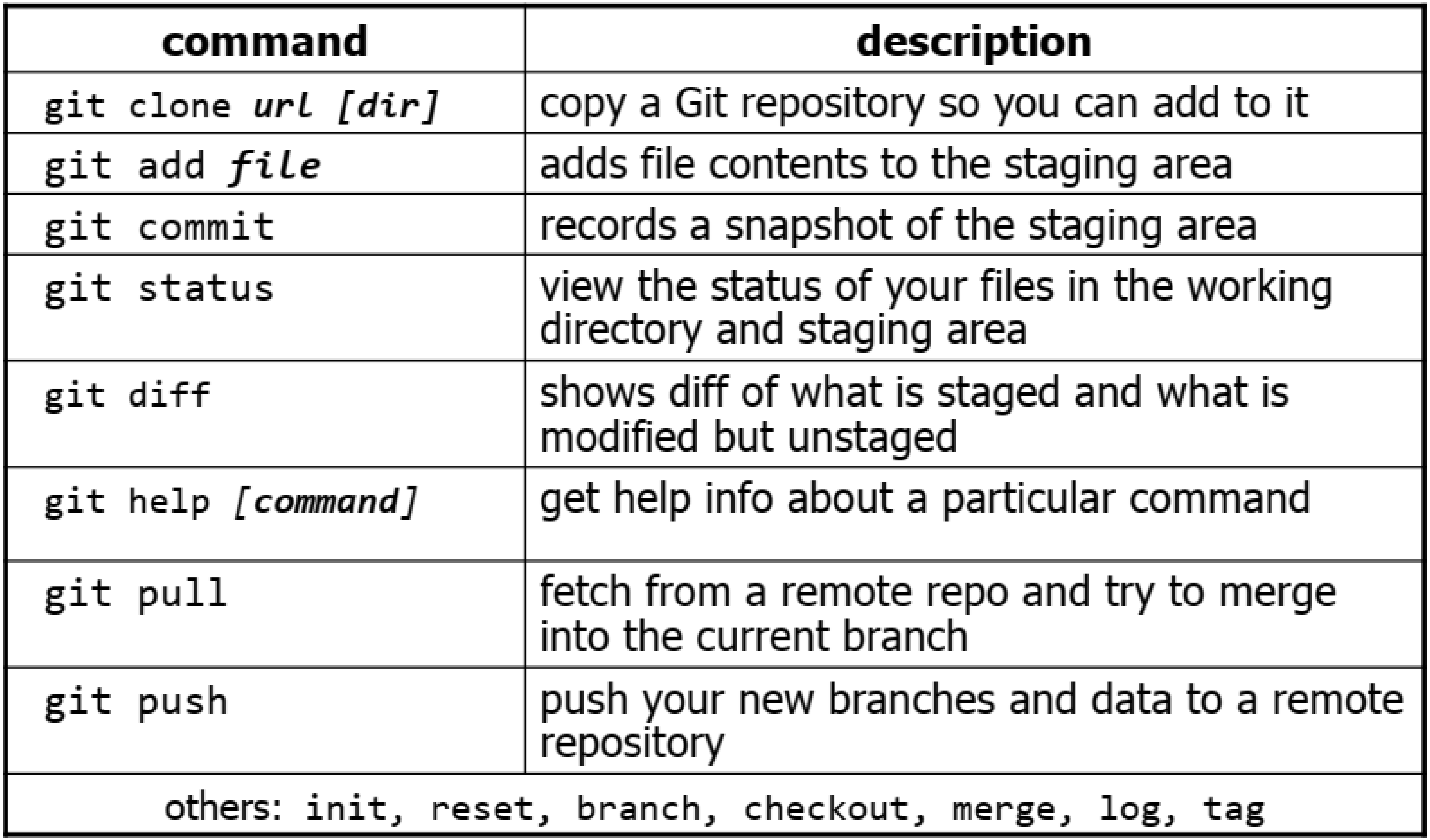
git diff HEAD

git clone/pull

Git is all about using and manipulating the commit graph

Local git area

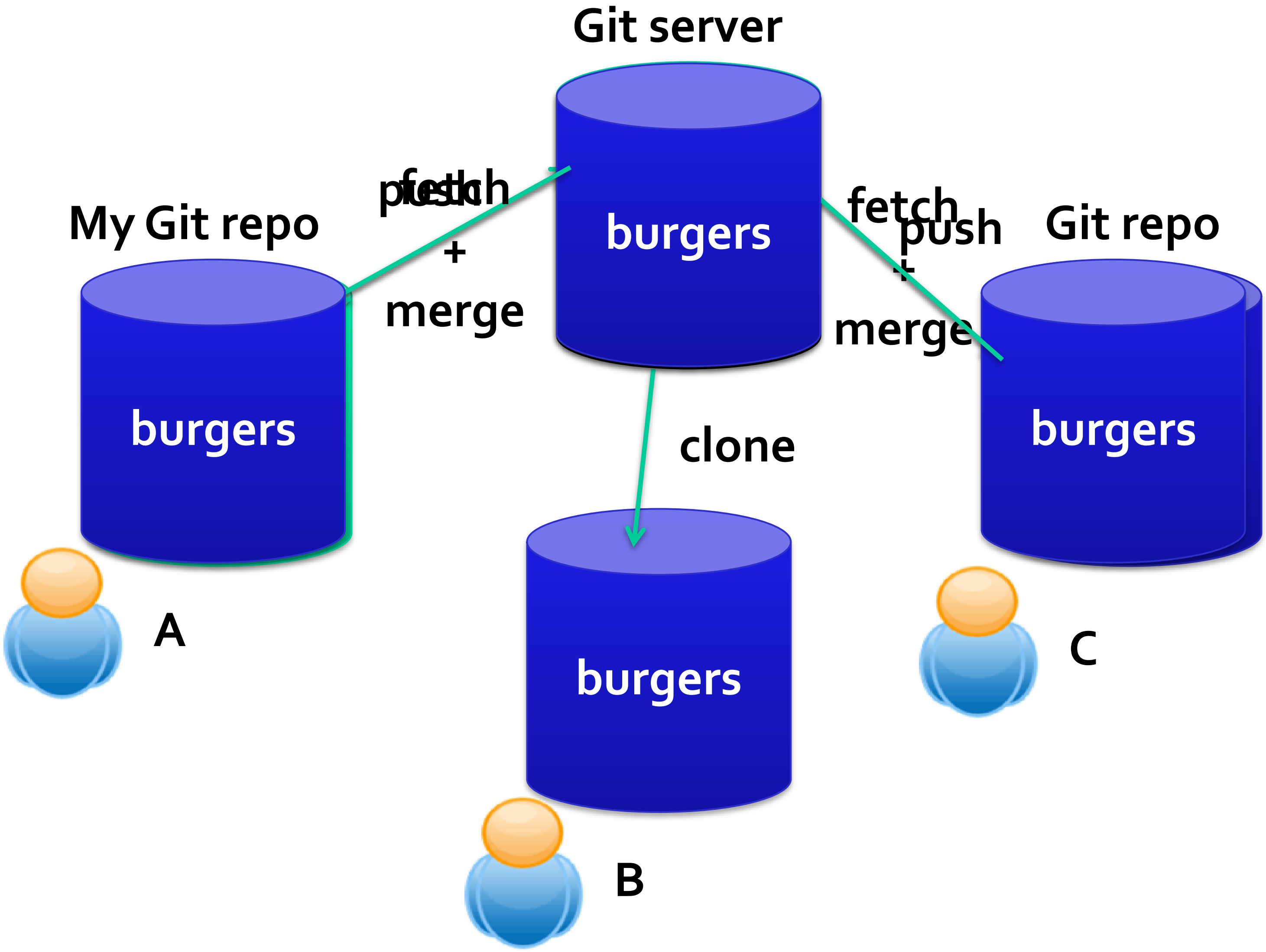
- Git commands



command	description
`git clone` *`url [dir]`*	copy a Git repository so you can add to it
`git add` *`file`*	adds file contents to the staging area
`git commit`	records a snapshot of the staging area
`git status`	view the status of your files in the working directory and staging area
`git diff`	shows diff of what is staged and what is modified but unstaged
`git help` *`[command]`*	get help info about a particular command
`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
others: `init, reset, branch, checkout, merge, log, tag`	

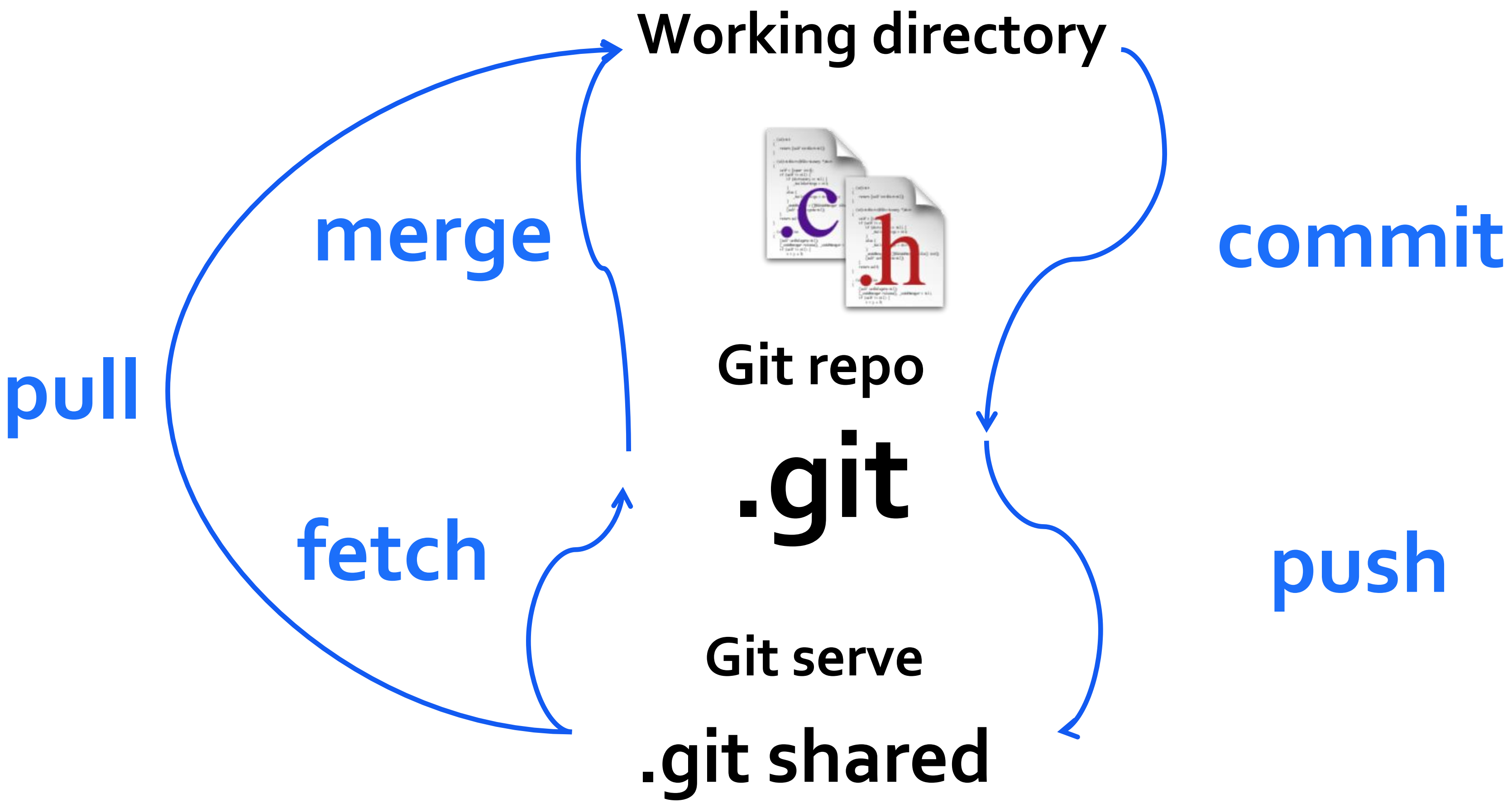
Working with others

- Cycle



Working with others

- Cycle



Working with others

- Clone a repository from elsewhere
  - git clone *URL*
  - git clone *URL mypath*
    - These make an exact copy of the repository at the given URL
  - git clone git://github.com/*rest_of_path/file.git*
    - Github is the most popular (free) public repository
- All repositories are equal
  - But you can treat some particular repository (such as one on Github) as the "master" directory
- Typically, each team member works in his/her own repository, and "merges" with other repositories as appropriate

Working with others

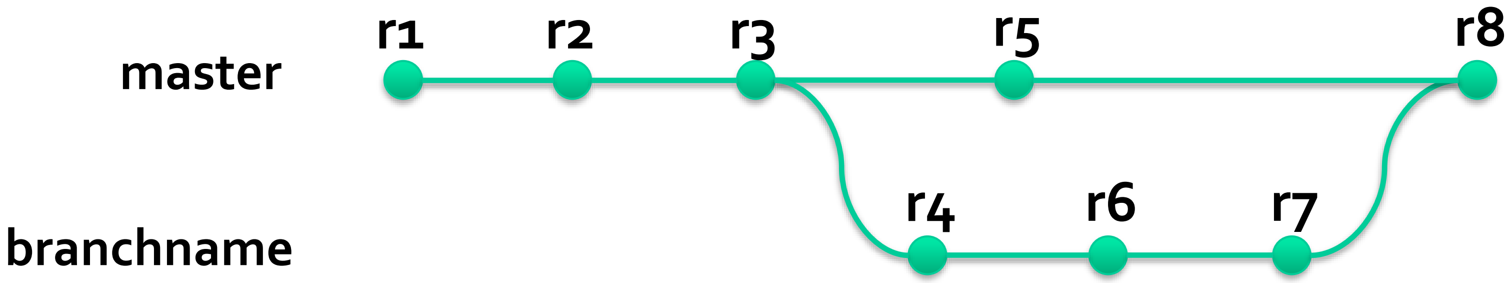
- All repositories are equal, but it is convenient to have one central repository in the cloud
- Here's what you normally do:
  - Download the current HEAD from the central repository
  - Make your changes
  - Commit your changes to your local repository
  - Check to make sure someone else on your team hasn't updated the central repository since you got it
  - Upload your changes to the central repository
- If the central repository *has* changed since you got it:
  - It is *your* responsibility to **merge your two versions**
    - This is a strong incentive to commit and upload often!
  - Git can often do this for you, if there aren't incompatible changes

Working with others

- Typical workflow
  - git pull *remote_repository*
    - Get changes from a remote repository and merge them into your own repository
  - git status
    - See what Git thinks is going on
    - Use this frequently!
  - Work on your files (remember to add any new ones)
  - git commit –m "*What I did*"
  - git push

Branches and merging

- Typical workflow

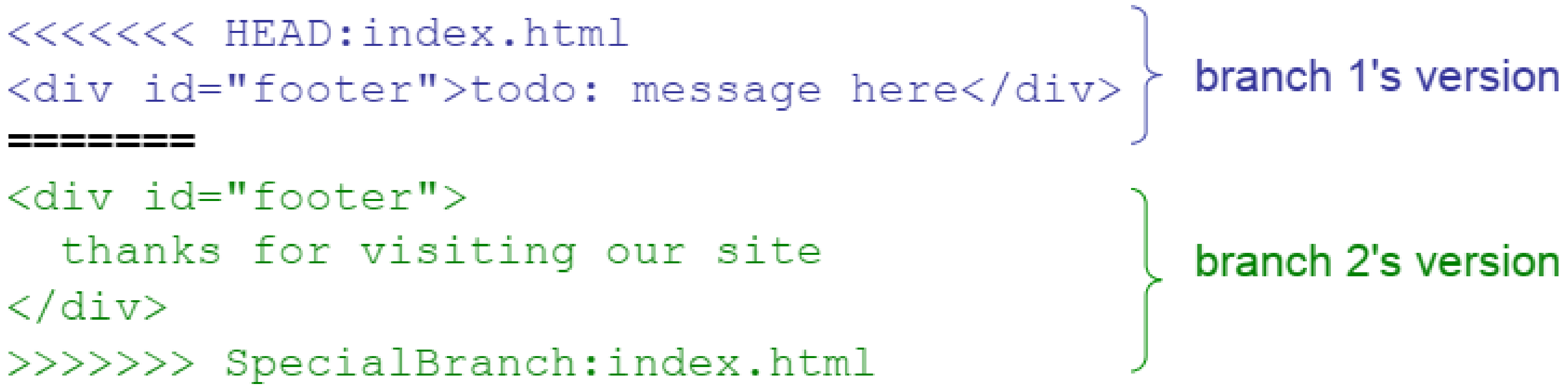


Branches and merging

- Git uses branching heavily to switch between multiple tasks.
  - To create a new local branch:
    - git branch *name*
  - To list all local branches: (* = current branch)
    - git branch
  - To switch to a given local branch:
    - git checkout *branchname*
  - To merge changes from a branch into the local master:
    - git checkout master
    - git merge *branchname*

Merging conflict

- 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)

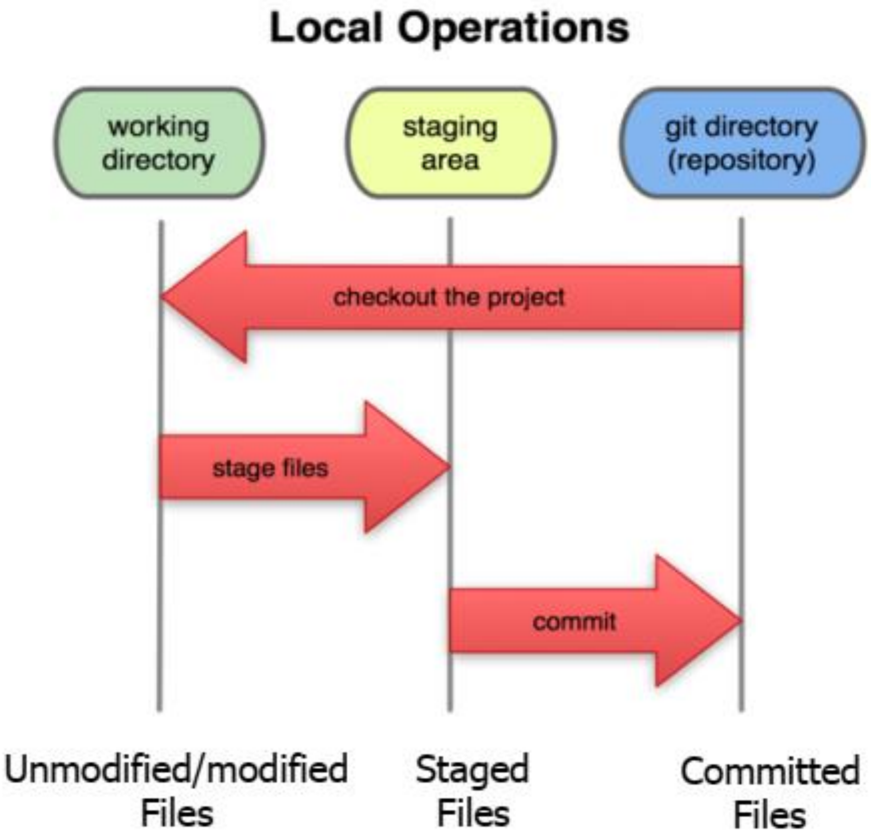
Some advises

- If you:
  - Make sure you are current with the central repository
  - Make some improvements to your code
  - Update the central repository before anyone else does
- Then you don't have to worry about resolving conflicts or working with multiple branches
  - All the complexity in git comes from dealing with these
- Therefore:
  - Make sure you are up-to-date before starting to work
  - Commit and update the central repository frequently
- If you need help: https://help.github.com/

Backup

Local git area

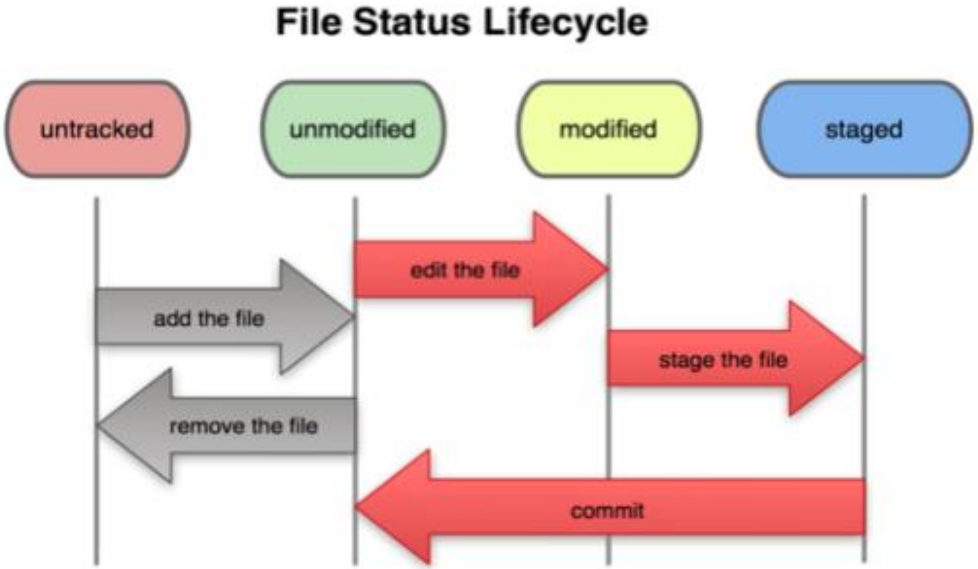
- In your local copy on git, files can be:
  - In your local repo (committed)
  - Checked out and modified, but not yet committed (working copy)
  - Or, in-between, in a **"staging" area**



- Staged files are ready to be committed.
- A commit saves a snapshot of all staged state.

Local git area

- Basic git workflow
  - Modify files in your working directory.
  - Stage files, adding snapshots of them to your staging area.
  - Commit, which takes the files in the staging area and stores that snapshot permanently to your Git directory.



Local git area

- When you commit your change to git, it creates a commit object:
  - A commit object represents the complete state of the project, including all the files in the project
  - The *very first* commit object has no "parents"
  - Usually, you take some commit object, make some changes, and create a new commit object; the original commit object is the parent of the new commit object
    - Hence, most commit objects have a single parent
  - You can also merge two commit objects to form a new one
    - The new commit object has two parents
- Commit objects form a **directed graph**
  - Git is all about using and manipulating this graph

Local git area

- Working with your repository
- A head is a reference to a commit object
- The "current head" is called HEAD (all caps)
- Usually, you will take HEAD (the current commit object), make some changes to it, and commit the changes, creating a new current commit object
  - This results in a linear graph: A → B → C → …→ HEAD
- You can also take any previous commit object, make changes to it, and commit those changes
  - This creates a branch in the graph of commit objects
- You can merge any previous commit objects
  - This joins branches in the commit graph