

Source/Version Control

Presented by Shadman Kolahzary

Why version control?

- ▶ Scenario 1:
 - ▶ Your program is working
 - ▶ You change “just one thing”
 - ▶ Your program breaks
 - ▶ You change it back
 - ▶ Your program is still broken--*why?*
- ▶ Has this ever happened to you?

Why version control?

- ▶ Your program worked well enough yesterday
- ▶ You made a lot of improvements last night...
 - ▶ ...but you haven't gotten them to work yet
- ▶ You need to turn in your program *now*
- ▶ Has this ever happened to you?

Version control for teams

- ▶ Scenario:
 - ▶ You change one part of a program--it works
 - ▶ Your co-worker changes another part--it works
 - ▶ You put them together--it doesn't work
 - ▶ Some change in one part must have broken something in the other part
- ▶ What were all the changes?

Version control for teams

- ▶ Scenario:
 - ▶ You make a number of improvements to a class
 - ▶ Your co-worker makes a number of *different* improvements to the *same* class
- ▶ How can you merge these changes?

diff tools

- ▶ There are a number of tools that help you spot changes (differences) between two files
- ▶ Tools include `diff`, `rcsdiff`, `jDiff`, `WinMerge`, etc.
- ▶ Of course, they won't help unless you kept a copy of the older version
- ▶ Differencing tools are useful for finding a *small* number of differences in a *few* files

Version control systems

- ▶ A version control system (often called a source code control system) does these things:
 - ▶ Keeps multiple (older and newer) versions of everything (not just source code)
 - ▶ Requests comments regarding every change
 - ▶ Displays differences between versions

Advantages of version control

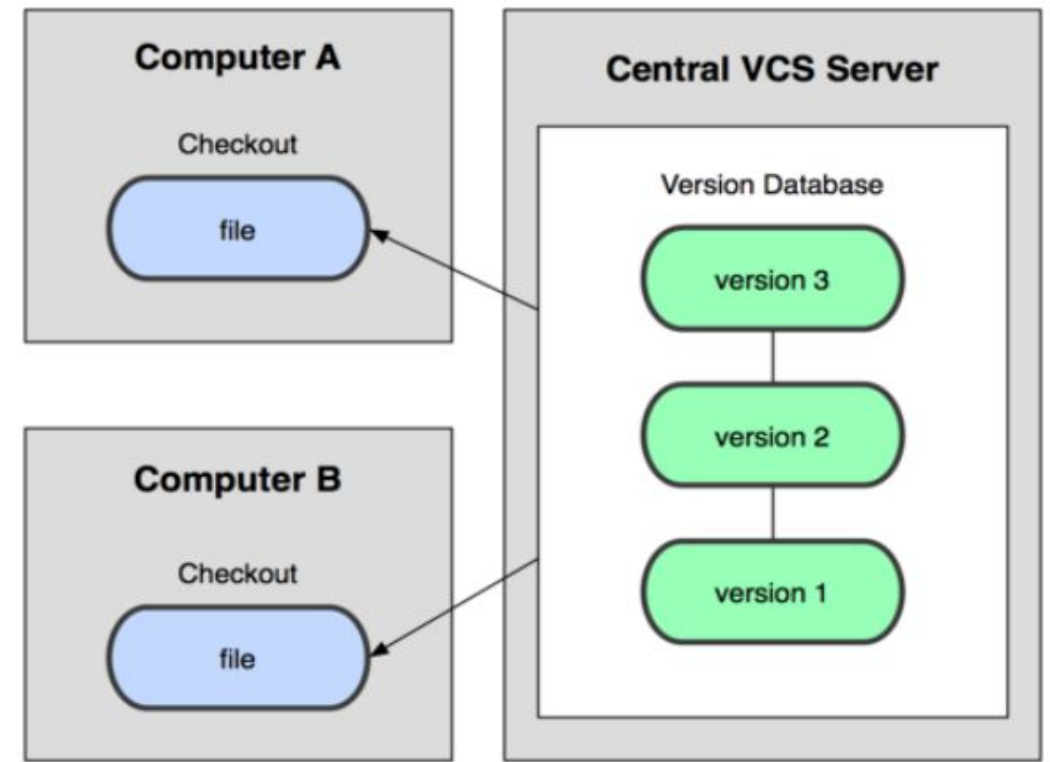
- ▶ For working by yourself:
 - ▶ Gives you a “time machine” for going back to earlier versions
 - ▶ Gives you great support for different versions (standalone, web app, etc.) of the same basic project
- ▶ For working with others:
 - ▶ Greatly simplifies concurrent work, merging changes
- ▶ For getting an internship or job:
 - ▶ Any company with a clue uses some kind of version control
 - ▶ Companies without a clue are bad places to work

Version control systems

- ▶ Some well-known version control systems are CVS, Subversion, Mercurial, and Git
 - ▶ CVS and Subversion use a “central” repository; users “check out” files, work on them, and “check them in”
 - ▶ Mercurial and Git treat all repositories as equal
- ▶ Distributed systems like Mercurial and Git are newer and are gradually replacing centralized systems like CVS and Subversion

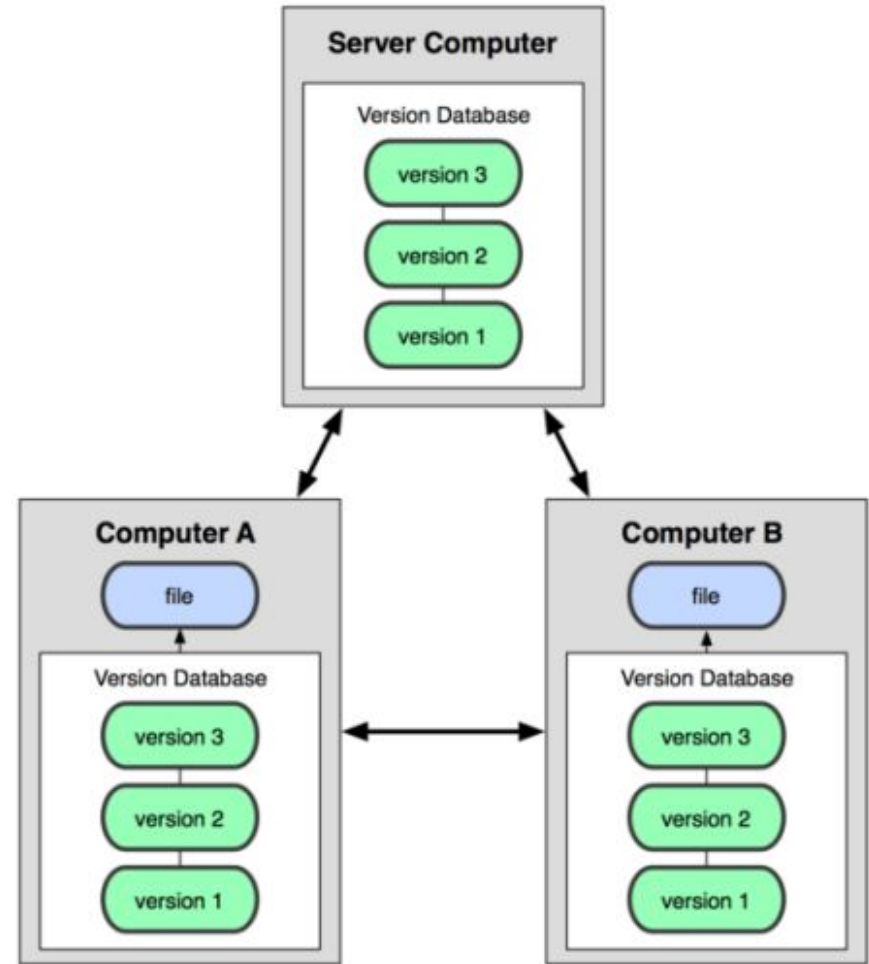
Centralized VCS

- 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 checkin 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



About Git

- Git's design was inspired by [BitKeeper](#) and [Monotone](#).
- 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



Notable users of git

- ▶ GNU Core Utilities
- ▶ DokuWiki
- ▶ Drupal
- ▶ LibreOffice
- ▶ MediaWiki
- ▶ Mono
- ▶ ASP.NET MVC
- ▶ ADO.NET Entity Framework
- ▶ NuGet
- ▶ Linux kernel
- ▶ Android
- ▶ Bugzilla
- ▶ GNOME
- ▶ GNU Emacs
- ▶ GRUB2
- ▶ KDE
- ▶ MySQL
- ▶ Perl 5
- ▶ PostgreSQL
- ▶ X.Org
- ▶ Cairo
- ▶ Qt Development Frameworks
- ▶ Samba
- ▶ OpenEmbedded
- ▶ Ruby
- ▶ Ruby on Rails
- ▶ Wine
- ▶ Fluxbox
- ▶ Openbox
- ▶ Compiz Fusion
- ▶ XCB
- ▶ Elinks
- ▶ XMMS2
- ▶ E2fsprogs

Companies & Projects Using Git

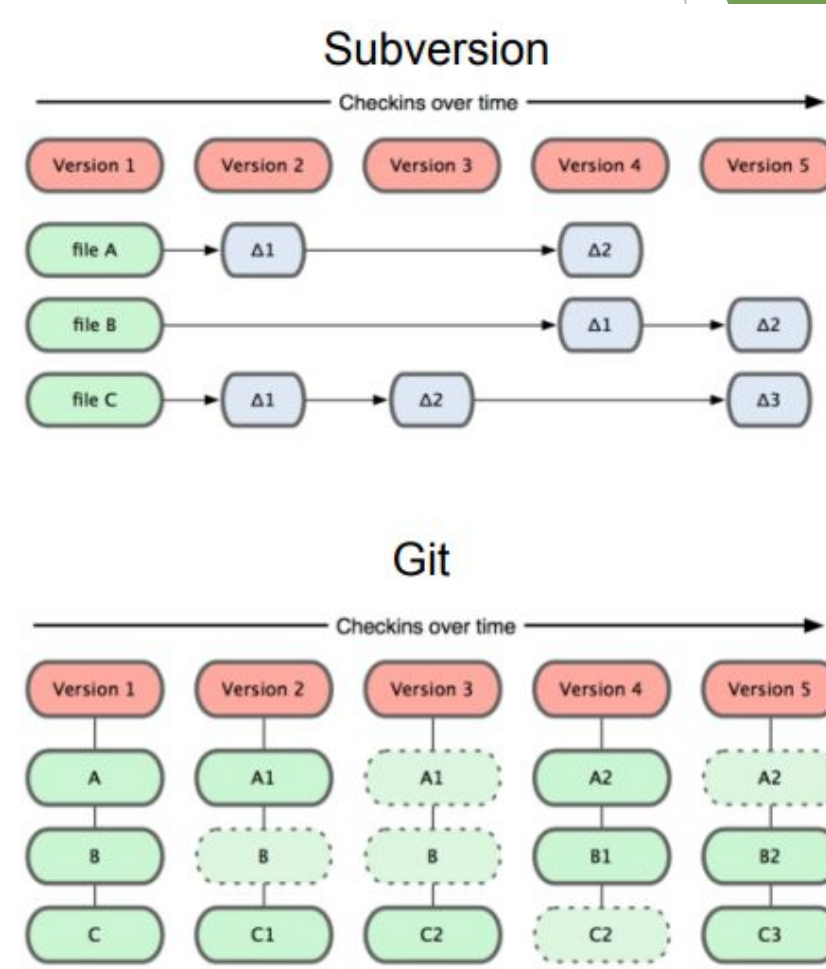


Installing/learning Git

- Git website: <http://git-scm.com/>
 - Free on-line book: <http://git-scm.com/book>
 - Reference page for Git: <http://gitref.org/index.html>
 - Git tutorial: <http://schacon.github.com/git/gittutorial.html>
 - Git for Computer Scientists:
<http://eagain.net/articles/git-for-computer-scientists/>
- At command line: (where verb = config, add, commit, etc.)
 - `git help verb`

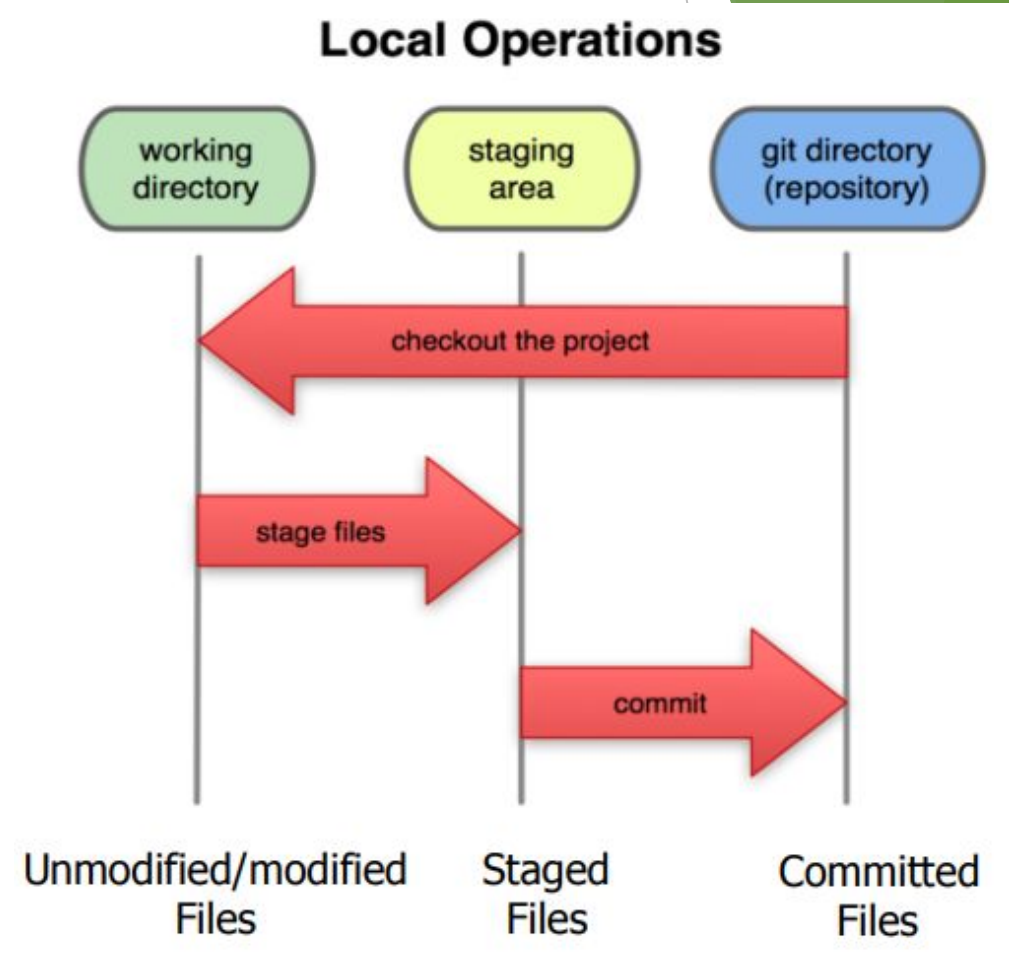
Git snapshots

- Centralized VCS like Subversion track version data on each individual file.
- Git keeps "snapshots" of the entire state of the project.
 - Each checkin version of the overall code has a copy of each file in it.
 - Some files change on a given checkin, some do not.
 - More redundancy, but faster.



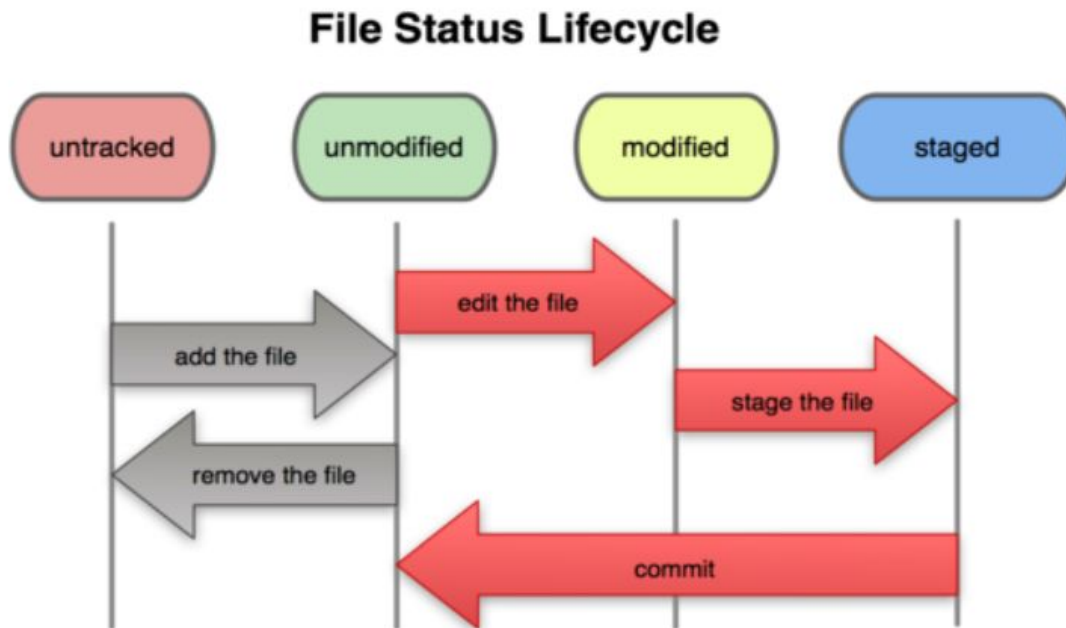
Local git areas

- 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.



Git commit checksums

- In Subversion each modification to the central repo increments the version # of the overall repo.
 - In Git, each user has their own copy of the repo, and commits changes to their local copy of the repo before pushing to the central server.
 - So Git generates a unique SHA-1 hash (40 character string of hex digits) for every commit.
 - Refers to commits by this ID rather than a version number.
 - Often we only see the first 7 characters:
 - `1677b2d Edited first line of readme`
 - `258efa7 Added line to readme`
 - `0e52da7 Initial commit`

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
<code>git clone <i>url</i> [<i>dir</i>]</code>	copy a Git repository so you can add to it
<code>git add <i>file</i></code>	adds file contents to the staging area
<code>git commit</code>	records a snapshot of the staging area
<code>git status</code>	view the status of your files in the working directory and staging area
<code>git diff</code>	shows diff of what is staged and what is modified but unstaged
<code>git help [<i>command</i>]</code>	get help info about a particular command
<code>git pull</code>	fetch from a remote repo and try to merge into the current branch
<code>git push</code>	push your new branches and data to a remote repository
others: <code>init</code> , <code>reset</code> , <code>branch</code> , <code>checkout</code> , <code>merge</code> , <code>log</code> , <code>tag</code>	

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 Hello.java Goodbye.java`
 - 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 #22"`
- 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)
 - 1677b2d Edited first line of readme
 - 258efa7 Added line to readme
 - 0e52da7 Initial commit
 - `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

Merge conflicts arise when two members of the same development team work on the same file and try to merge in their respective changes.

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).

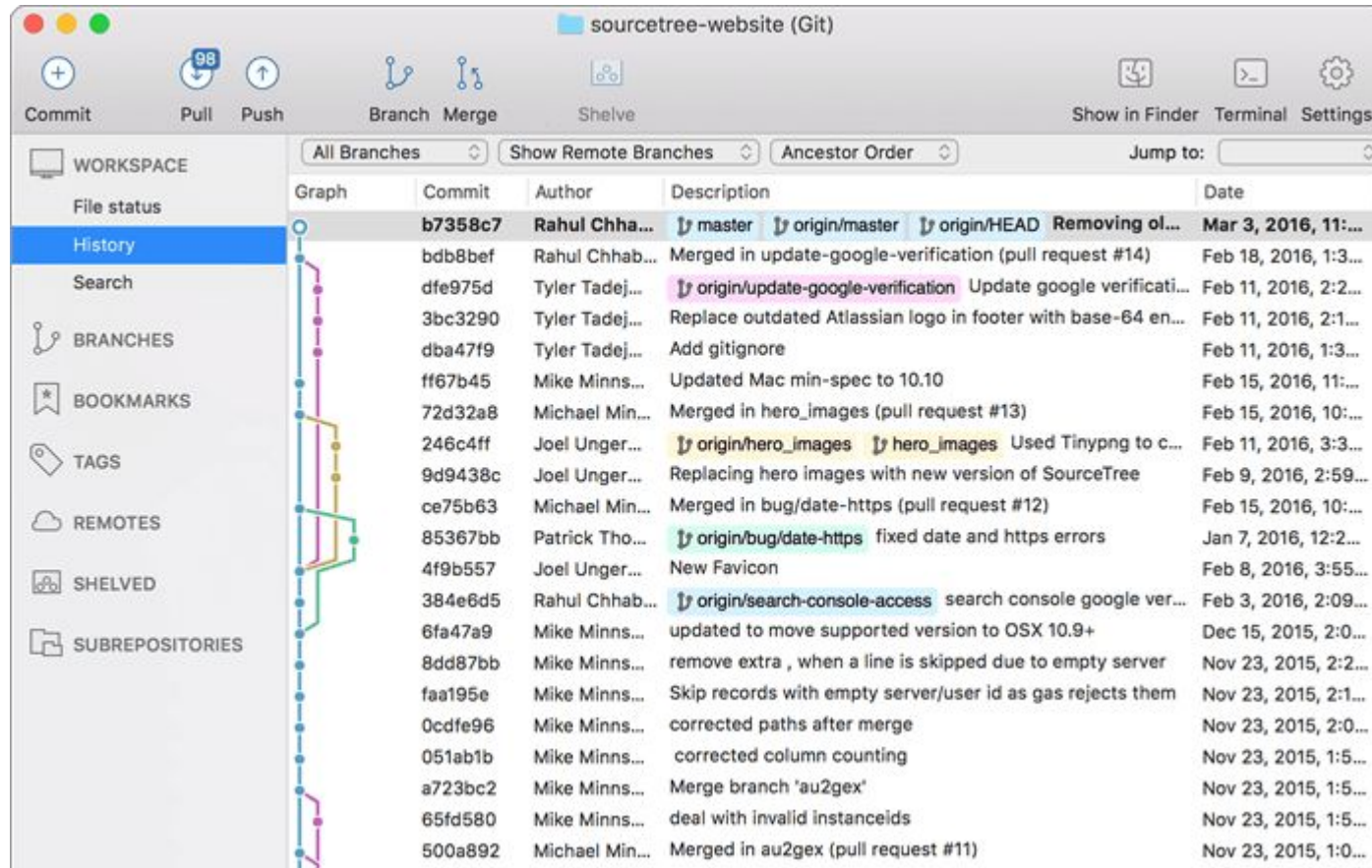
Interaction w/ remote repo

- **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`

GitHub

- GitHub.com is a site for online storage of Git repositories.
 - You can create a remote repo there and push code to it.
 - Many open source projects use it, such as the Linux kernel.
- Question: Do I always have to use GitHub to use Git? - Answer: No! You can use Git locally for your own purposes.
 - Or you or someone else could set up a server to share files.

Tools (SourceTree)



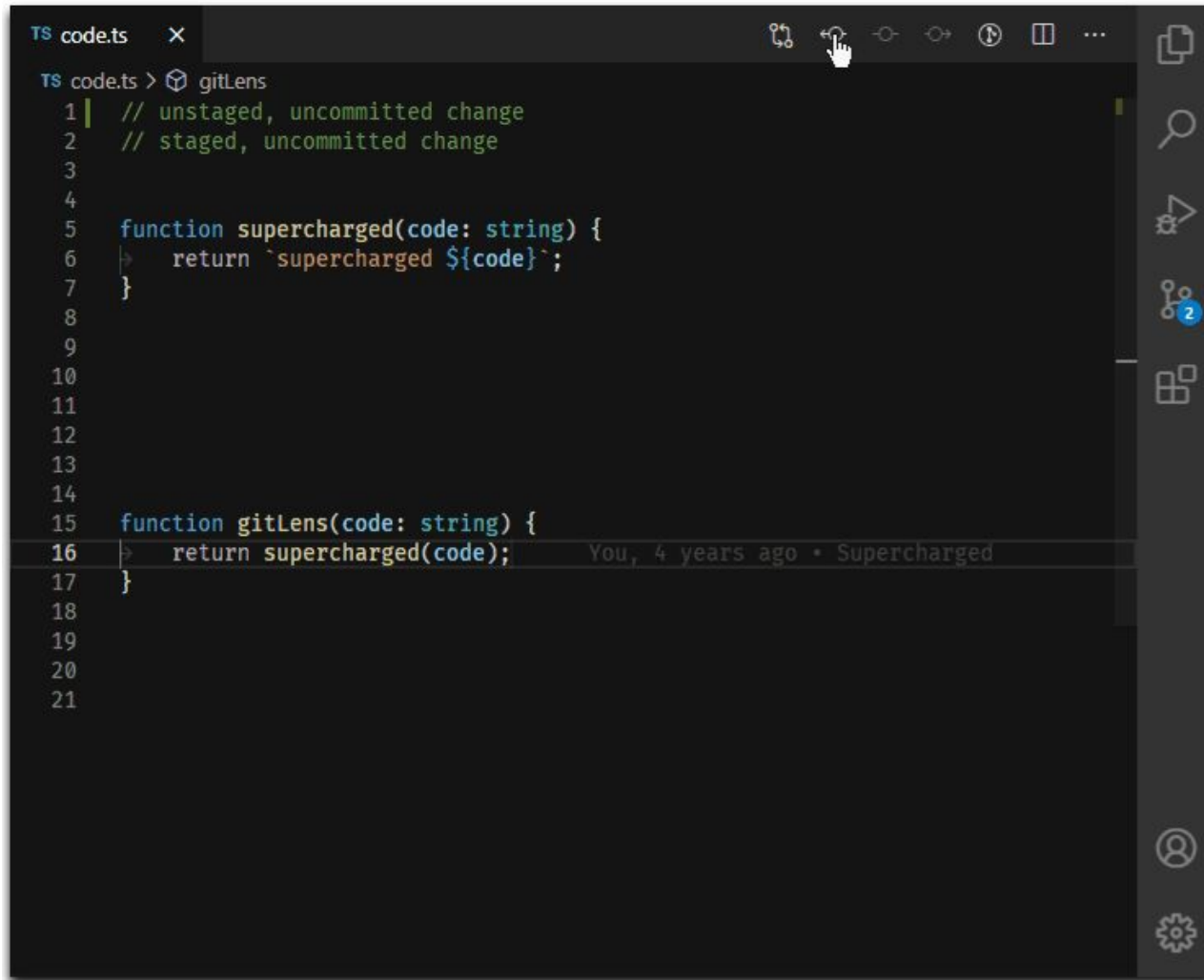
Tools (GitKraken)

The screenshot displays the GitKraken application interface, which is used for managing Git repositories. The interface is divided into several sections:

- Left Panel:** Contains a sidebar with repository information (electron, master branch), a list of local and remote branches, pull requests, and issues. It also includes a section for selecting an issue tracker (GitHub, GitHub Enterprise, GitKraken Boards, GitLab, GitLab Self-Managed, Jira Cloud, Jira Server, Trello, or None).
- Central Panel:** Shows a graphical representation of the commit history, with branches and merges visualized as lines and nodes. The commit messages are listed on the right side of this panel.
- Right Panel:** Provides a detailed view of the selected commit (feat(extensions): expose ExtensionRegistryObserver events in Session (#25385)). It includes the commit message, the author (Samuel Maddock), the date (9/23/2020 @ 12:29 PM), and a list of modified files (docs/api/session.md, shell/browser/api/electron_api_session.cc, shell/browser/api/electron_api_session.h, spec-main/extensions-spec.ts).

The interface is dark-themed and includes various navigation and action buttons at the top, such as Undo, Redo, Fetch, Push, Branch, Stash, and Pop. The bottom status bar shows the current view (100%), a Feedback button, and the version number (7.3.2).

Tools (VSCode GitLens)



```
TS code.ts > gitLens
1 | // unstaged, uncommitted change
2 | // staged, uncommitted change
3 |
4 |
5 | function supercharged(code: string) {
6 |   return `supercharged ${code}`;
7 | }
8 |
9 |
10 |
11 |
12 |
13 |
14 |
15 | function gitLens(code: string) {
16 |   return supercharged(code);
17 | }
18 |
19 |
20 |
21 |
```

You, 4 years ago • Supercharged

Read more...

- GitFlow
- Trunk-based Development
- Conventional Commits
- Continuous Integration
- Continuous Delivery
- DevOps

References

- ▶ <https://git-scm.com/>
- ▶ <https://github.com/>
- ▶ <https://www.sokanacademy.com>