Git

Kameswari Chebrolu

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

https://xkcd.com/1597/

Motivation

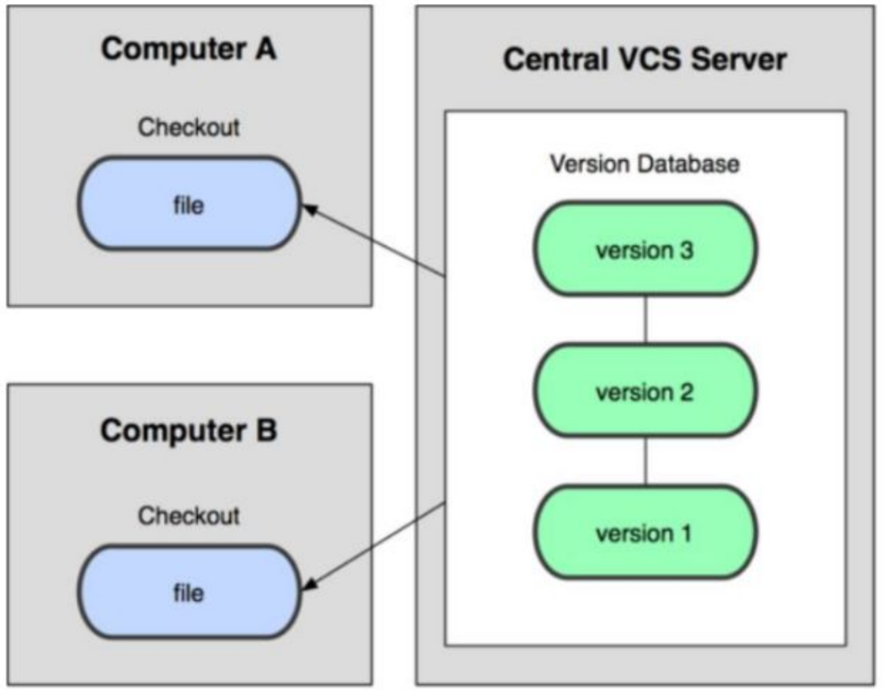
- You edit a file
- You change it some more
- And then some more…
- Darn!!! You messed up the file
- If only you know how the file changed!
  - Can revert to some older version and carry on from there

Version Management

- Version control: a system that records changes to (set of) files over time
  - Files can be code, scripts, documents, configuration files, data etc
- Roll-back functionality:
  - Mistakes happen! Can undo mistakes and go back to a working version
- Branching:
  - Can work on different issues/features in different branches (and discard branch if bad idea)
- Merging: Efficient collaboration
  - Different people can work on same code/project without interfering
- Traceability: who made the changes, and when and why the changes were made?

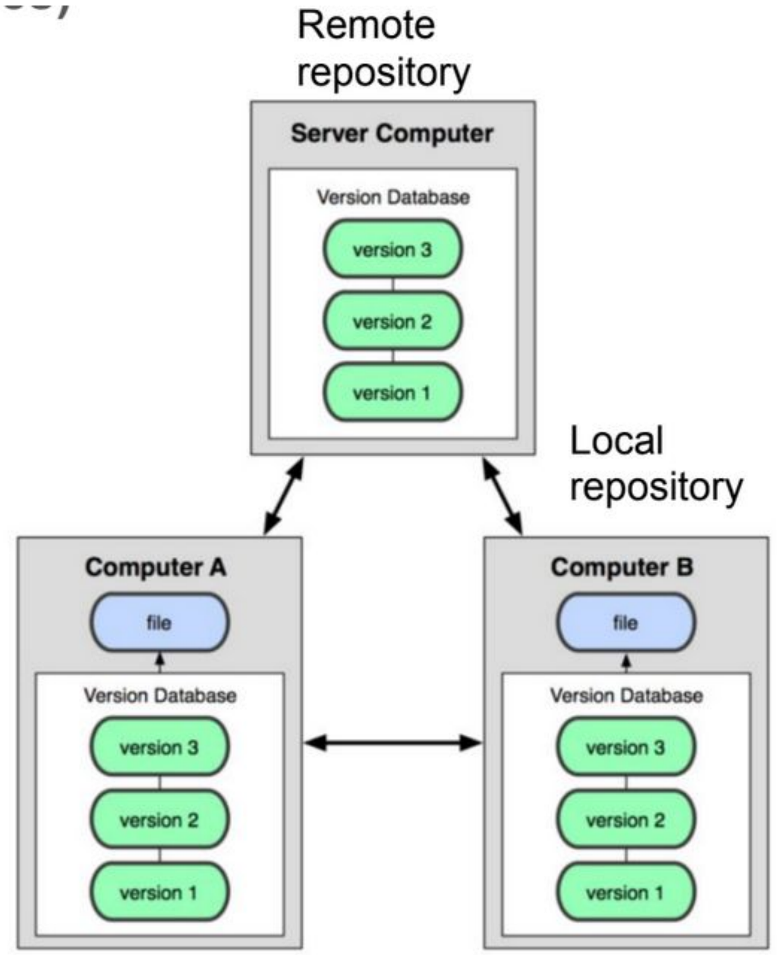
Centralized

- Example: cvs, svn
- Centralized server is vulnerable



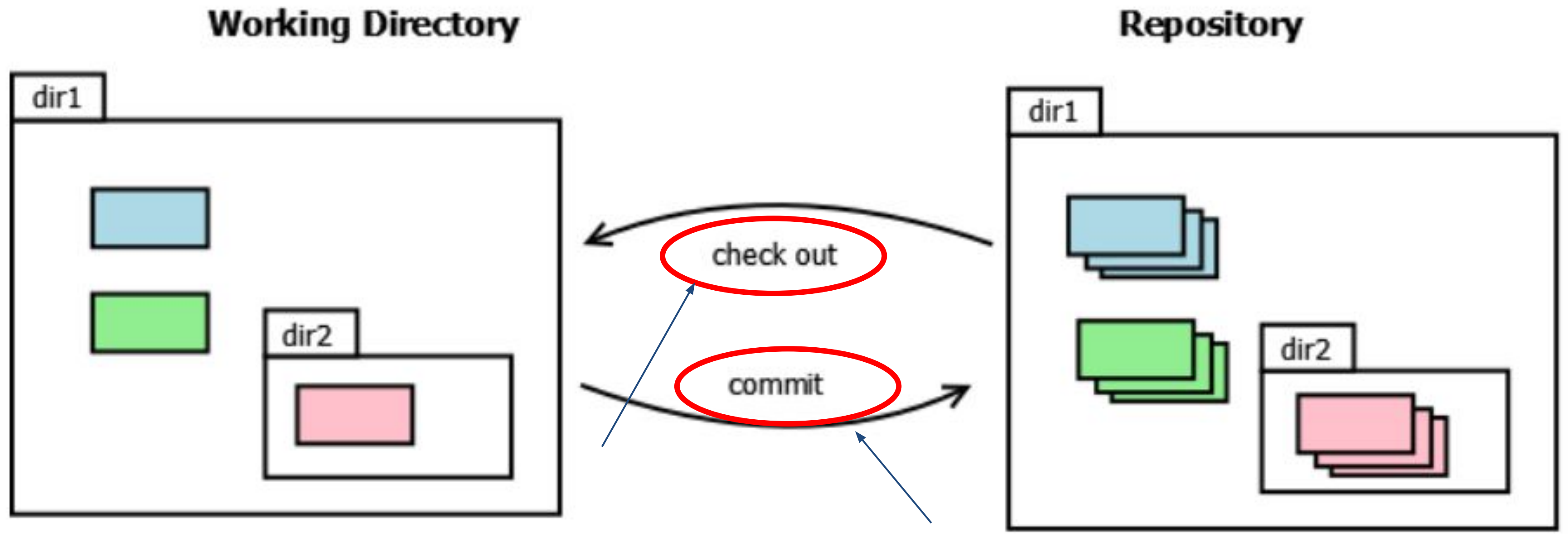
Distributed

- Example: git, Darcs
- Each client fully mirrors the repository.
  - If the server dies, any of the clients can help
  - User can interact with other users independent of central repo



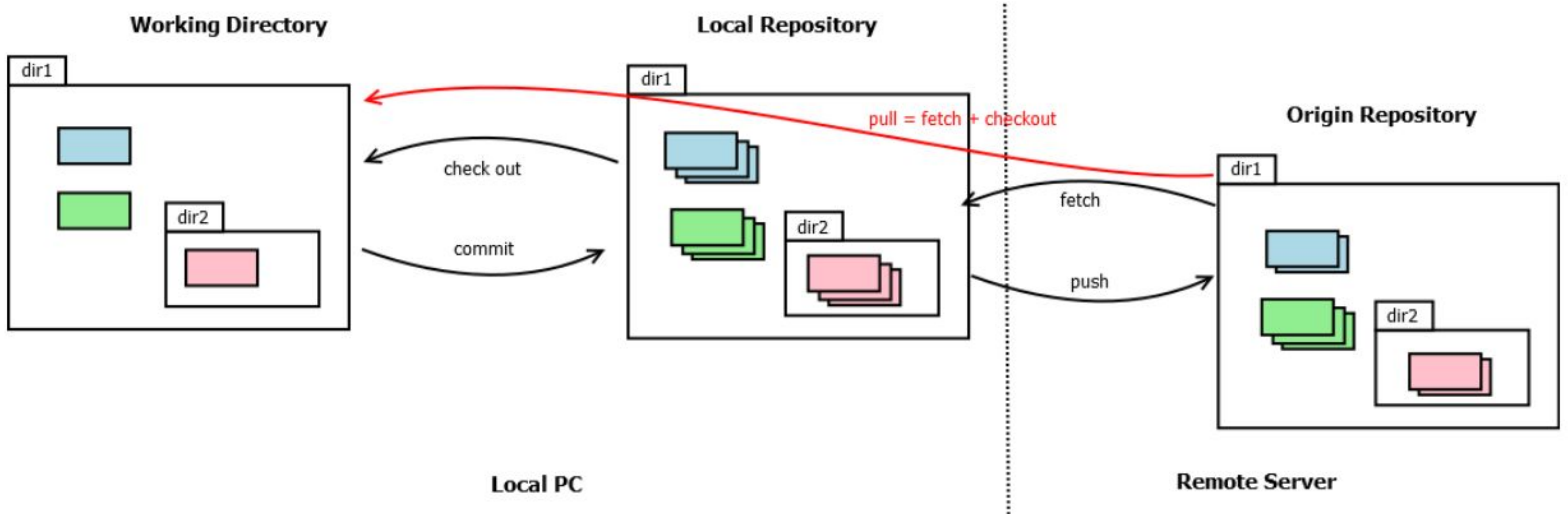
Repositories and Working Directory

- Repository: collection of versions of files
  - Tracks deleted and newly added files
  - Users do not, edit or even read files in the repo
- Working Directory: Current version of files
  - Users work on a copy of the files in their working directory



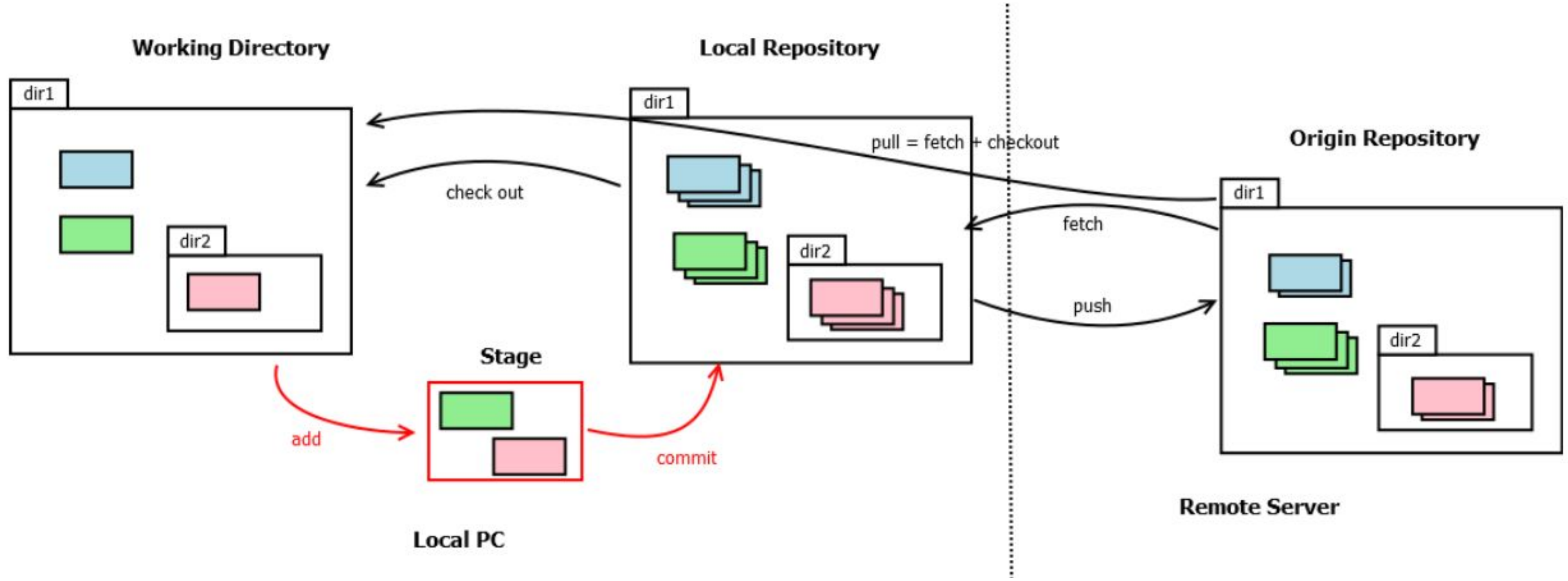
- Commit: send current contents of a file to the repository
    - current contents become a new version.
- Checkout: ask repository to give a copy of a version of a file

Git Architecture



- Local Repository: On local machine
- Origin repository: Remote for reliability
  - Many users will share origin
  - kept more or less in sync with local repository
- Push: push changes from our local repository to the origin
- Fetch: fetch changes anyone else may have made from the origin to our local repository
  - Fetching simply updates local repository
  - Need checkout for them to reflect in working directory.
- Pull: combines a fetch and a check out (most often used)
  - Changes reflect directly in working directory

Staging



- Commit/checkout/fetch/push/pull happen at directory level!
- What if we want to push some files, not all?
- Staging: We "add" files to stage and then commit from stage instead of the working directory

Origin

- Where is the origin repository?
- Any machine which supports SSH/HTTPS server will do
- Cloud Options: GitLab, GitHub, BitBucket, AWS Code Commit etc
  - A git hosting system with lots of additional features
  - E.g. project management, ticket management, bug tracking, access management etc

Creating a (local) git repository

- You can configure git via config
  - Username, email etc
  - E.g. git config --global user.name "kameswari chebrolu"
- “init” : Used to create a Git repository
  - git init
- After initialization, other files can be added

git status

- Tells current state of the repository
  - current working branch
  - what files are in staging area and not committed
  - what files are untracked etc
  - git status
- .gitignore file: helps specify files that git should ignore (even under untracked files)
  - E.g. temporary files (.o files)
  - These files won’t show in status

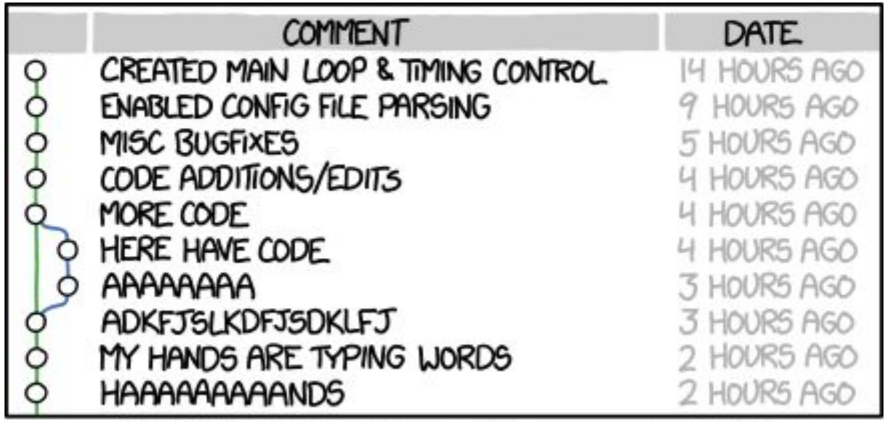
git add

- Add files to staging area
  - git add file1.txt file2.txt

git commit

- git commit: Commit the staged snapshot, launches a text editor for commit message
- git commit -a: Commit a snapshot of all changes in the working directory
  - But this only includes modifications to tracked files (those added with git add at some point in the past).
- git commit -m "commit message": shortcut to avoid editor
  - Use meaningful messages here, see xkcd comic :-)
  - Can also do git commit -am "commit message" (combines both)
- git commit --amend: modifies the last commit
  - Instead of creating a new commit, staged changes will be added to the previous commit

https://xkcd.com/1296/



	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Git log

git log
git log file1.txt (commit history of that file)

- A long hexadecimal number you see is the commit’s hash, helps identify a commit
  - can use just 5 digits mostly in commands
- “HEAD” is a pointer to the most recent commit

git show

git show :filename
Example: git show :file1.txt
Shows the content of file1.txt in the staging area

git show commit:filename
Example: git show HEAD:file1.txt
Shows the content of file1.txt in HEAD

Example: git show 5b80ea8:file1.txt
Shows the content of file1.txt in the commit object 5b80ea8

- `create file.txt`

Working area	Staging area	Commit
`file.txt - v1`		

- `git add file.txt`

Working area	Staging area	Commit
`file.txt - v1`	`file.txt - v1`	

- `git commit -m "msg"`

Working area	Staging area	Commit
`file.txt - v1`	`file.txt - v1`	`file.txt - v1`

- `edit file.txt`

Working area	Staging area	Commit
`file.txt - v2`	`file.txt - v1`	`file.txt - v1`

- add `file.txt`

Working area	Staging area	Commit
`file.txt - v2`	`file.txt - v2`	`file.txt - v1`

- `edit file.txt`

Working area	Staging area	Commit
`file.txt - v3`	`file.txt - v2`	`file.txt - v1`

- `git commit -m "msg"`
  `git commit file.txt -m "msg"`

Working area	Staging area	Commit
`file.txt - v3`	`file.txt - v2`	`file.txt - v2`
`file.txt - v3`	`file.txt - v3`	`file.txt - v3`

git diff

git diff <commit>: shows the diff between the current working tree and the <commit>

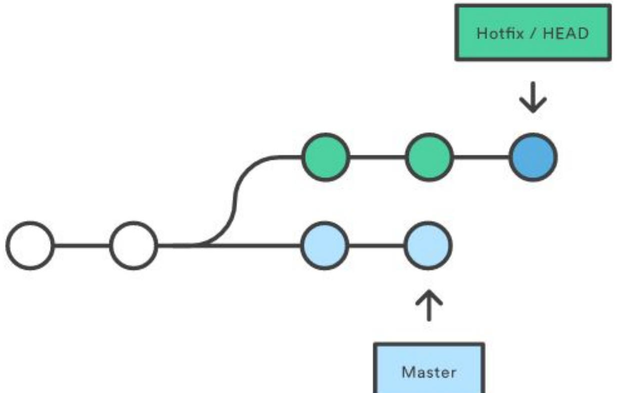
git diff --cached <commit>: shows the diff between your staged changes and the <commit>

Undoing Changes: checkout and commit

- You can move backwards in time by checking out an older commit.
  - git checkout commit-id
  - Will replace the contents of working directory by the contents of that older commit
  - Useful for “look but don’t touch” way to explore the older code
  - Get back to most recent commit via git checkout main
- Ability to rollback individual files to old versions: git checkout commit-id path-to-a-file
  - Then can use git commit if you want everything else to use current and this file to be some older version
- Want to entirely abandon one or more commits and start working again from an earlier version?
  - git reset --hard commit-id
  - More on “reset” later, but use reset with caution

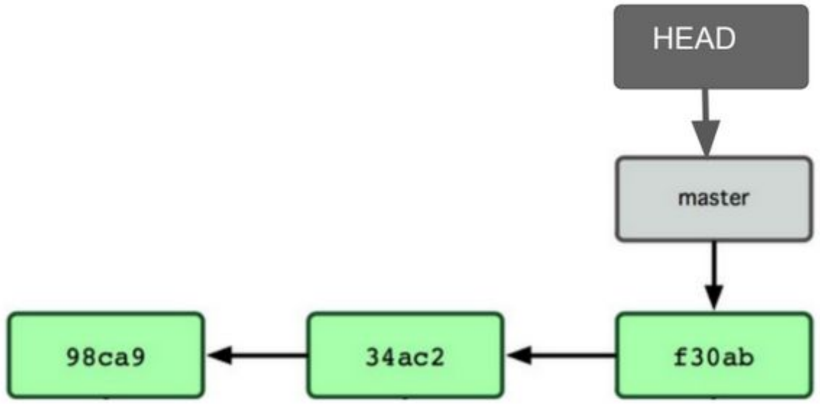
Branching

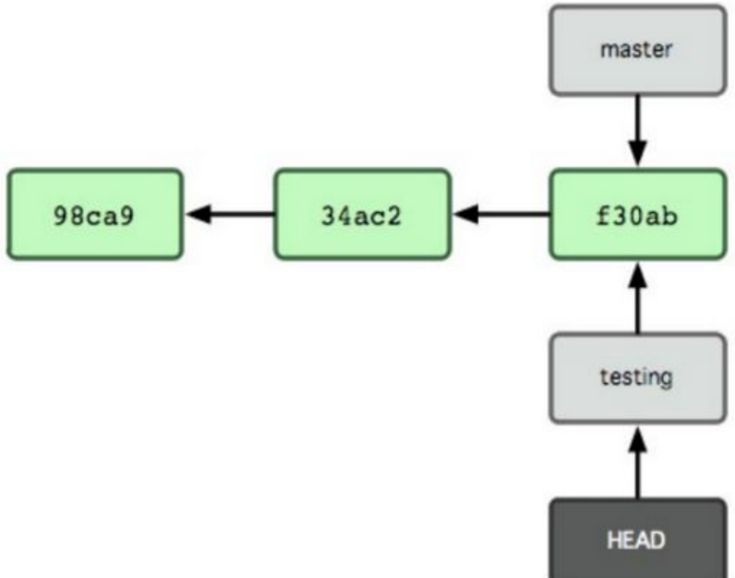
- Useful in solo projects, but critical in team projects
- So far, linear development; can move forward and backward
- What if you want to fix a bug (or try a feature), but don't want to mess up the master?



git branch/switch

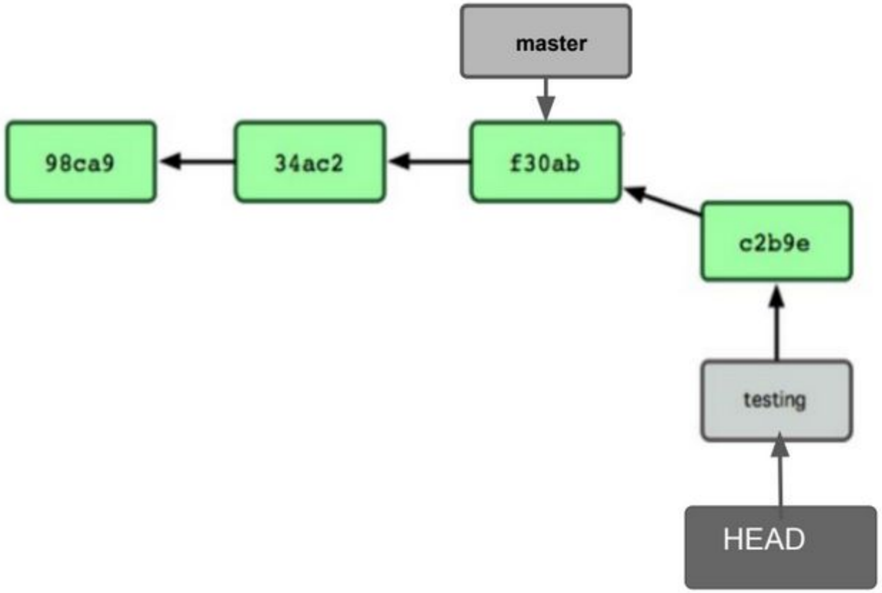
- git branch: List the branches
- git switch -c testing: create a new branch
  - "testing" is the name of this new branch

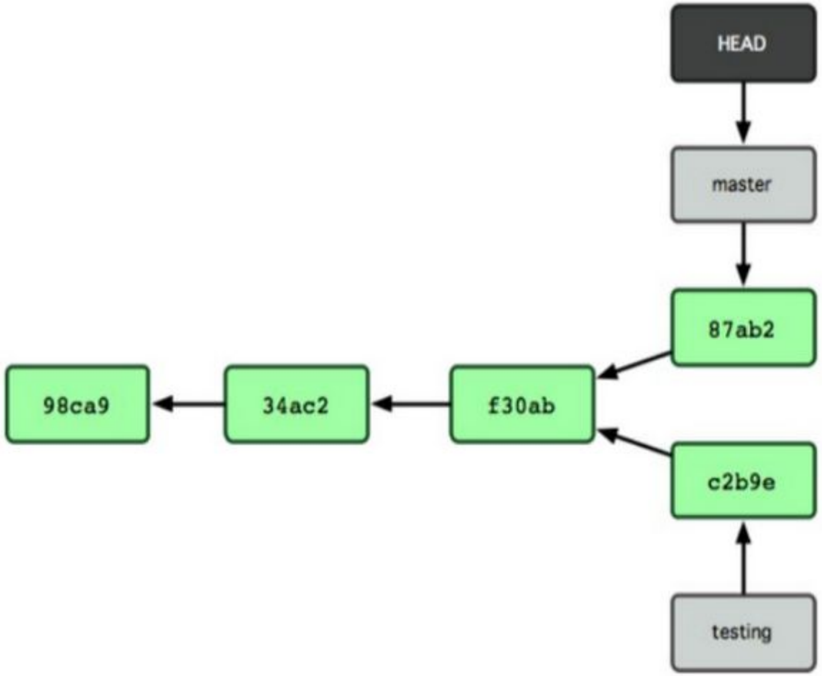




Development along testing

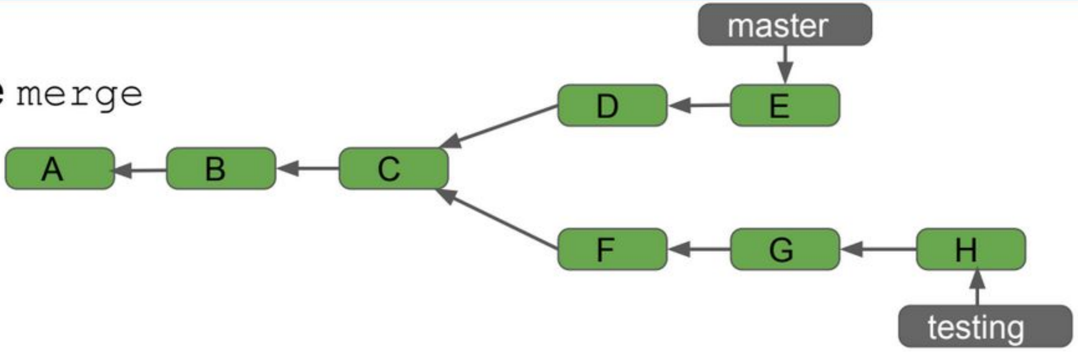
Separate development along master



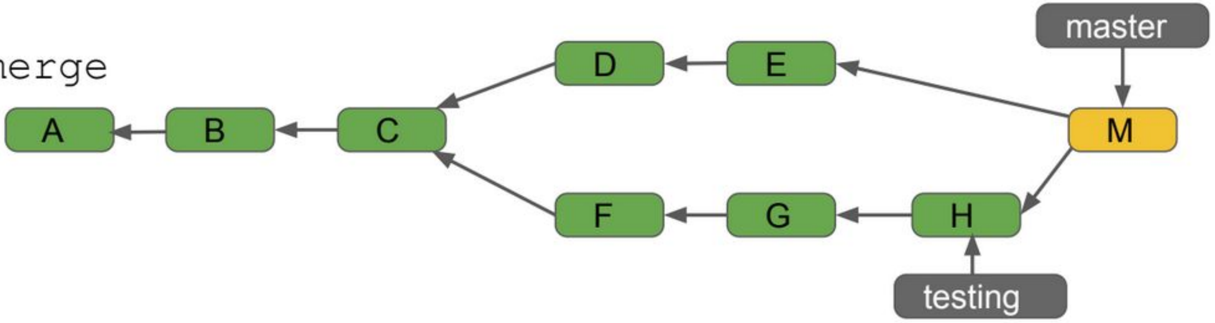


git merge

- Before `merge`



- After `merge`



- git checkout master (ensure you are in master branch; you want to merge testing into this)
- git merge -m "merging" testing (merge testing into master)
- Often this may result in a conflict, which you need to resolve.
  - After you resolve, you need to add and commit the files with conflict into master
- Note testing still exists and not affected by merge
  - git checkout testing

Reference

https://www.cs.odu.edu/~zeil/cs252/latest/Public/git/index.html

https://sillevl.gitbooks.io/git/content/advanced/reset-checkout-revert/ (advanced-reverting changes, not in syllabus)