Git

About Git

- Created by Linus Torvalds, creator of Linux, in 2005
  – Came out of Linux development community
  – Designed to do version control on Linux kernel

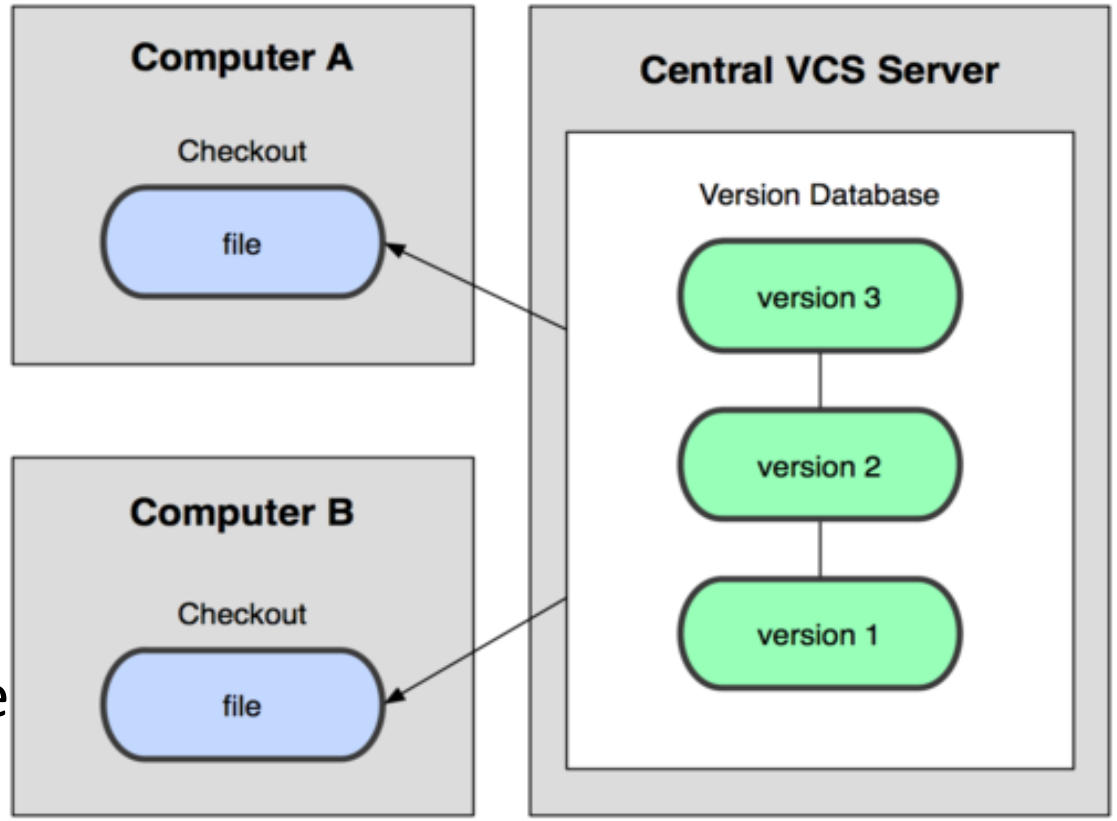
- Goals of Git:
  – Speed
  – Support for non-linear development (thousands of parallel branches)
  – Fully distributed
  – Able to handle large projects efficiently
  – (A "git" is a cranky old man. Linus meant himself.)

Centralized VCS

- In Subversion, CVS, Perforce, etc.
  A central server repository (repo)
  holds the "official copy" of the code
  – the server maintains the sole version history of the repo

- You make "checkouts" of it to your local copy
  – you make local modifications
  – your changes are not versioned

- When you're done, you "check in" back to the server
  – your checking increments the repo's version



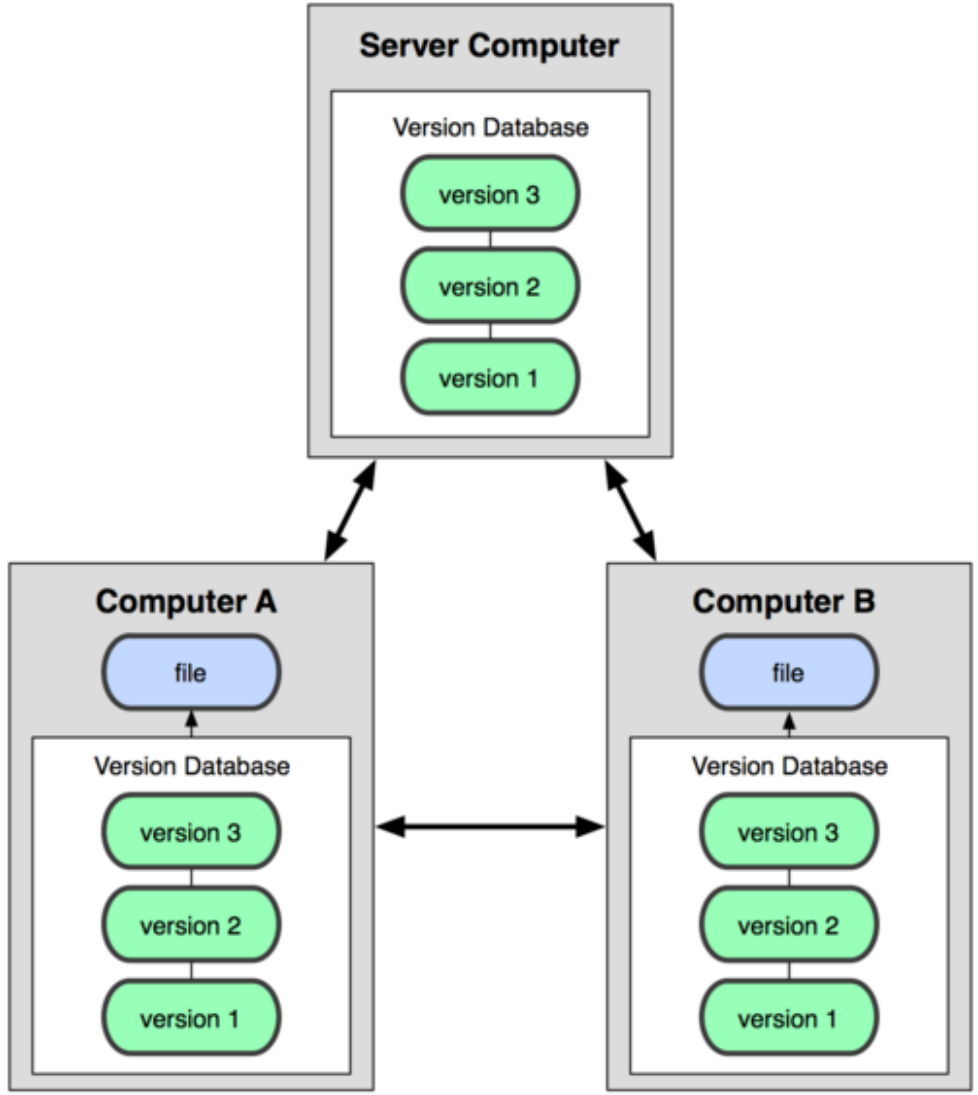
Distributed VCS (Git)

- In git, mercurial, etc., 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

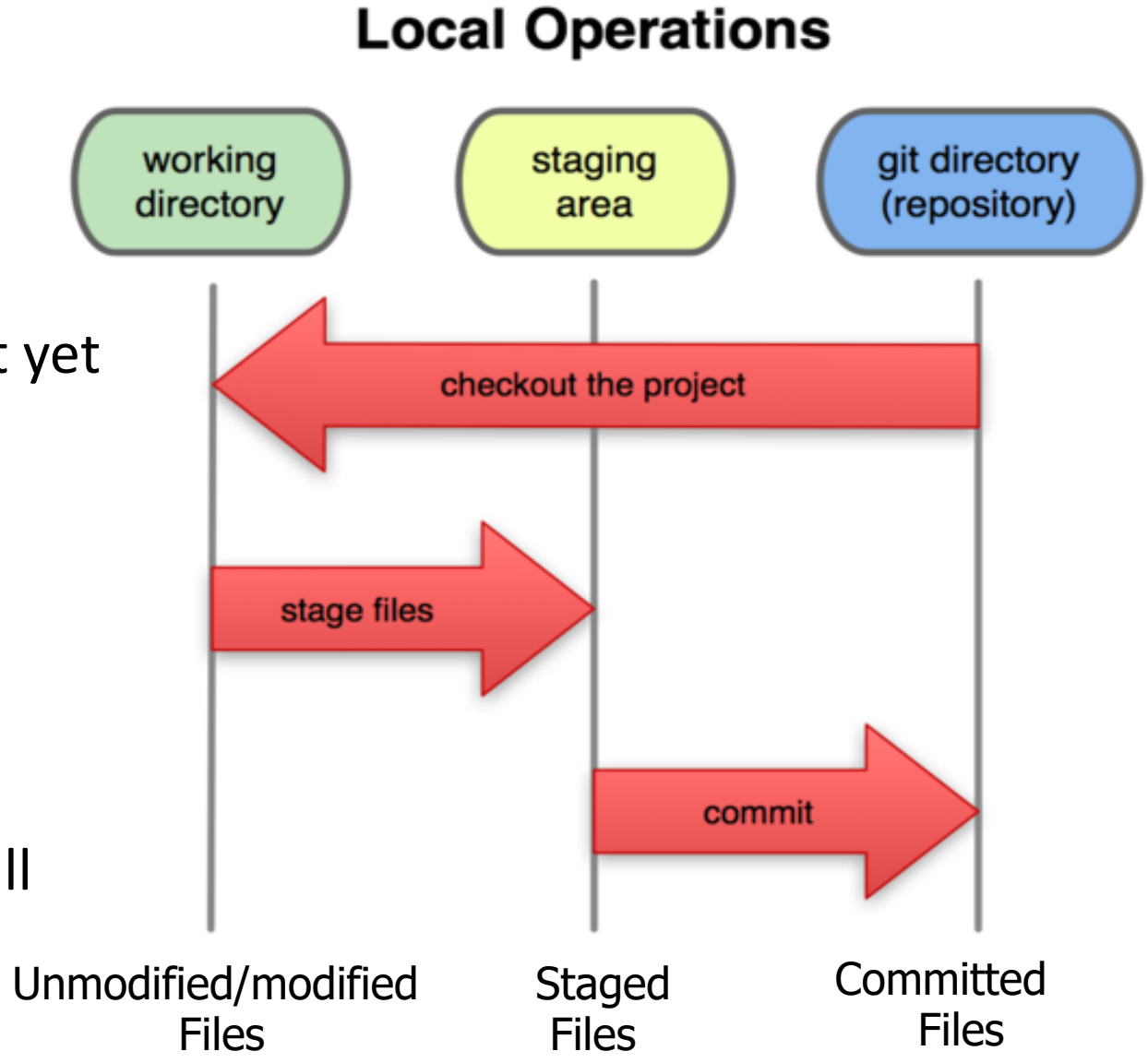
- Many operations are local:
  - check in/out from local repo
  - commit changes to local repo
  - local repo keeps version history

- When you're ready, you can "push" changes back to server



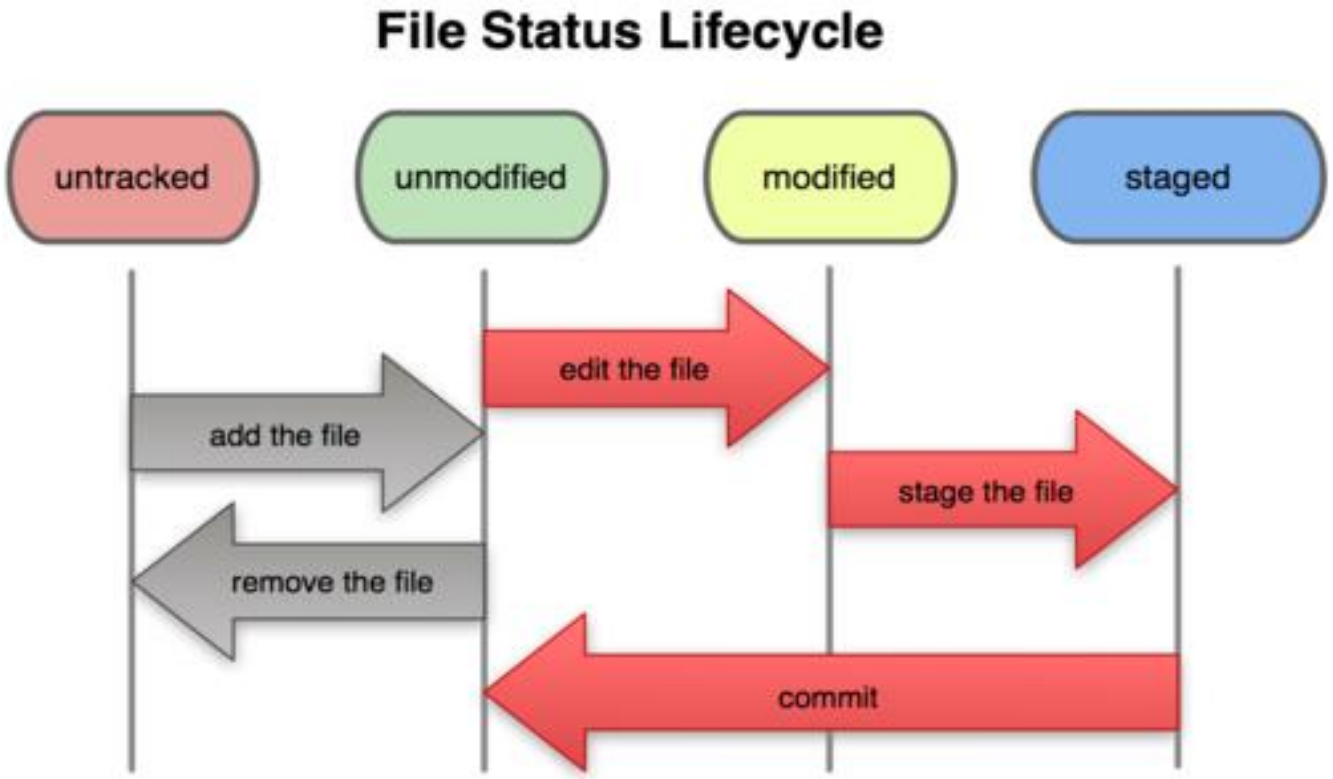
Local git areas

- 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.



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.



Initial Git configuration

- Set the name and email for Git to use when you commit:

```
– git config --global user.name "Bugs Bunny"
– git config --global user.email bugs@gmail.com
```

– You can call git config –list to verify these are set.

- Set the editor that is used for writing commit messages:

```
– git config --global core.editor nano • (it is vim by default)
```

Creating a Git repo

- To create a new **local Git repo** in your current directory:
  - `git init`
- This will create a .git directory in your current directory. • Then you can commit files in that directory into the repo.
  - `git add *filename*`
  - `git commit –m "*commit message*"`
- To **clone a remote repo** to your current directory:
  - `git clone *url localDirectoryName*`
- 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 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`	

Add and commit a file

- The first time we ask a file to be tracked, and every time before we commit a file, we must add it to the staging area:
  - `git add file1.txt file2.txt`
    - Takes a snapshot of these files, adds them to the staging area.
    - In older VCS, "add" means "start tracking this file." In Git, "add" means "add to staging area" so it will be part of the next commit.
- To move staged changes into the repo, we commit:
  - `git commit –m "Fixing bug #7"`
- To undo changes on a file before you have committed it:
  - `git reset HEAD --` *filename* (unstages the file)
  - `git checkout --` *filename* (undoes your changes)
  - All these commands are acting on your local version of repo.

Viewing / undoing changes

- 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 -5` (to show only the 5 most recent updates), etc.

Branching 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*

Merge conflicts

- 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
```

branch 1's version

branch 2's version

- Find all such sections and edit them to the proper state (whichever of the two versions is newer / better / more correct).

Interaction w/ remote repo

- **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:

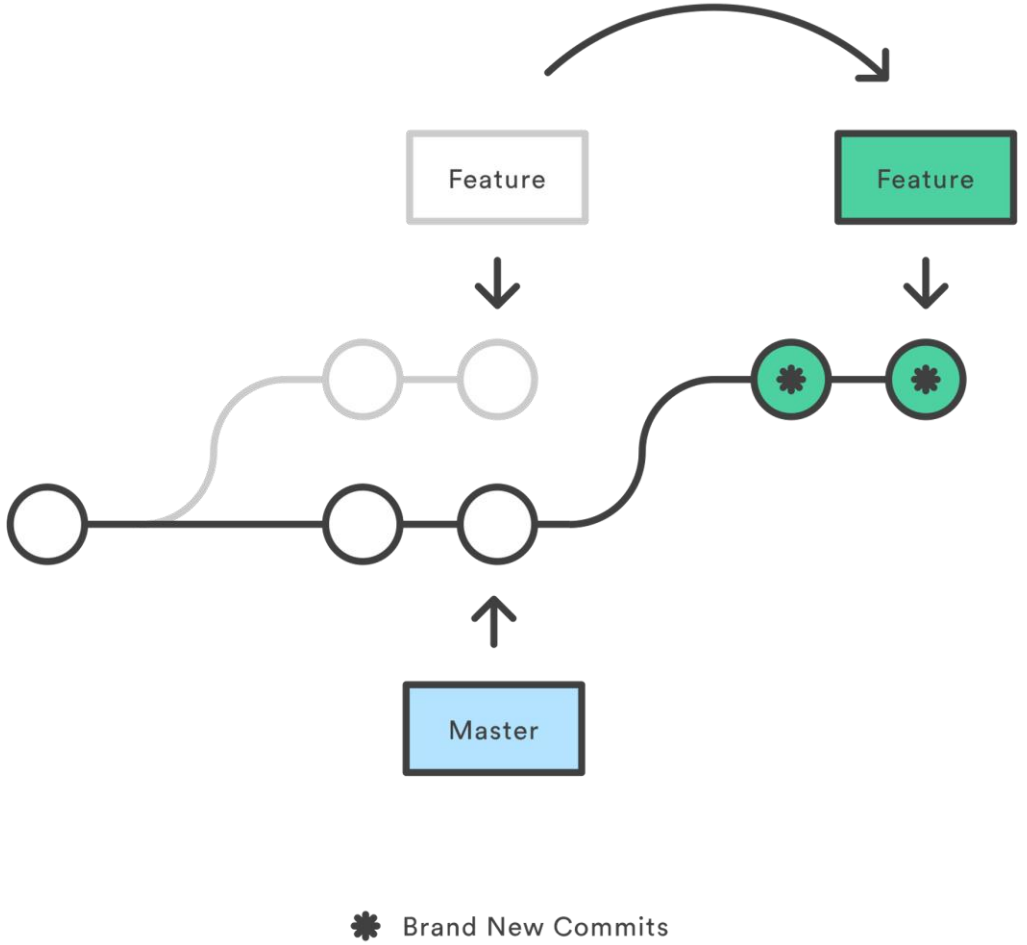
  - `git pull origin master`

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

  - `git push origin master`

Git rebase

Rebasing is the process of moving or combining a sequence of commits to a new base commit. Rebasing is most useful and easily visualized in the context of a feature branching workflow. The general process can be visualized as the following:



From a content perspective, rebasing is changing the base of your branch from one commit to another making it appear as if you'd created your branch from a different commit. Internally, Git accomplishes this by creating new commits and applying them to the specified base. It's very important to understand that even though the branch looks the same, it's composed of entirely new commits.

- `git rebase <base>`

https://git-scm.com/docs/git-rebase

Useful Info

https://githowto.com
https://git-scm.com/doc
https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud
https://mukulrathi.com/git-beginner-cheatsheet
https://chris.beams.io/posts/git-commit