CS35L – Spring 2019

Slide set:	9.1
Slide topics:	Source control, Git
Assignment:	9

Software development process

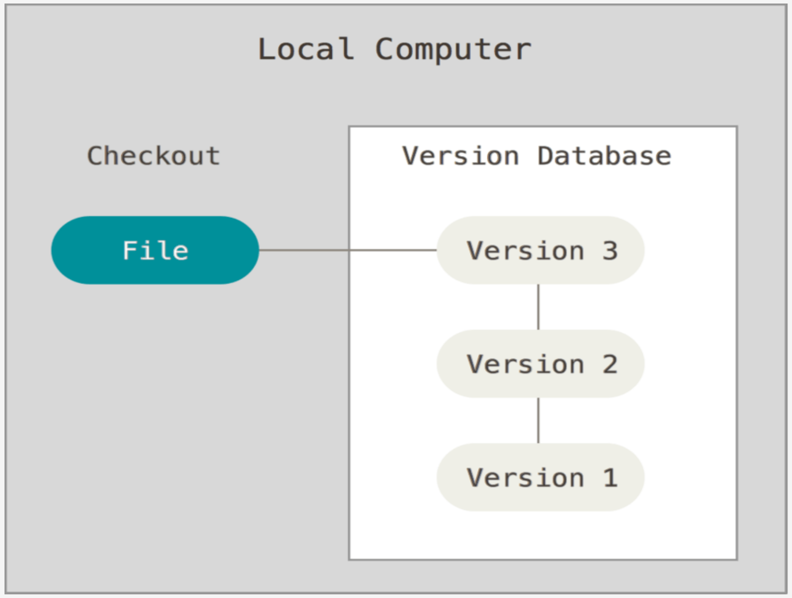
- Involves making a lot of changes to code
  - New features added
  - Bugs fixed
  - Performance enhancements
- Software team has many people working on the same/different parts of code
- Many versions of software released
  - Ubuntu 10, Ubuntu 12, etc
  - Need to be able to fix bugs for Ubuntu 10 for customers using it, even though you have shipped Ubuntu 12.

Source/Version Control

- Track changes to code and other files related to the software
  - What new files were added?
  - What changes made to files?
  - Which version had what changes?
  - Which user made the changes?
- Track entire history of the software
- Version control software
  - GIT, Subversion, Perforce

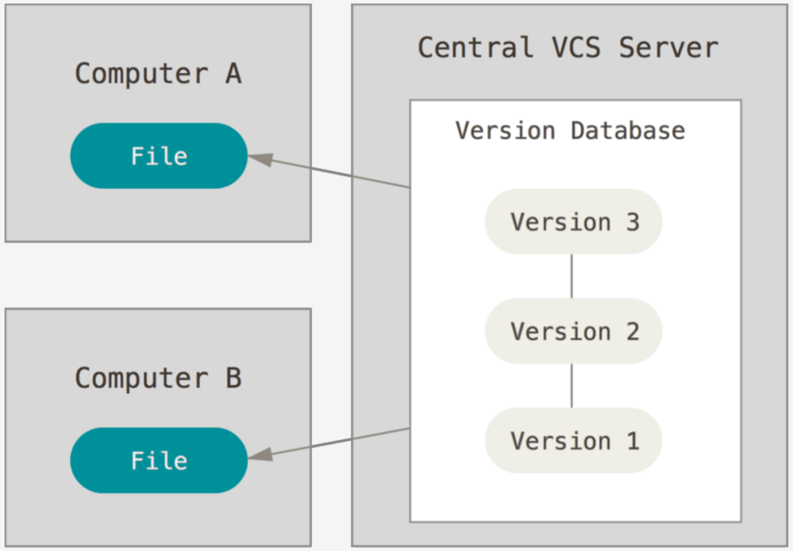
Git Features

- Speed
- Simple design
- Strong support for non-linear development (thousands of parallel branches)
- Fully distributed
- Able to handle large projects like the Linux kernel efficiently (speed and data size)



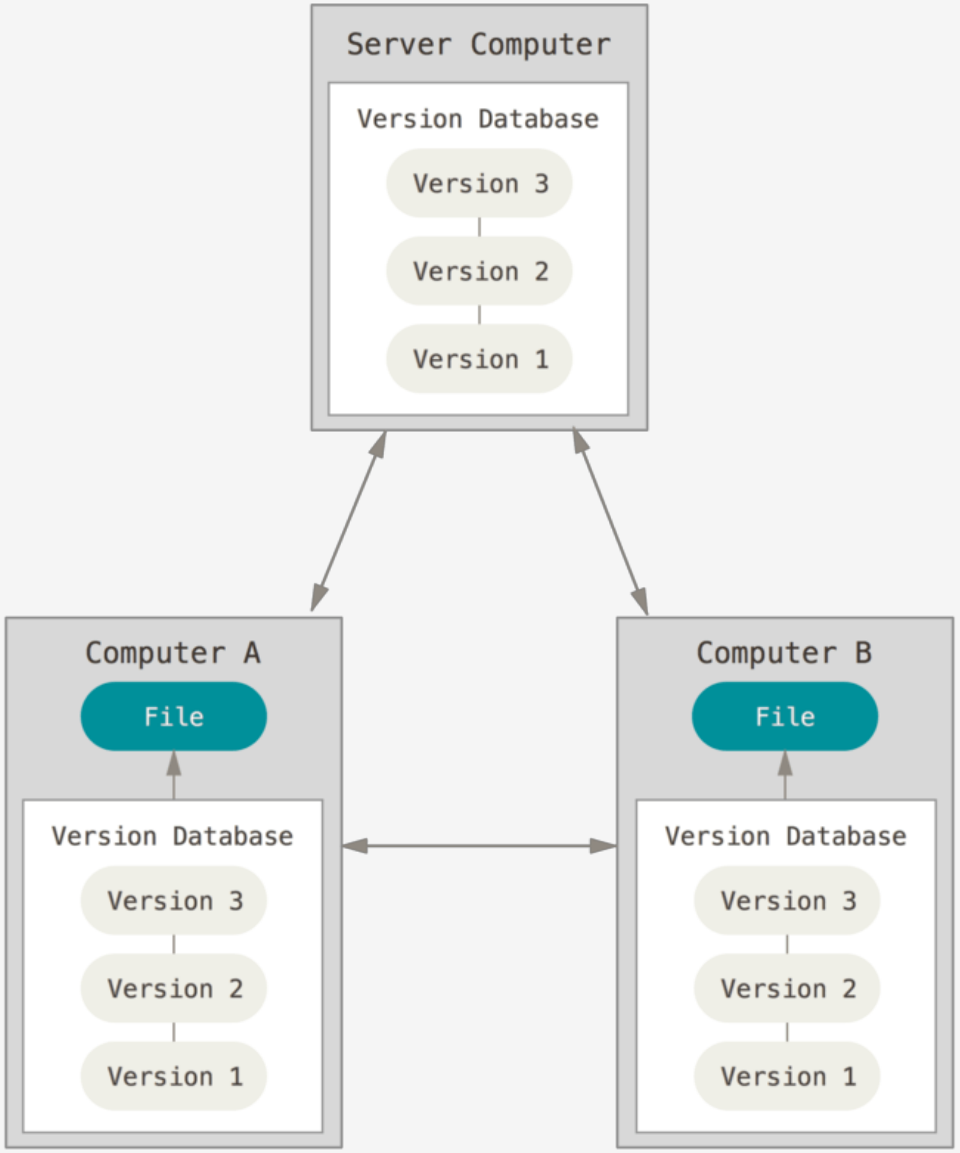
Local VCS

- Organize different versions as folders on the local system
- No server involved
- Other users should copy it via disk/network



Centralized VCS

- A single server that contains all the versioned files
- A number of clients that check out files from that central place



DISTRIBUTED VCS

- Version history is replicated at every user's machine
- Users have version control all the time
- Changes can be communicated between users
- Git is distributed

Git Integrity

Everything in Git is checksummed

Git knows about changes in the entire repo

The mechanism that Git uses for this checksumming is called a SHA-1 hash.

Git uses hash values to refer to every object

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

Git States

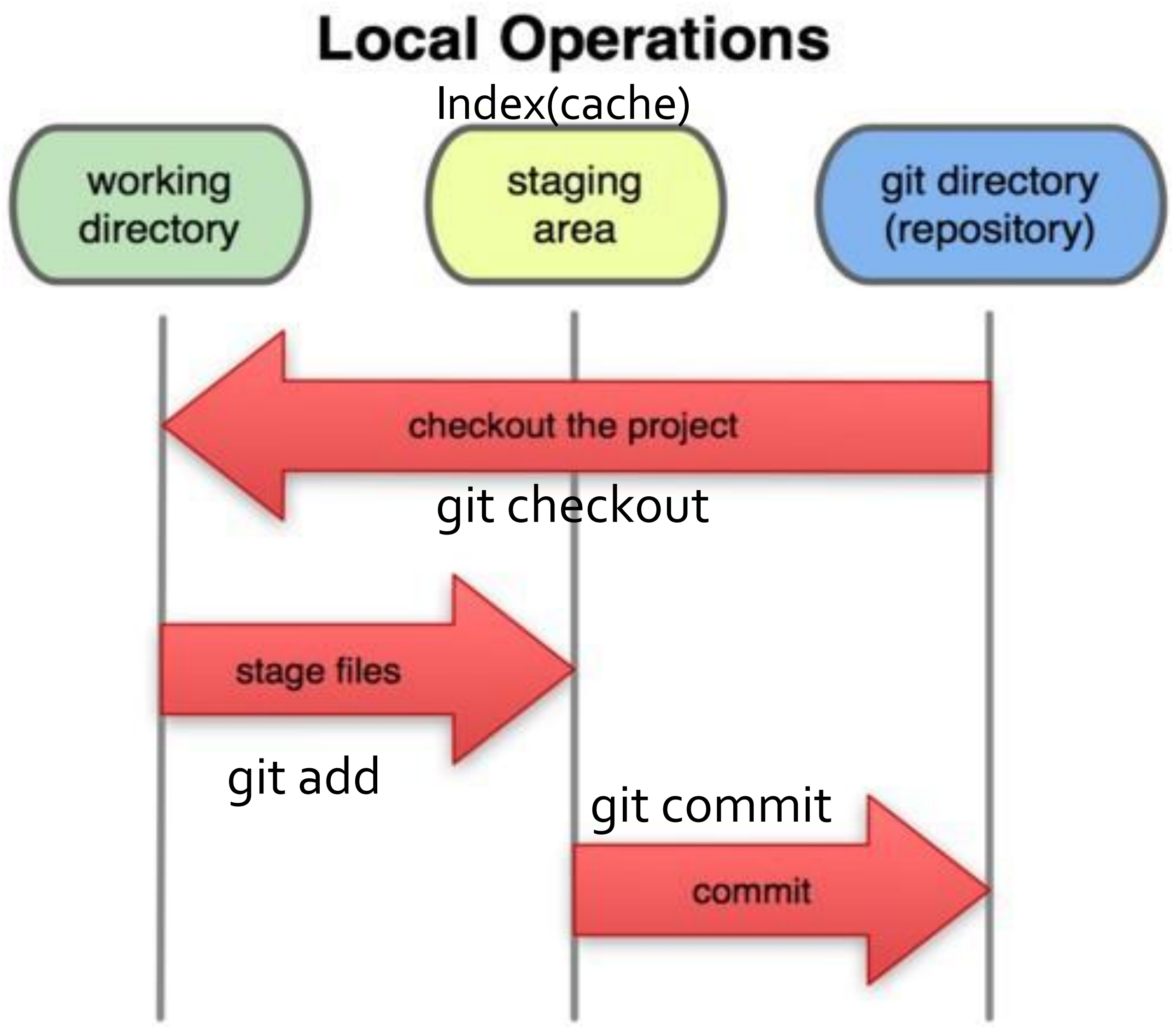


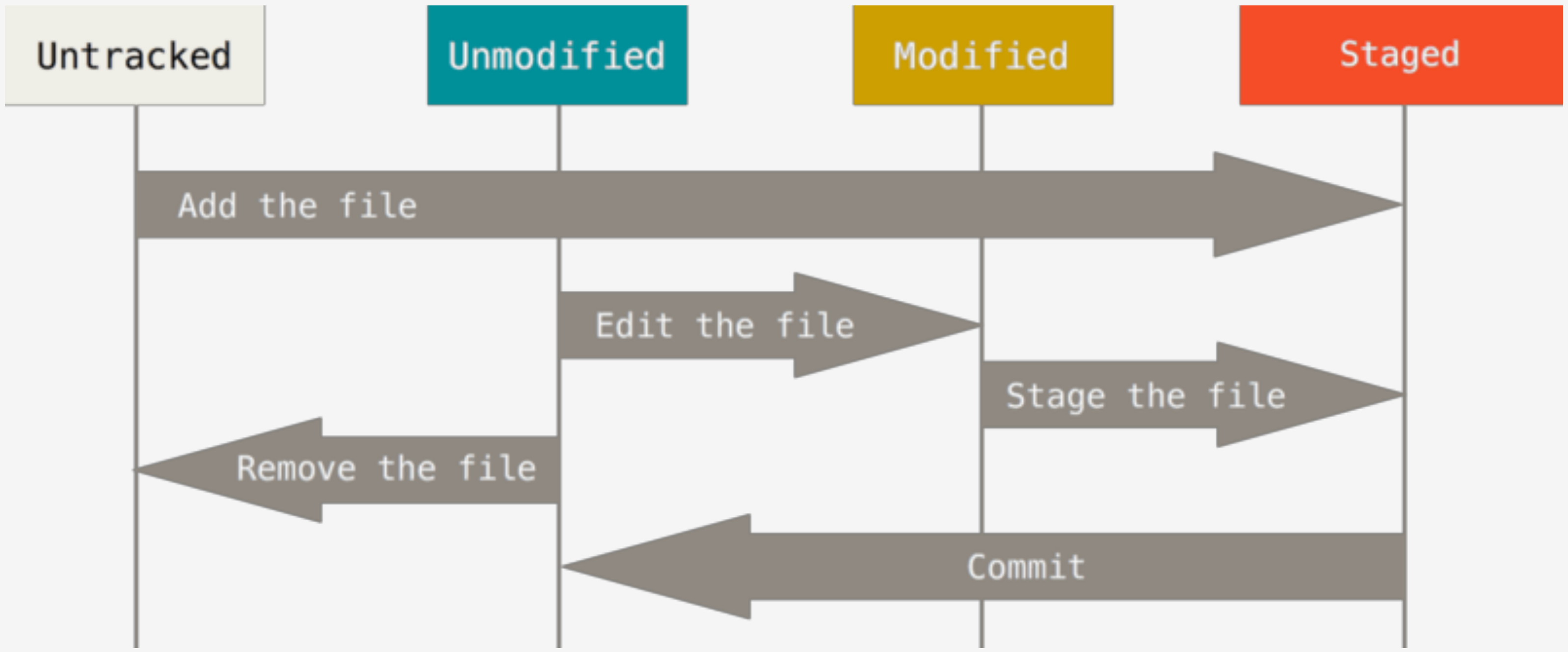
Image Source: git-scm.com

Terms used

HEAD	• Refers to the currently active head • Refers to a commit object
Branch	• Refers to a head and its entire set of ancestor commits
Master	• Default branch

First steps

- Configuration
  - `git config --list`
  - `git config --global user.name "John Doe"`
  - `git config --global user.email johndoe@example.com`
- Repository Creation
  - `git clone`
  - `git init`
- Adding/Staging files
  - You modify files in your working tree.
  - You selectively stage just those changes you want to be part of your next commit, which adds *only* those changes to the staging area.
  - `git add`
  - `git rm`


Untracked
Unmodified
Modified
Staged
Add the file
Edit the file
Stage the file
Remove the file
Commit

PROCESS

Staged files

- View current state
  - `git status (-s)`
- Ignore files (and patterns)
  - `.gitignore` file
  - Contains a list of the filenames/patterns of filenames to ignore while staging
- View changes
  - `git diff`

    Shows ‘diff’ of changes in the working directory (unstaged)
  - `git diff --staged`
    - Compares staged content with last commit
- Removing from staging area
  - `git reset HEAD <file>...`

Committing

```
git commit

#skip staging
git commit -a
```

- From working tree to (local) repository
- Anything you staged will be committed
  Anything you didn't stage is still sitting there modified; you can do another stage, then commit to add it to your history.
- Every time you perform a commit, you're recording a snapshot of your project that you can revert to or compare to later.
- You can skip the staging area if you don't want to "craft" a commit
- Removing files (put into staging area for deletion from repo)
  - `git rm`
  - "regular" deletion will just remove from the working tree, and you won't be able to "commit" the deletion

Viewing commits & history

- Viewing history
  - `git log`
  - `git log -p`

    Patch option that shows the difference in commits as successive
- Redoing a commit (e.g. with a new message)
  - `git commit --amend`

Git commands

- Repository creation
  - $ git init (Start a new repository)
  - $ git clone (Create a copy of an exisiting repository)
- Branching
  - $ git branch <new_branch_name>
    $ git checkout <tag/commit> -b <new_branch_name> (creates a new branch)
- Commits
  - $ git add (Stage modified/new files)
  - $ git commit (-m) (check-in the changes to the repository)
- Getting info
  - $ git status (Shows modified files, new files, etc)
  - $ git diff (compares working copy with staged files)
  - $ git log (-p) (Shows history of commits)
  - $ git show (Show a certain object in the repository)
- Getting help
  - $ git help

First Git Repository

- `$mkdir gittest`
- `$cd gittest`
- `$git init`
  - creates an empty git repo (.git directory with all necessary subdirectories)
- `$echo “Hello World” > hello.txt`
- `$git add .`
  - Adds content to the index
  - Must be run prior to a commit
- `$git commit –m “First check in”`

Working With Git

- `$ echo “I love Git” >> hello.txt`

- `$ git status`
  - Shows list of modified files
  - hello.txt

- `$ git diff`
  - Shows changes we made compared to index

- `$ git add hello.txt`

`$ git diff`

No changes shown as diff compares to the index

`$ git diff HEAD`

Now we can see changes in working version

`$ git commit –m “Second commit”`