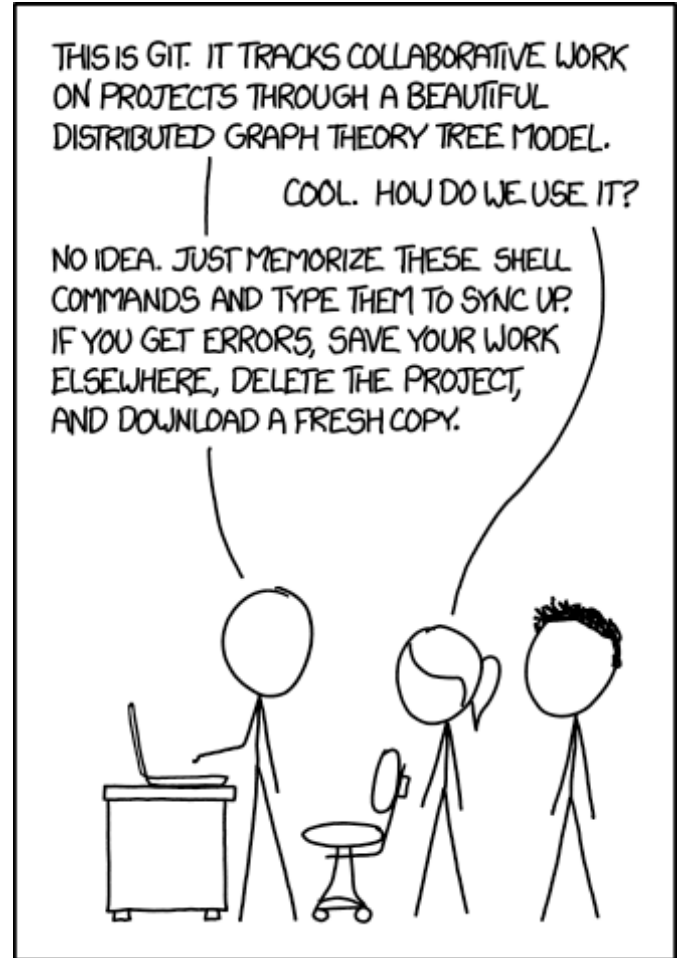


Git

Kameswari Chebrolu

<https://xkcd.com/1597/>



Outline

- Motivation
- Architecture and Terminology
- Popular Commands (Local Repository)
- Working with Remote Repository

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

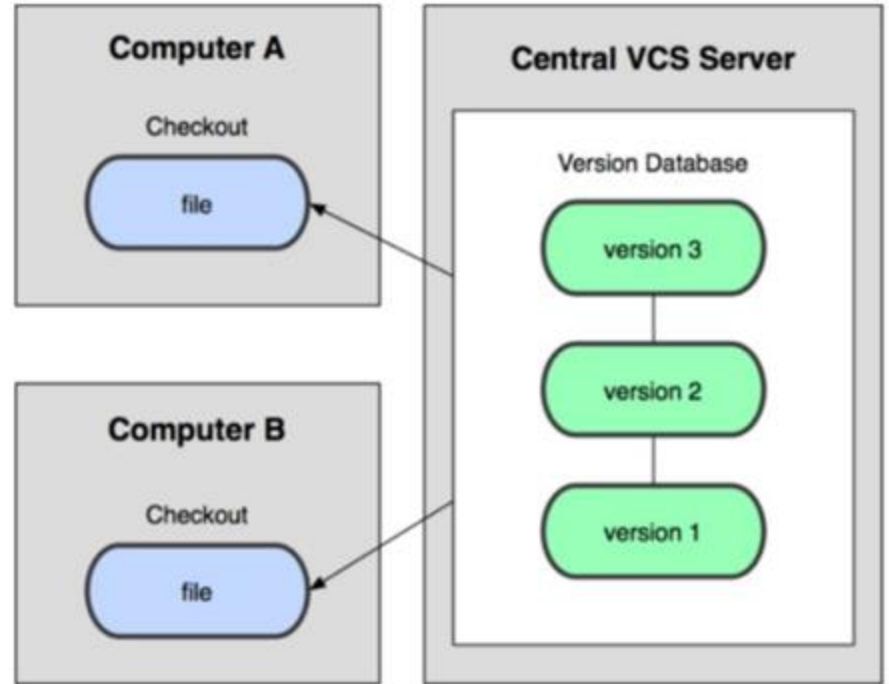
Version Management

- 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?

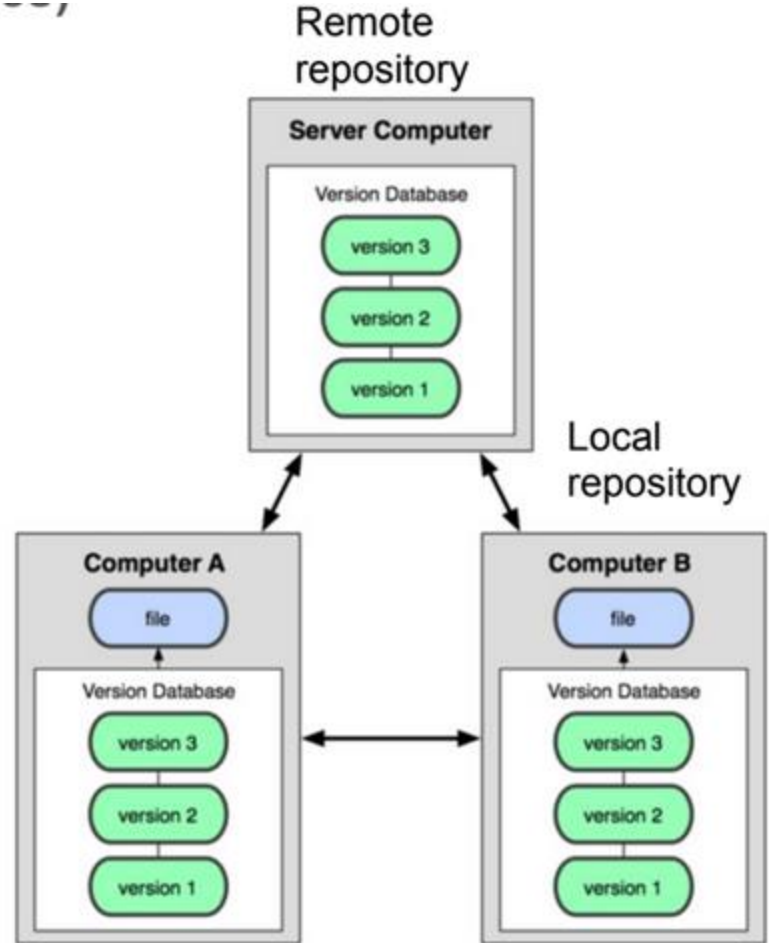
Centralized

- Example: cvs, svn
- Centralized server is vulnerable

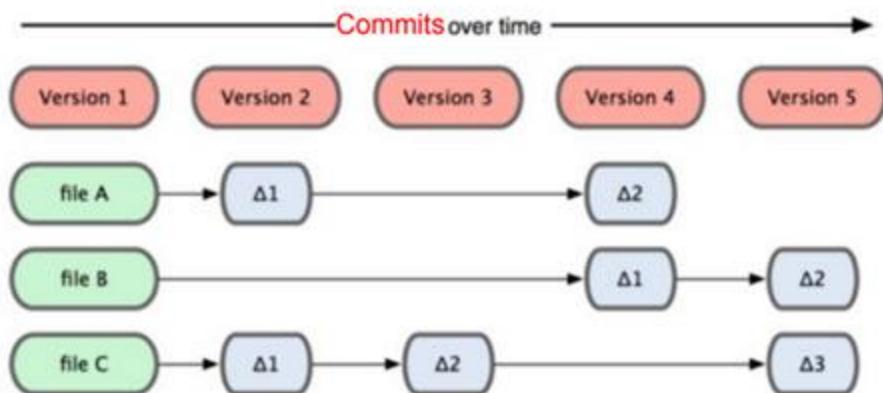


Distributed

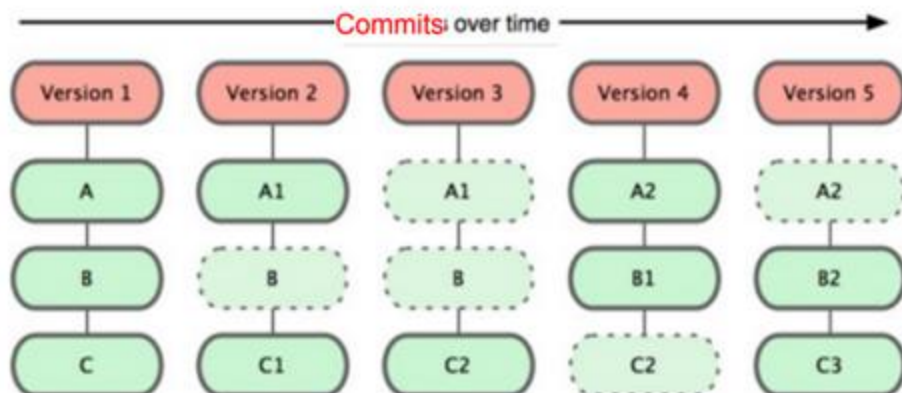
- Example: git, Darcs
- Each client fully mirrors the repository.
 - If the server dies, any of the clients can help
 - User can interact with other users independent of central repo



other vc systems

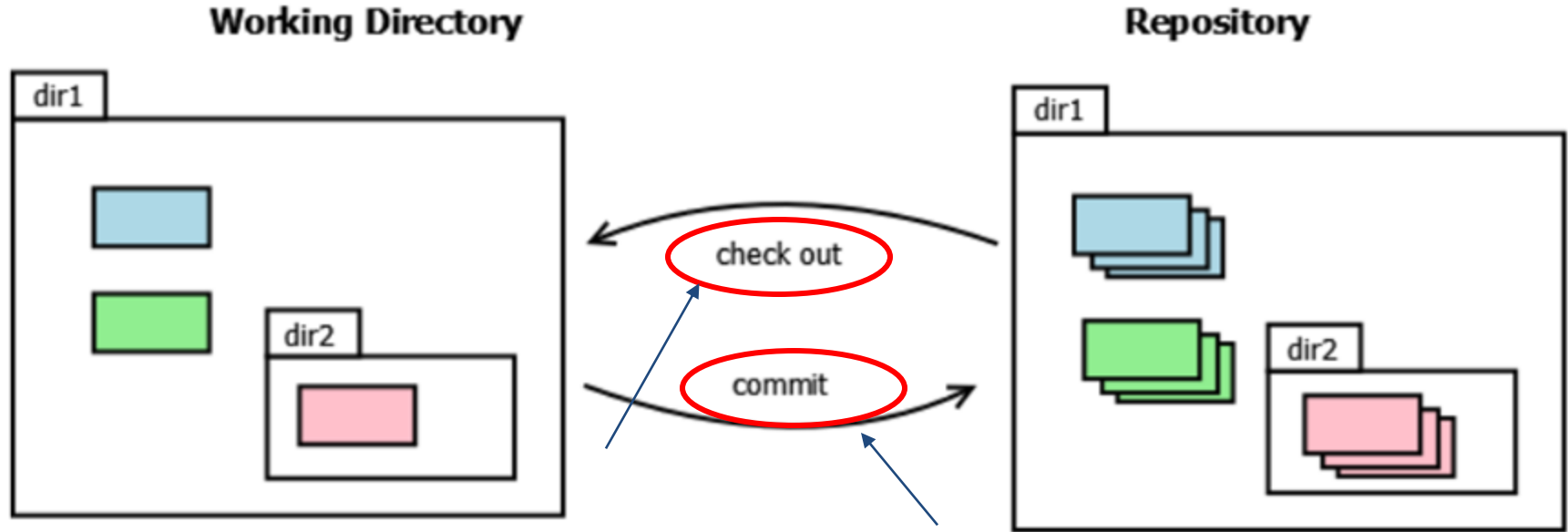


git



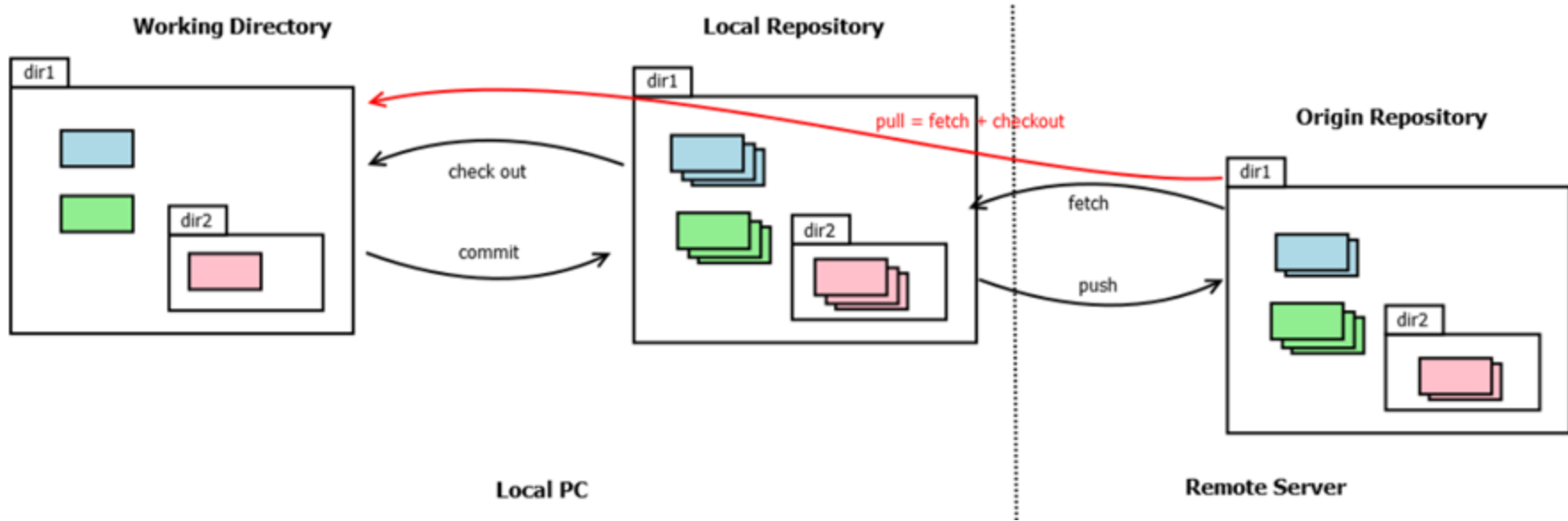
Repositories and Working Directory

- 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 file(s) to the repository
 - current contents become a new version.
- Checkout: ask repository to give a copy of file(s)

Git Architecture

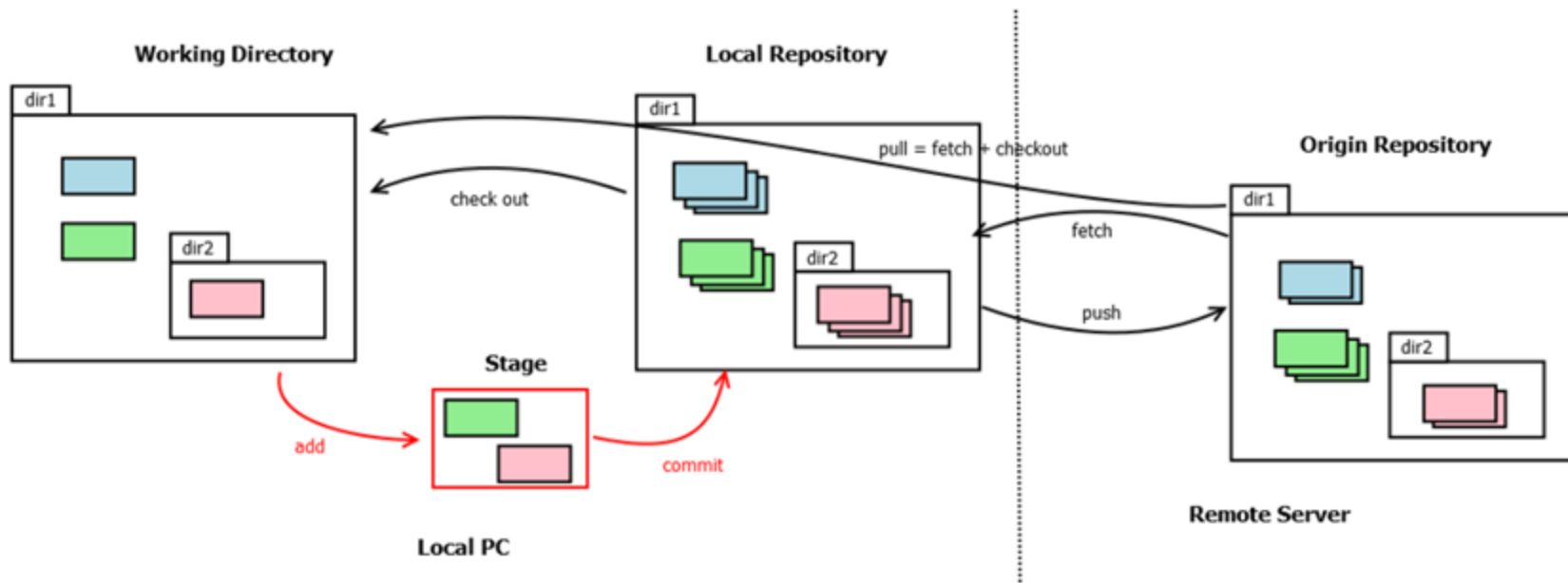


- Local Repository: On local machine
- Origin repository: Remote for reliability
 - Many users will share origin
 - Kept more or less in sync with local repository
- Push: push changes from local repository to the origin
- Fetch: fetch changes anyone else may have made from the origin to local repository
 - 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

Origin

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

Staging



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

Outline

- ~~• Motivation~~
- ~~• Architecture and Terminology~~
- Popular Commands (Local Repository)
- Working with Remote Repository

Outline for Commands

- **Creating a git repository (config, init)**
- Staging and committing changes (status, add, commit)
- Viewing history and changes (log, show, diff)
- Branching and Merging (checkout, branch, switch, merge)
- ~~Undoing and Resetting changes (reset, revert)~~
- ~~Synching with remote repository (remote, pull, push, fetch)~~
- ~~Deleting and Clean up (branch, clean)~~

Creating a (local) git repository

- Config command: you can configure git via “config”
- Set user details
 - `git config --global user.name "Your Name"`
 - `git config --global user.email "your@email.com"`
- Check configuration
 - `git config --list` # Shows all settings
 - `git config user.name` # Shows specific setting
- Set default editor
 - `git config --global core.editor "vim"`

- `git init` – Initializes a Git Repository
 - Converts a directory into a Git repo
 - Used when starting a new project or making an existing folder version-controlled
 - Creates a hidden `.git` folder that stores all version control data
- `.gitignore` file: helps specify files that git should ignore
 - Specifies files/folders to exclude from tracking.
 - E.g. temporary files (`.o` files)

Example .gitignore

Ignore compiled files

*.o

*.out

*.exe

Ignore backup and temporary files

*.swp

*.bak

*.tmp

Ignore build directories; root in explanation refers to the main folder where git init ran

#will ignore build folder anywhere in the root folder

#will ignore dist folder only in the root folder

build/

/dist/

Outline for Commands

- Creating a git repository (config, init)
- **Staging and committing changes (status, add, commit)**
- Viewing history and changes (log, show, diff)
- Branching and Merging (checkout, branch, switch, merge)
- Undoing and Resetting changes (reset, revert)
- Syncing with remote repository (remote, pull, push, fetch)
- Deleting and Clean up (branch, clean)

git status

- Tells current state of the repository and staging area
 - Current working branch
 - What files are in staging area and not committed
 - What files are untracked etc
- Note: files specified in .gitignore won't show in status

git add

- Add files to staging area
- Basic Usage:
 - `git add <file>` # Stage a specific file
 - Can also add multiple files: `git add file1.txt file2.txt`
 - `git add .` # Stages all modified and new files but does not include deleted files.
 - `git add -A` # Stage all changes, including deletions
 - `git add *.txt` # Stage all `.txt` files

git commit

- Saves staged changes to the local repository
 - Also launches a text editor for commit message
- `git commit`: Commit the **staged** snapshot
- `git commit -a`: Commit a snapshot of **all changes in the working directory**
 - 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
○	AAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAAANDS	2 HOURS AGO

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

- `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`

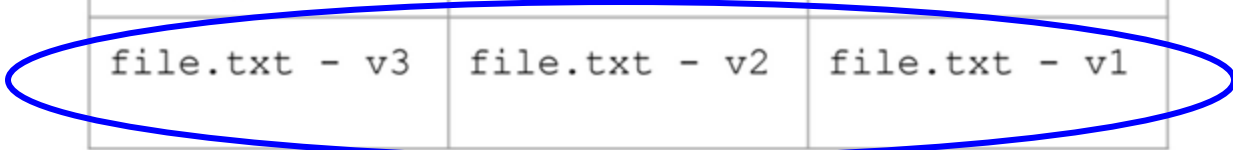
Working area	Staging area	Commit
file.txt - v2	file.txt - v1	file.txt - v1

- 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

Outline for Commands

- Creating a git repository (config, init)
- Staging and committing changes (status, add, commit)
- **Viewing history and changes (log, show, diff)**
- Branching and Merging (checkout, branch, switch, merge)
- ~~Undoing and Resetting changes (reset, revert)~~
- ~~Synching with remote repository (remote, pull, push, fetch)~~
- ~~Deleting and Clean up (branch, clean)~~

git log

- Displays the commit history of a repository
 - A long hexadecimal number is commit's hash
 - Helps identify a commit
 - can use just 5 digits mostly in commands
- Can also find commit history of a specific file
 - `git log file1.txt`
- Key options:
 - `git log --oneline` → Shows a compact view (commit hash + message)
 - `git log -p` → Shows commit diffs
 - `git log --graph --decorate --all` → Visualizes branches and merges
 - `git log --since="2025-01-01"` → Shows commits after a date

git diff

- git diff shows differences between Git states
- git diff <commit>: shows the diff between the current working repo and the <commit>
 - git diff HEAD → Compares working directory + staged changes against the latest commit
- git diff --cached <commit>: shows the diff between your staged changes and the <commit>
- git diff <commit1> <commit2> → Compares two commits

Linux command: diff

- diff stands for difference
- Compares the contents of two files and display the differences between them
 - highlight changes, additions, and deletions in a clear and readable format
- Tells us which lines in one file have to be changed to make the two files identical

Content of a.txt:

apple
banana
cherry
date
papaya
fig
grape

Content of b.txt

apple
blueberry
cherry
papaya
fig
grapefruit
melon
kiwi

diff a.txt b.txt

2c2

< banana

> blueberry

4d3

< date

7c6,8

< grape

> grapefruit

> melon

> kiwi

diff -u a.txt b.txt

--- a.txt 2025-02-08 16:17:06.304669604 +0530

+++ b.txt 2025-02-08 16:17:14.840549365 +0530

@@ -1,7 +1,8 @@

apple

-banana

+blueberry

cherry

-date

papaya

fig

-grape

+grapefruit

+melon

+kiwi

Explanation

- `diff a.txt b.txt`
 - Output:
 - Line numbers corresponding to the first file; A special symbol; Line numbers corresponding to the second file
 - E.g. `7c6,8`
 - line 7 in the first file needs to be changed to match line number 6-8 in the second file
 - Lines preceded by a `<` are lines from the first file.
 - Lines preceded by `>` are lines from the second file.
 - The three dashes (`—`) merely separate the lines of file 1 and file 2

- `diff -u a.txt b.txt` (unified mode)
 - Output:
 - The first file is indicated by `---`, and the second file is indicated by `+++`.
 - The first two lines provide information about file 1 and file 2, including the modification date and time
 - `@@ -1,7 +1,8 @@` denote the line range for both files
 - In this example, first file is 7 lines and second file is 8 lines
 - Subsequent lines represent the contents of the files with specific indicator
 - Unchanged lines are displayed without any prefix
 - Lines in the first file to be deleted are prefixed with -
 - Lines in the second file to be added are prefixed with +.

git show

- Displays detailed information about a commit or an object
- `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

- `git show <commit>` → Shows commit details (message, diff, author, date)
 - `git show HEAD` → Shows the latest commit
 - `git show <branch>` → Shows the latest commit on a branch

Outline for Commands

- Creating a git repository (config, init)
- Staging and committing changes (status, add, commit)
- Viewing history and changes (log, show, diff)
- **Branching and Merging (checkout, branch, switch, merge)**
- ~~Undoing and Resetting changes (reset, revert)~~
- ~~Synching with remote repository (remote, pull, push, fetch)~~
- ~~Deleting and Clean up (branch, clean)~~

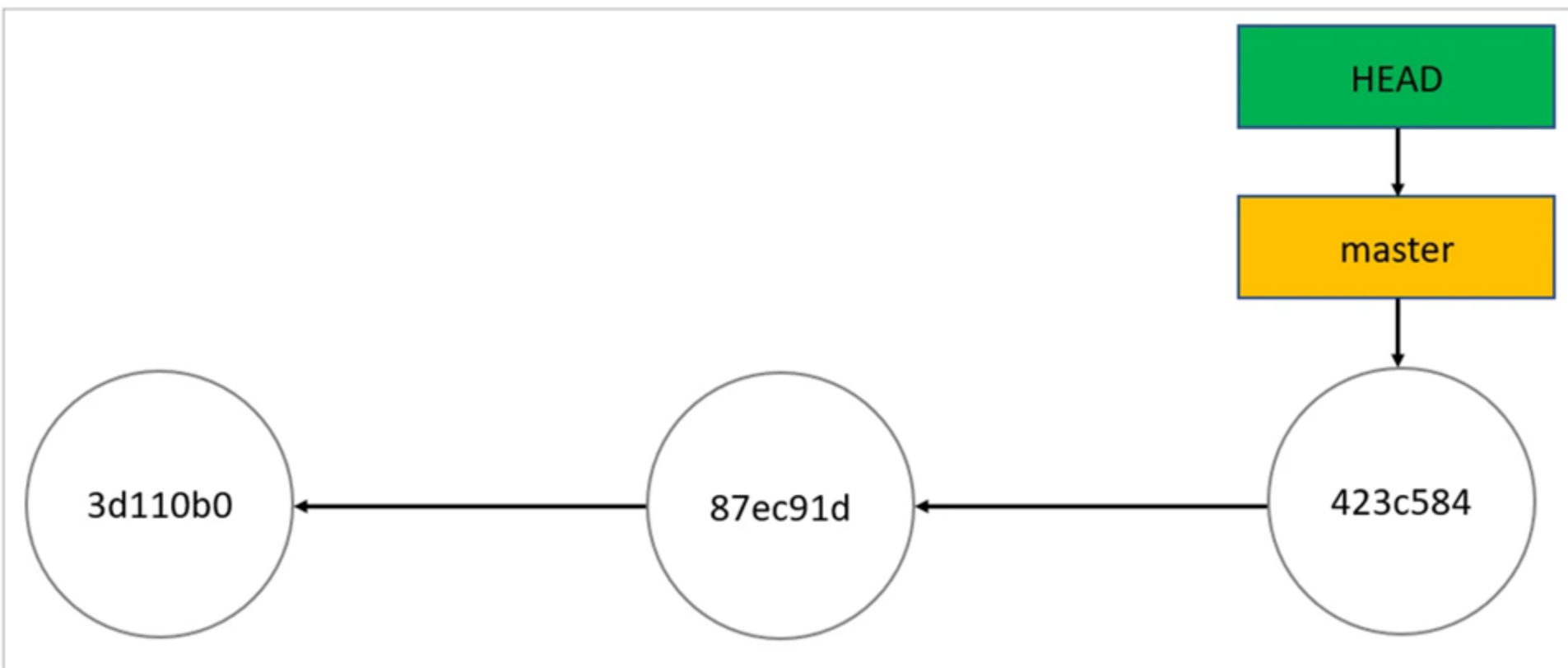
checkout

- You can move backwards in time by checking out an older commit
 - `git checkout commit-id`
 - Will replace contents of working directory by contents of older commit
 - Useful for “look but don’t touch” way to explore the older code
 - Then get back to most recent commit via `git checkout master`

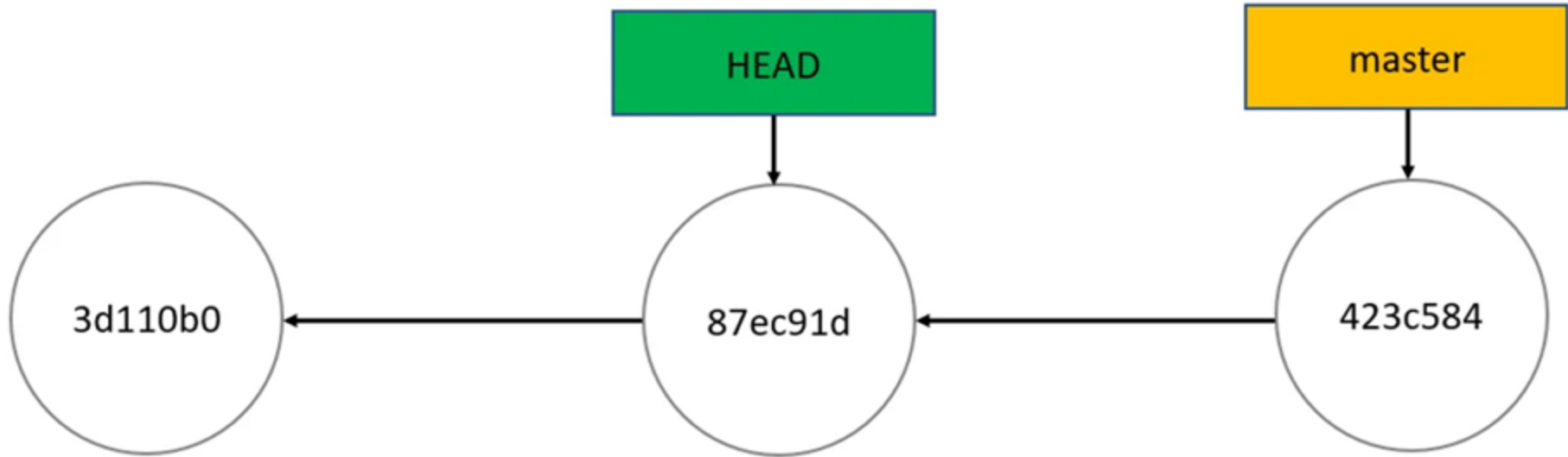
- Can also rollback individual files to old versions
 - `git checkout commit-id path-to-file`
 - Then use `git commit`
 - Everything else is what is in current repo and this file is some older version
- Note: `git restore` can also be used instead of `git checkout`
 - Recommended in newer Git versions!
 - Restore does not move the HEAD

HEAD

- HEAD answers the question: “Where am I right now?”
- Most of the time, HEAD points to a branch name
 - So far we have seen only one branch, master!
 - HEAD is synonymous with “last commit in the current branch.”
 - This is the normal state
- In a detached HEAD state; HEAD is pointing directly to a commit instead of a branch

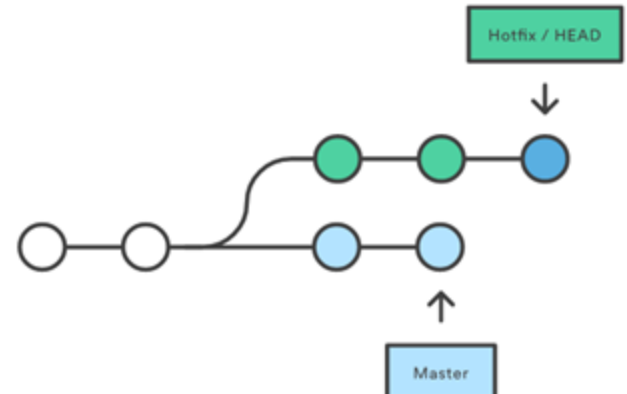


After running git checkout 87ec91d, the repo looks like this



Branching

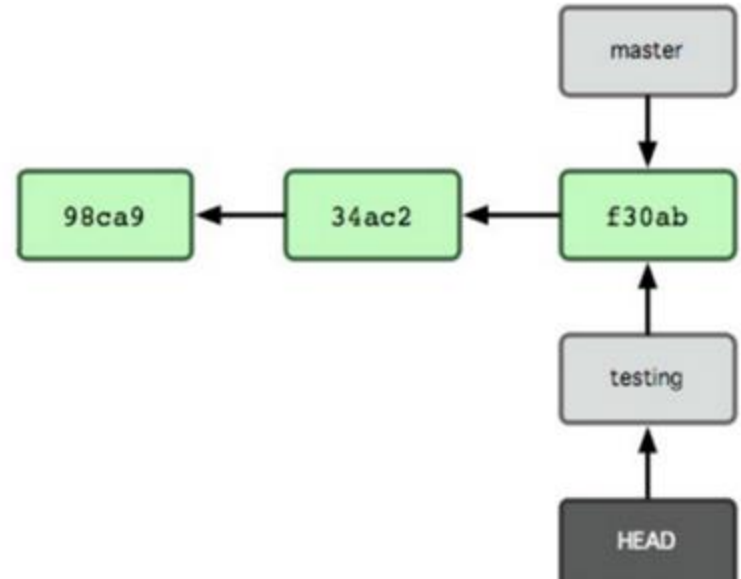
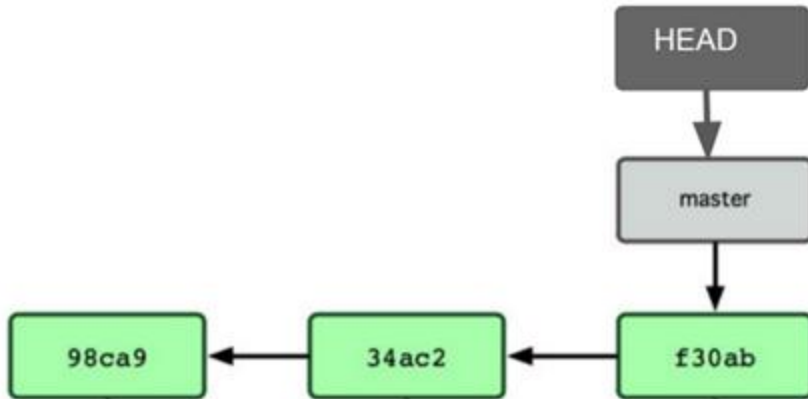
- Useful in solo projects, but critical in team projects
- So far, linear development; can move forward and backward
- What if you want to fix a bug (or try a feature), but don't want to mess up the master?



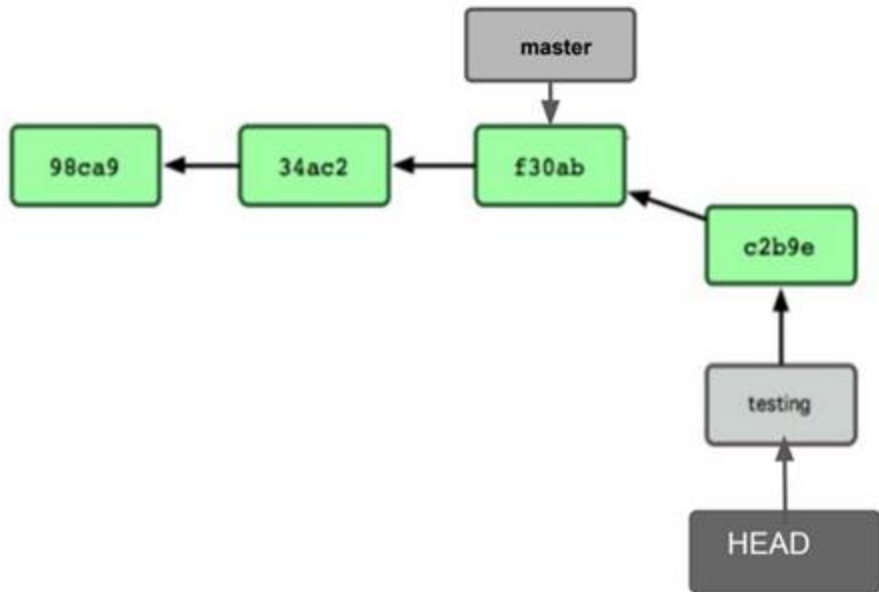
git branch/switch

- List branches: `git branch` (local branches)
- Create branch: `git branch new-branch`
- Switch branch: `git switch new-branch` (`git checkout new-branch`)
- Create & switch: `git switch -c new-branch` (`git checkout -b new-branch`)
- Delete branch: `git branch -d new-branch` (`-D` to force)
- Rename branch: `git branch -m old-name new-name`
- List remote branches: `git branch -r`
- List all branches: `git branch -a` (both local and remote)

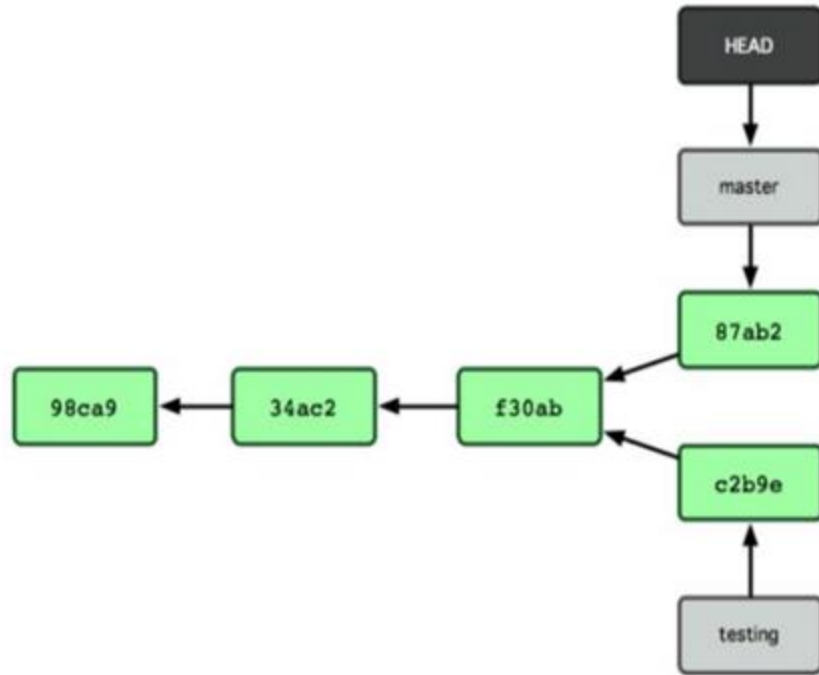
git branch/switch



Development along testing



Separate development along master

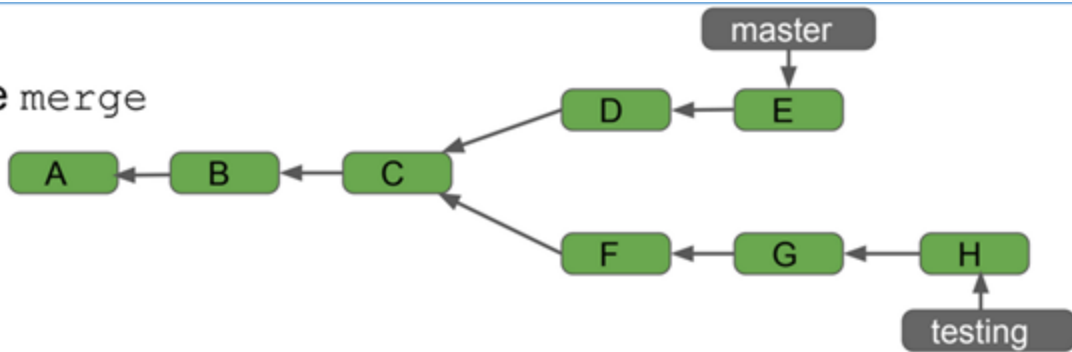


git merge

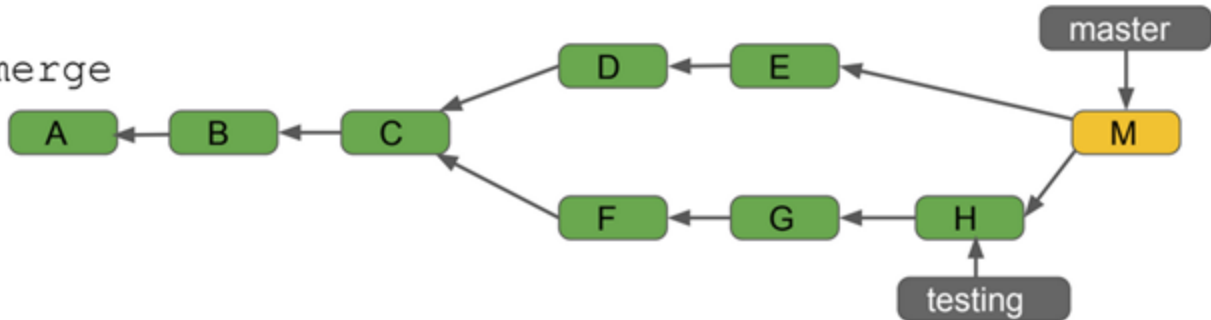
- git checkout master (you are in master branch; want to merge testing into this)
- git merge -m "merging" testing (merge testing into master)
- May result in a conflict, which needs to be resolved
 - After resolution, need to add and commit the files into master
 - Note testing still exists and not affected by merge
 - Try git checkout testing

git merge

- Before merge



- After merge



Outline for Commands

- Creating a git repository (config, init)
- Staging and committing changes (status, add, commit)
- Viewing history and changes (log, show, diff)
- Branching and Merging (checkout, branch, switch, merge)
- ~~• Undoing and Resetting changes (reset, revert)~~
- ~~• Synching with remote repository (remote, pull, push, fetch)~~
- ~~• Deleting and Clean up (branch, clean)~~

Working with remote repository

- Not part of syllabus but useful to know
- See git notes on how to set this up
 - Demo!

More Commands (not part of syllabus)

Checkout yourself

- `git revert`
- `git reset`
- `git rebase`
- `git restore`
- `git clean`

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)

<https://www.geeksforgeeks.org/diff-command-linux-examples/>