CSE 15L:
Software Tools and Techniques Laboratory

Winter 2019 - http://ieng6.ucsd.edu/~cs15x

Dr. ILKAY ALTINTAS Dr. BEN OCHOA

Lecture 10

February 11th, 2019

Today's Topics

1. Software Version Control
2. Introduction to Git
3. More UNIX Scripting

Version Control

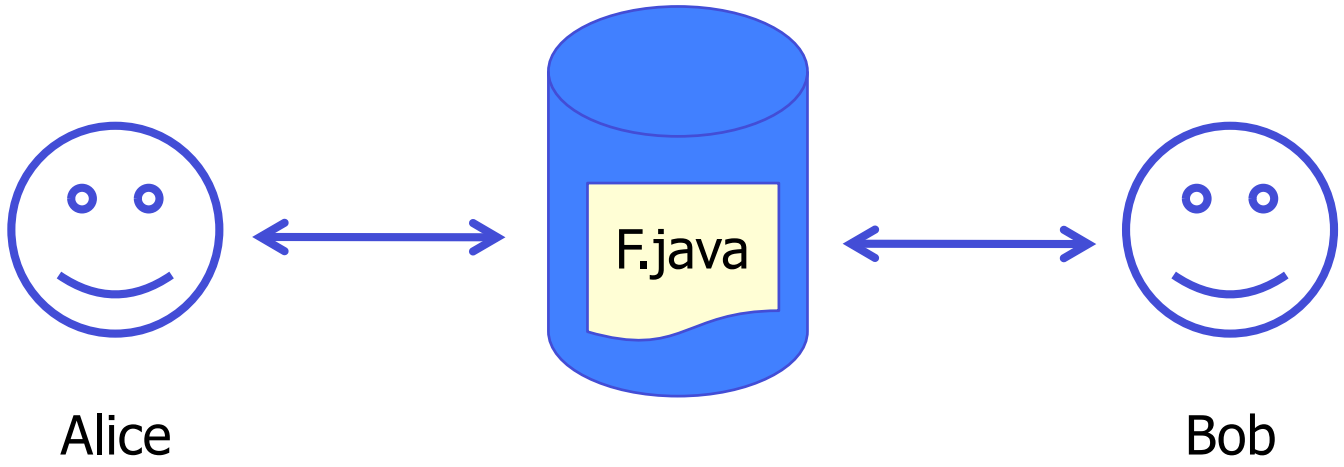
Relevance:
Why do we need a version control system?

- To collaborate with other developers.
- To recover to an earlier version of your code or a prior release if needed.
- To improve communication on the changes to the project.

Basic version control

Scenario

Assume now, you are developing in a small team.
The team uses a regularly backed-up file-sharing mechanism (such as a shared network drive).
What can go wrong?

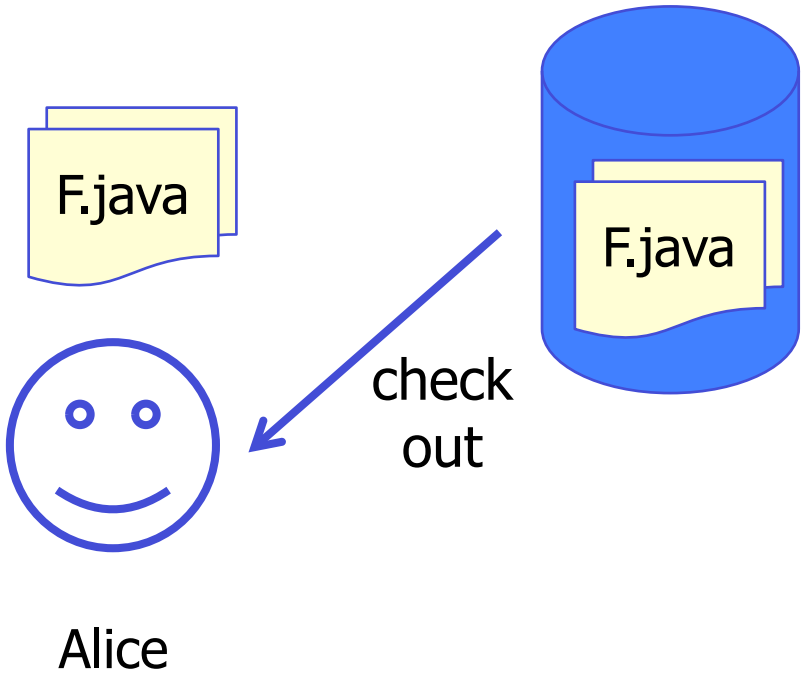


Solution: Version Management with Dedicated Repository

Introduce a Version Management Repository that holds the **master copy** of all artifacts (code, tests, build files, graphics, …) of your project

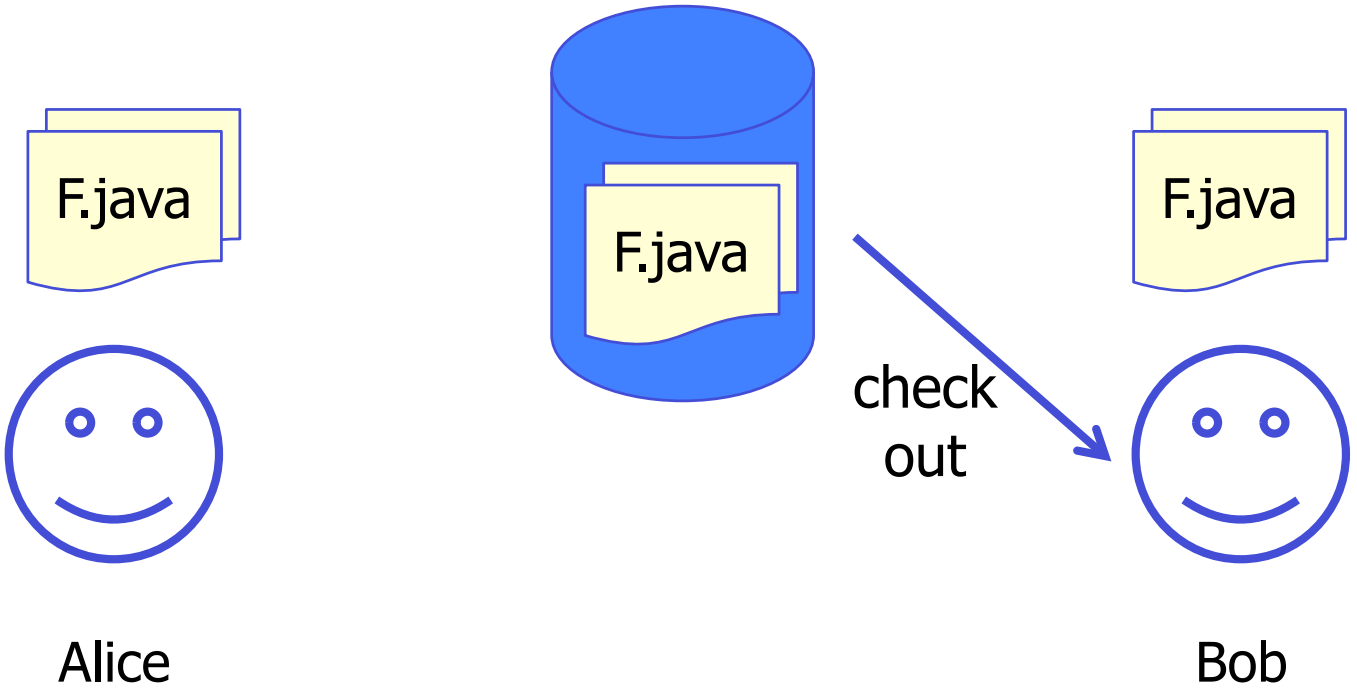
Solution: Version Management with Dedicated Repository

If Alice needs access to the code, she "**checks out**" a **local copy**



Solution: Version Management with Dedicated Repository

If Bob also needs access to the code, he also "**checks out**" a **local copy**

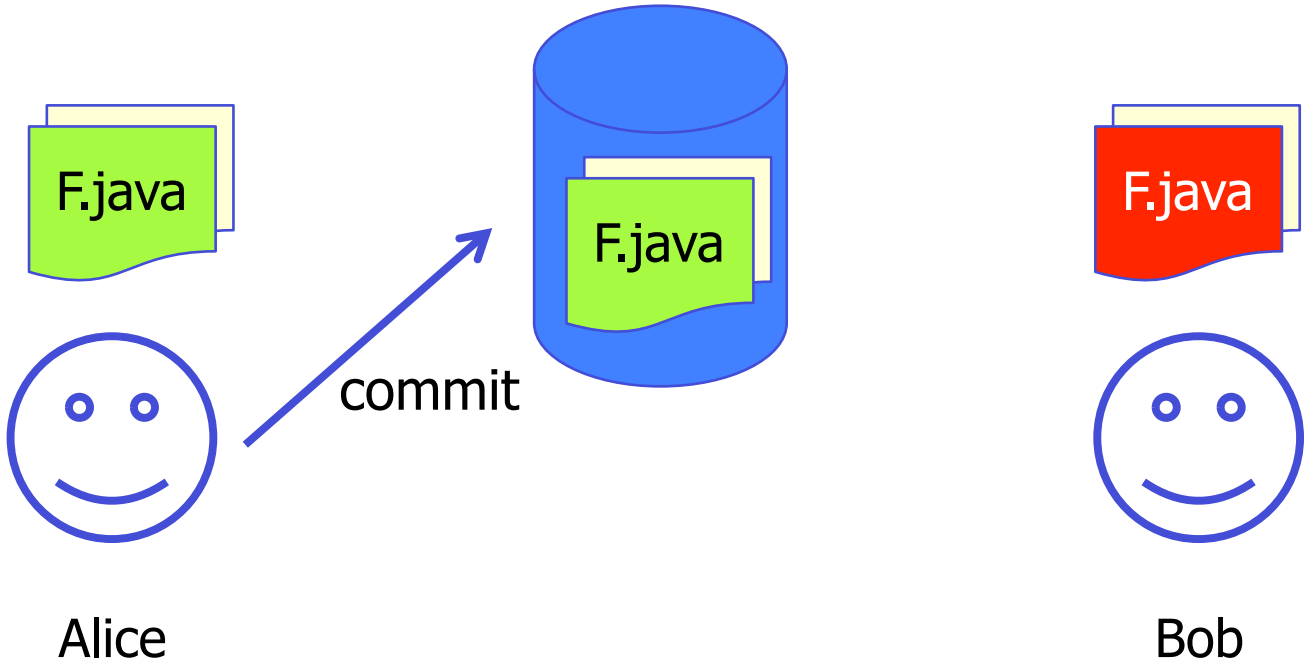


Solution: Version Management with Dedicated Repository

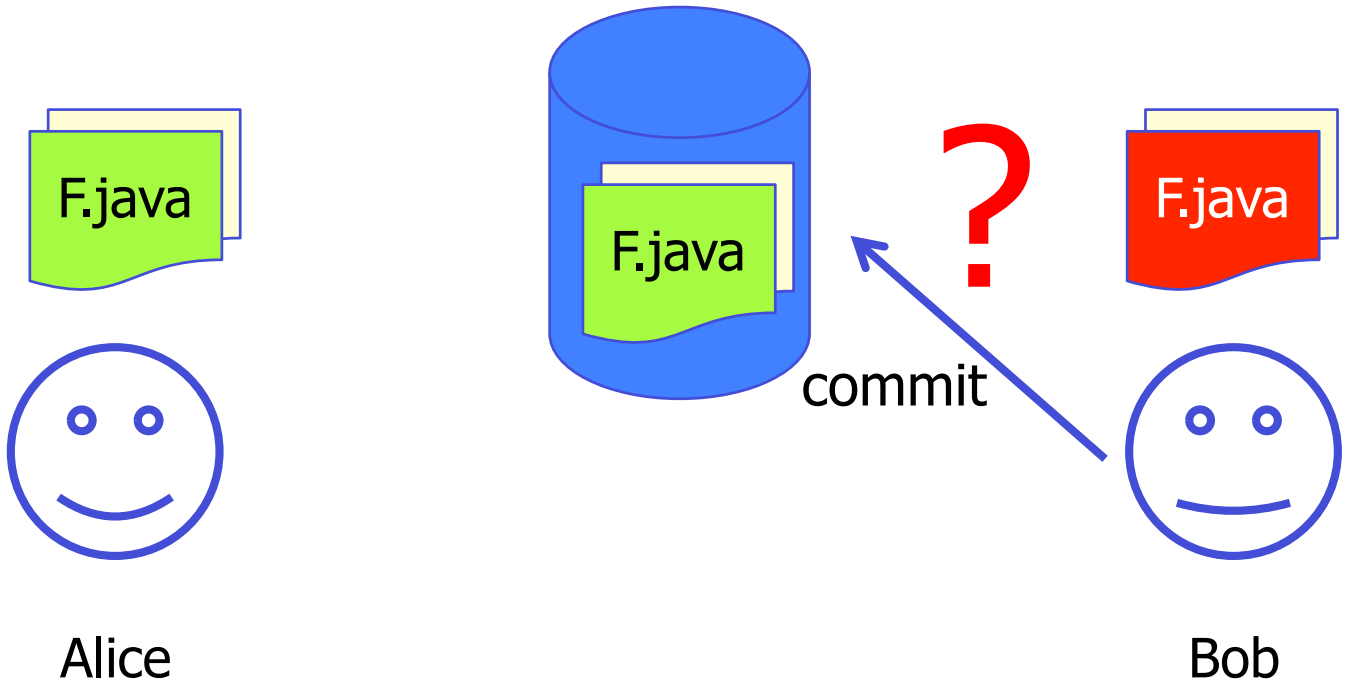
Bob and Alice **individually** make changes to **their** own **local copy**

Solution: Version Management with Dedicated Repository

When Alice is done with her edits, she **"commits"** her local copy **to the repository**. Her changes are automatically merged into the master copy of the file in the repository.

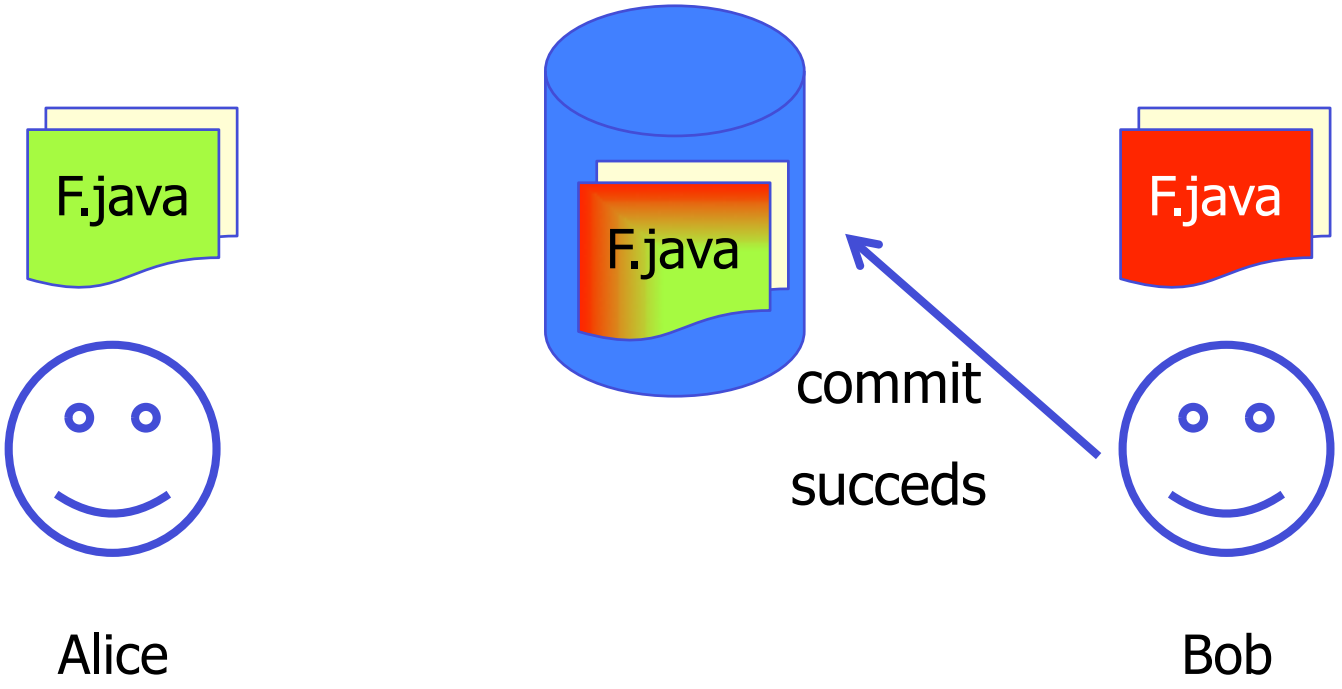


Now Bob is done with his edits, he also wants to **"commit."** What can happen?



Scenario 1: Alice and Bob have edited in **different** areas of the file – **No conflict**!

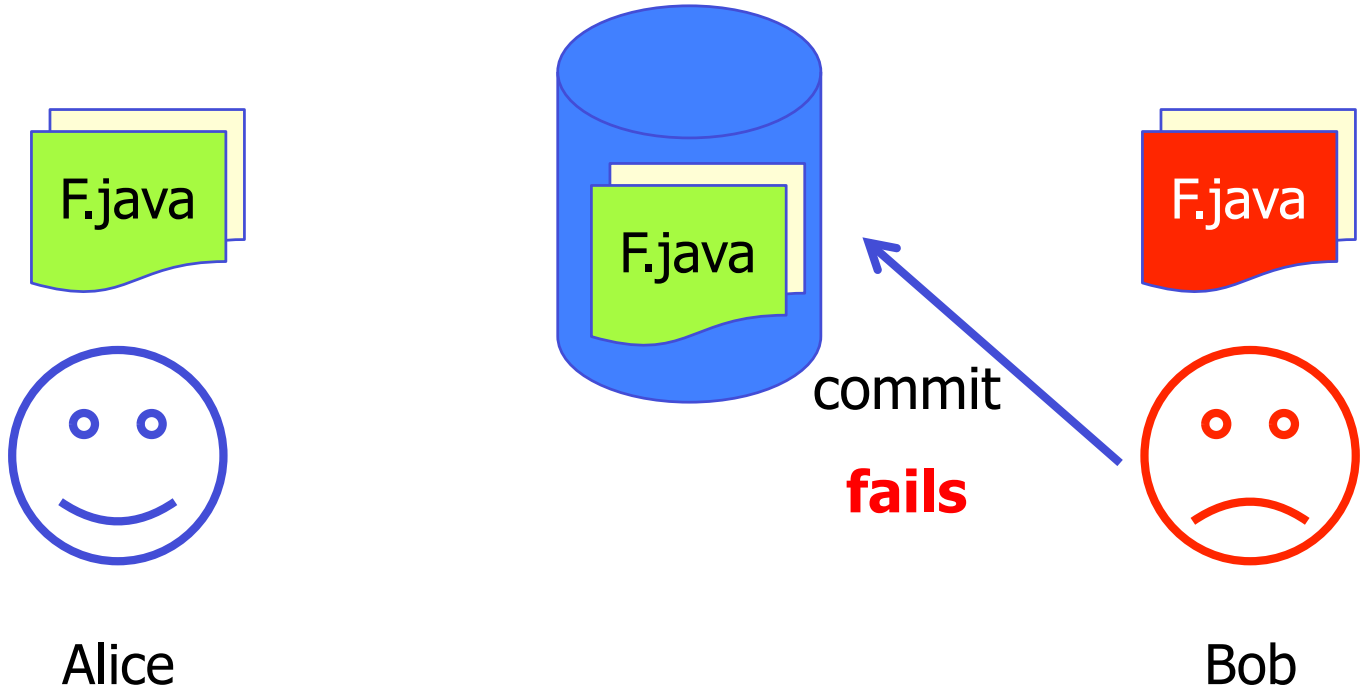
Bob's changes will be merged into the master copy.



Solution: Version Management with Dedicated Repository

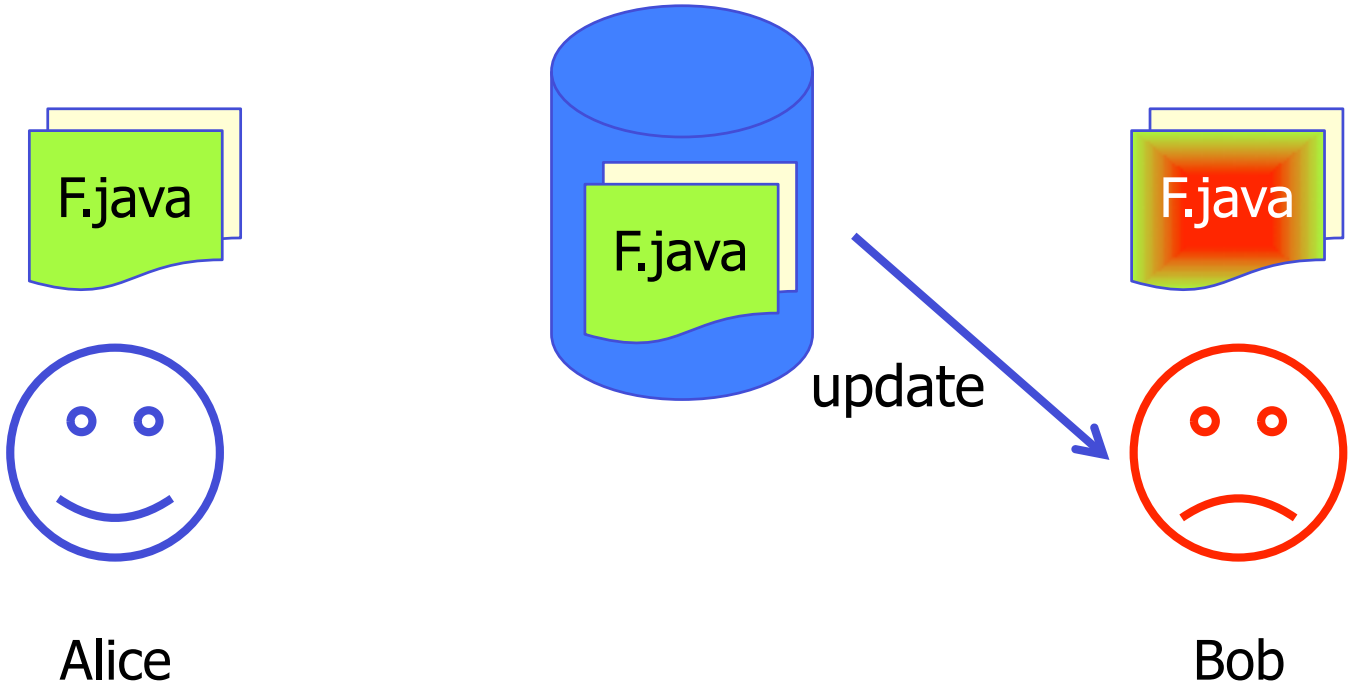
Scenario 2: Alice and Bob have edited in the **same** areas of the file – **Conflict**!

Bob's changes will **NOT** be merged into the master copy.



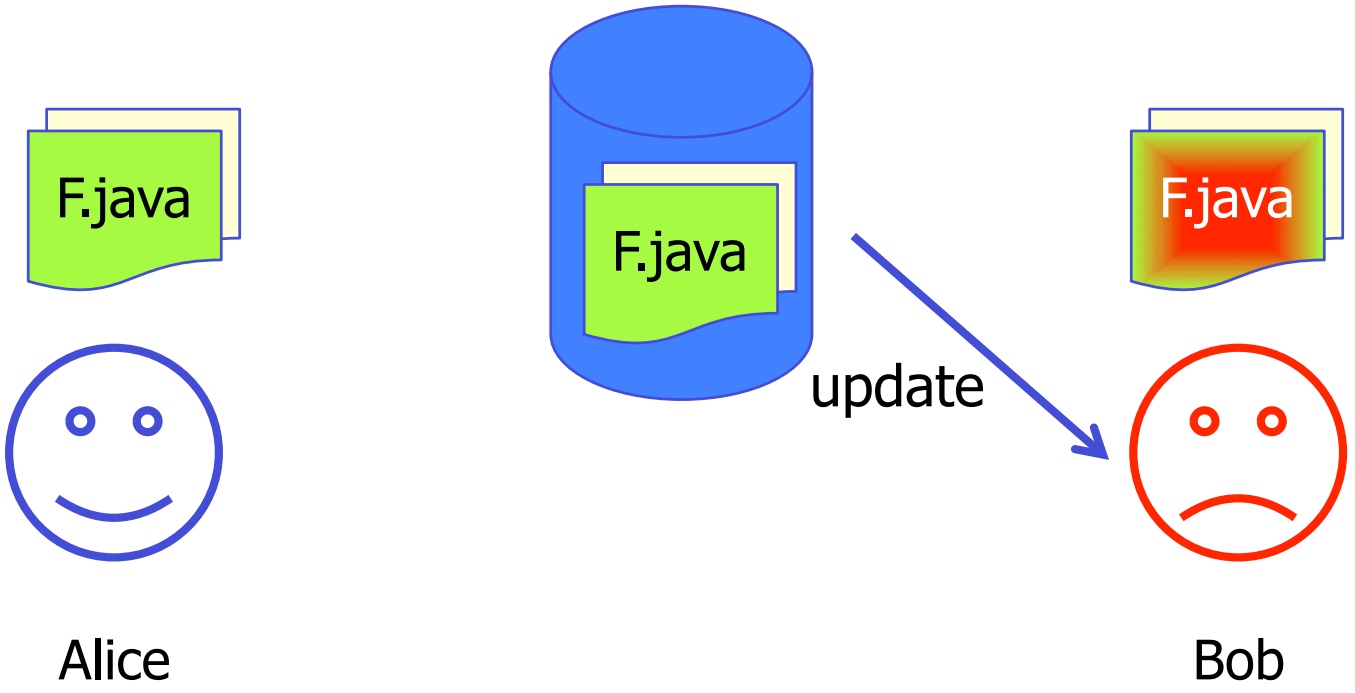
Solution: Version Management with Dedicated Repository

To resolve the conflict, Bob first "updates" his local copy.



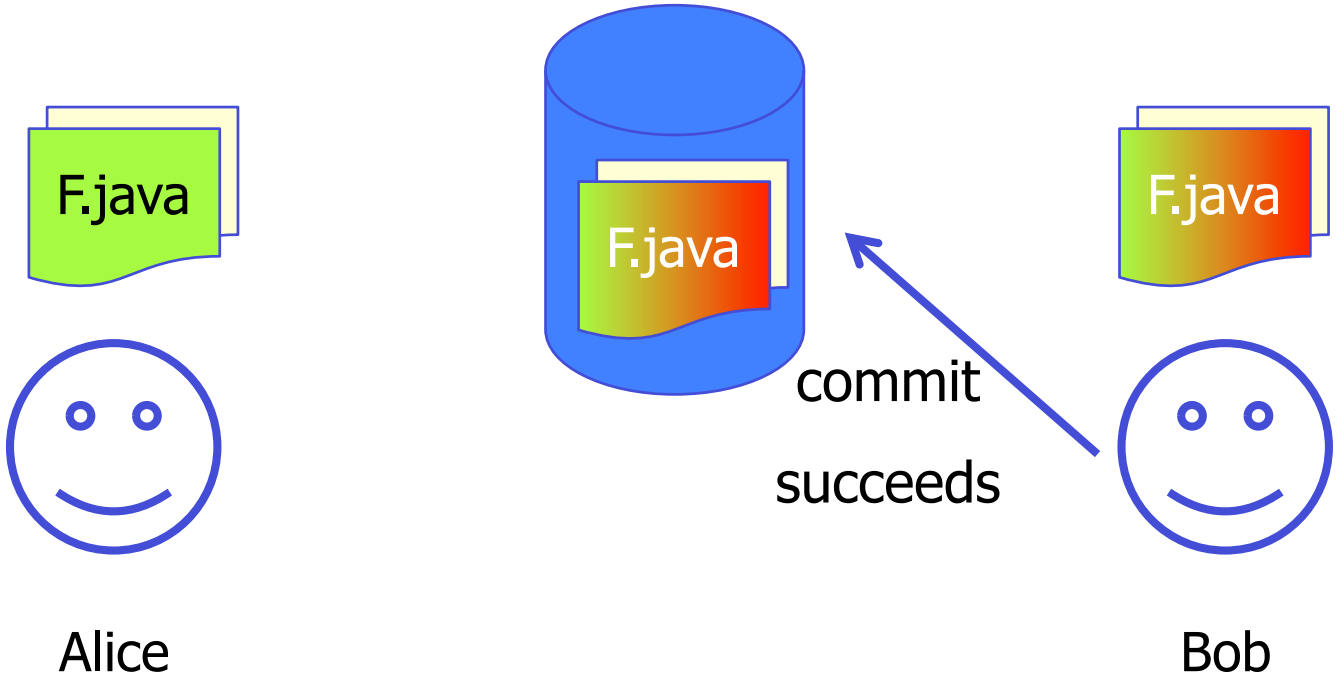
Solution: Version Management with Dedicated Repository

He gets notifications of the conflicts **inside the affected files**. He makes decisions to resolve the conflicts.



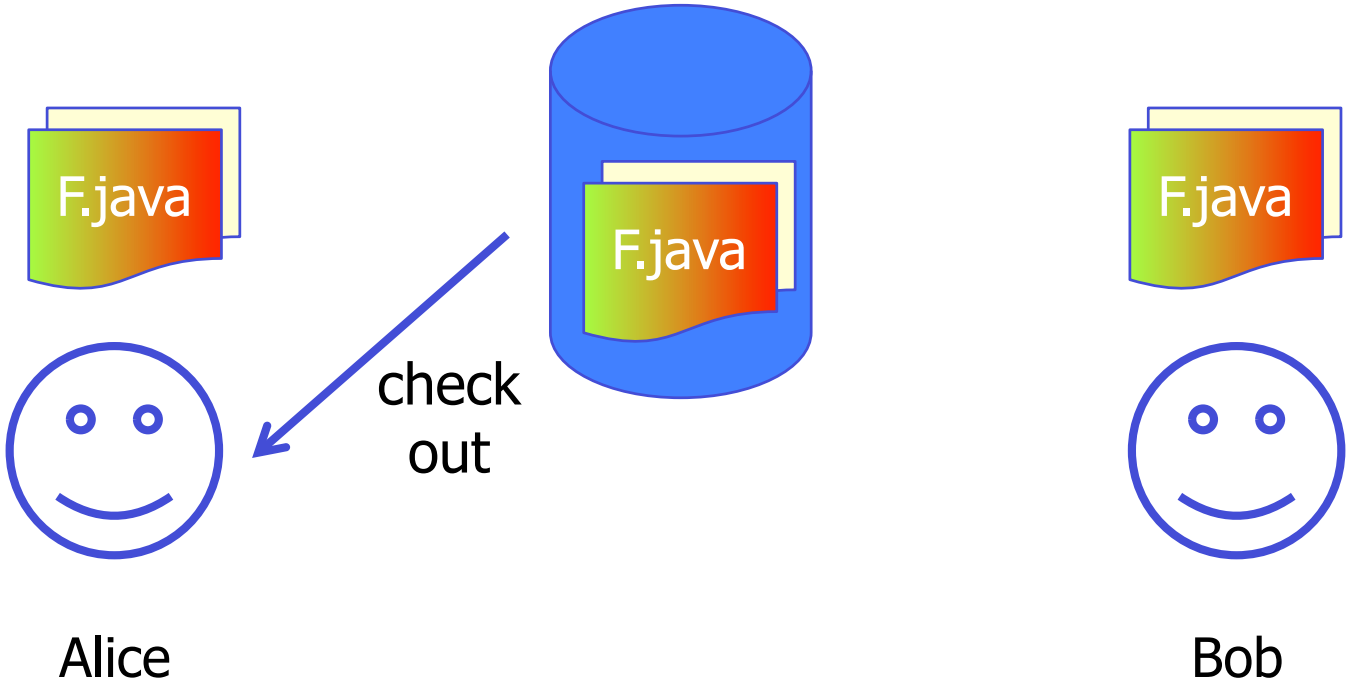
Solution: Version Management with Dedicated Repository

Once he has resolved all conflicts, Bob commits again.
This time he succeeds!

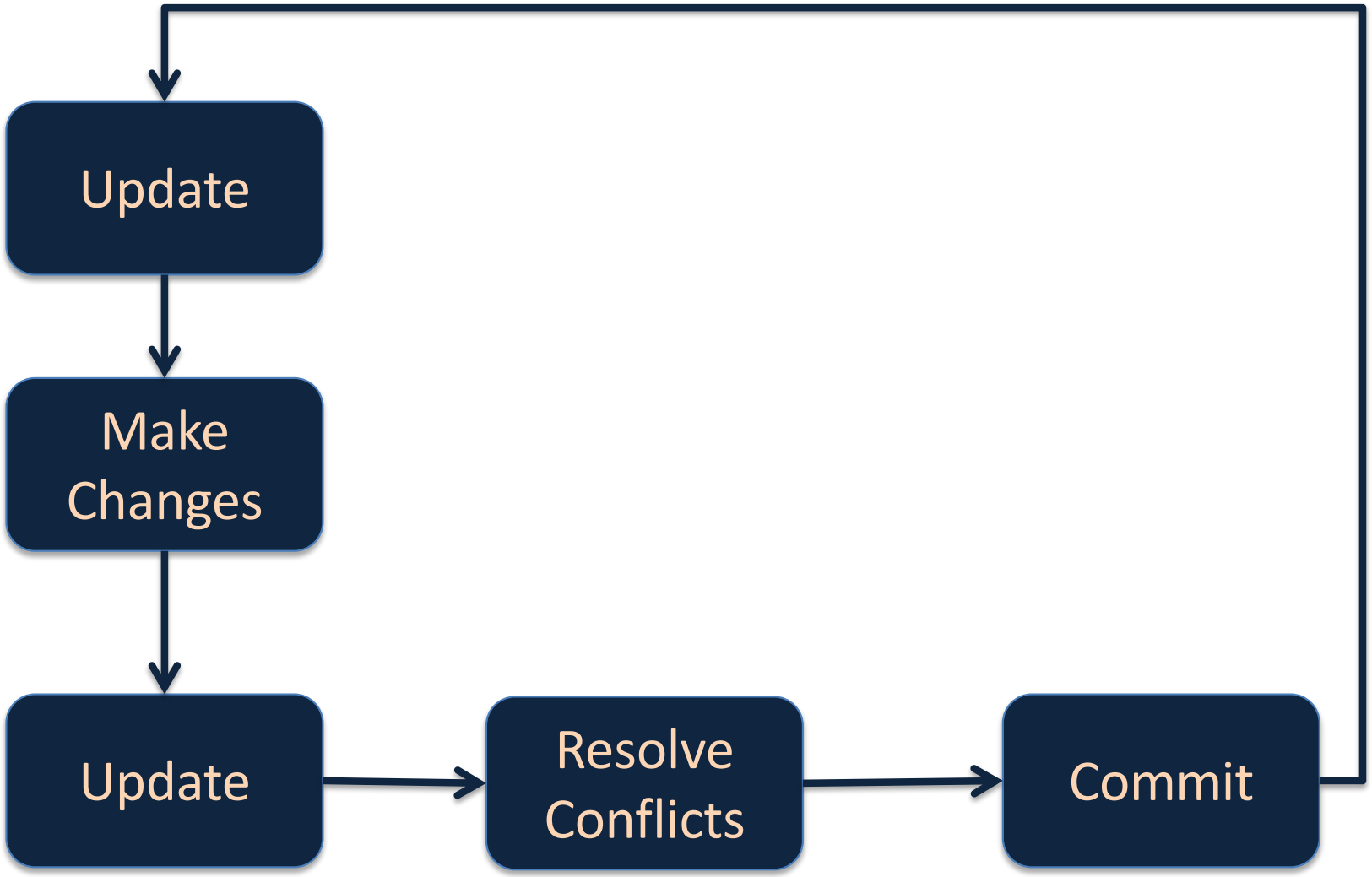


Solution: Version Management with Dedicated Repository

Alice updates at the beginning of her next work increment.



Software Version Control Workflow



Basic Intro to Git

Git Resources

- At the command line: (where verb = config, add, commit, etc.)
  $ git help <verb>
  $ git <verb> --help
  $ man git-<verb>

- Free on-line book: http://git-scm.com/book
- Git tutorial: http://schacon.github.com/git/gittutorial.html
- Reference page for Git: http://gitref.org/index.html
- Git website: http://git-scm.com/

- Git for Computer Scientists (http://eagain.net/articles/git-for-computer-scientists/)

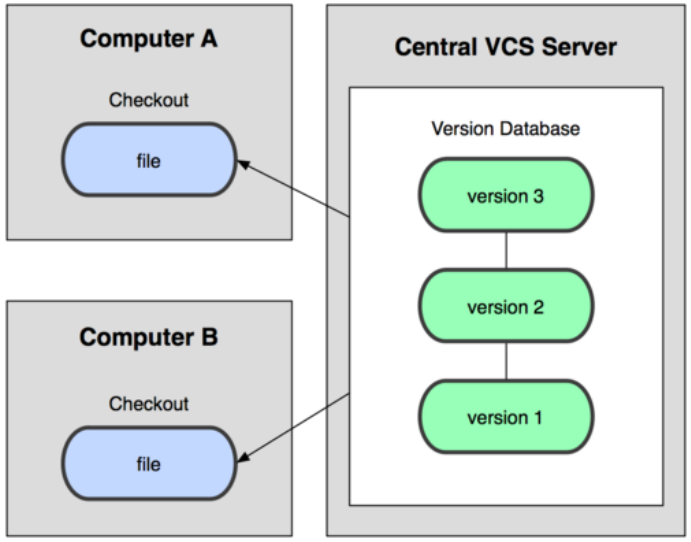
- Top 10 Git tutorials for beginners (http://sixrevisions.com/resources/git-tutorials-beginners/)

Git History

- Came out of Linux development community
  - Linus Torvalds, 2005
- Initial goals:
  - Speed
  - Support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects like Linux efficiently

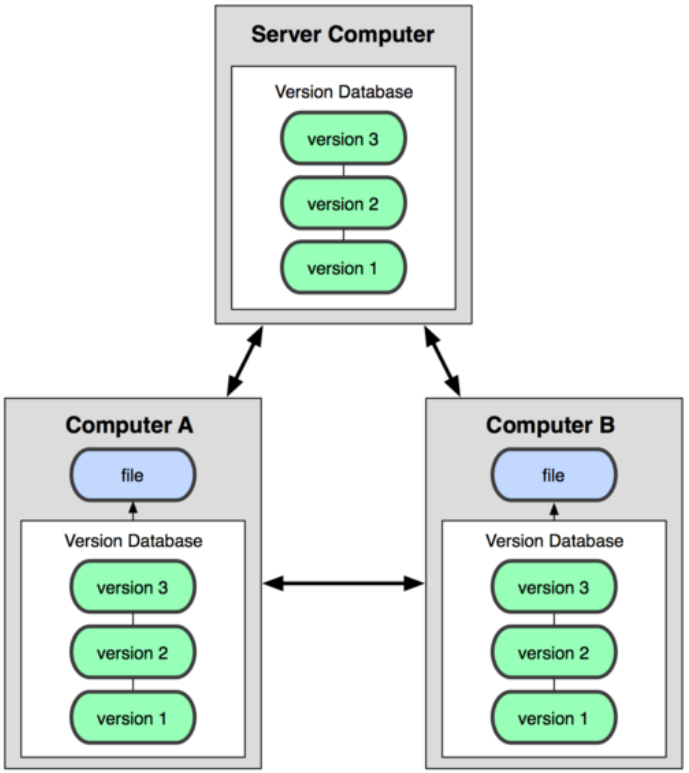
Git uses a distributed model

Centralized Model



(CVS, Subversion, Perforce)

Distributed Model

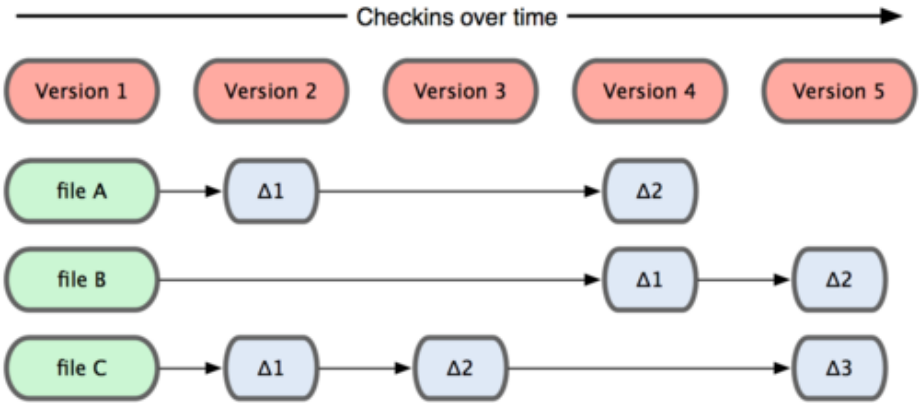


(Git, Mercurial)

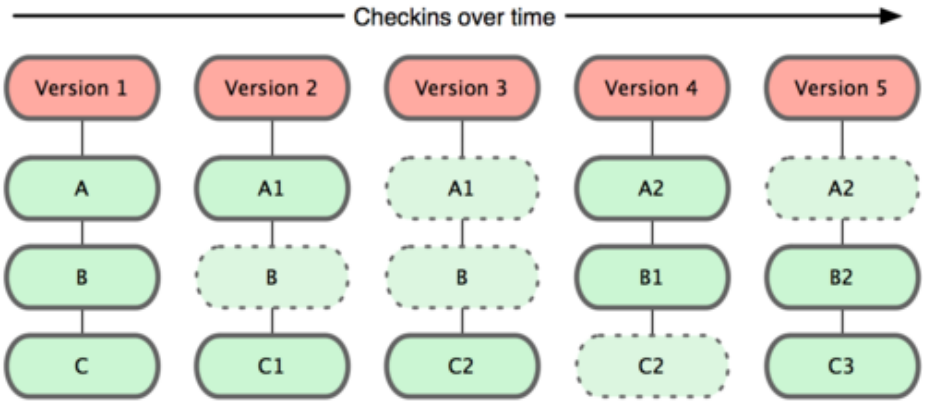
Result: Many operations are local

Git takes snapshots

Subversion



Git



Git uses checksums

- In Subversion each modification to the **central** repo incremented the version # of the overall repo.

- How will this numbering scheme work **when 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?

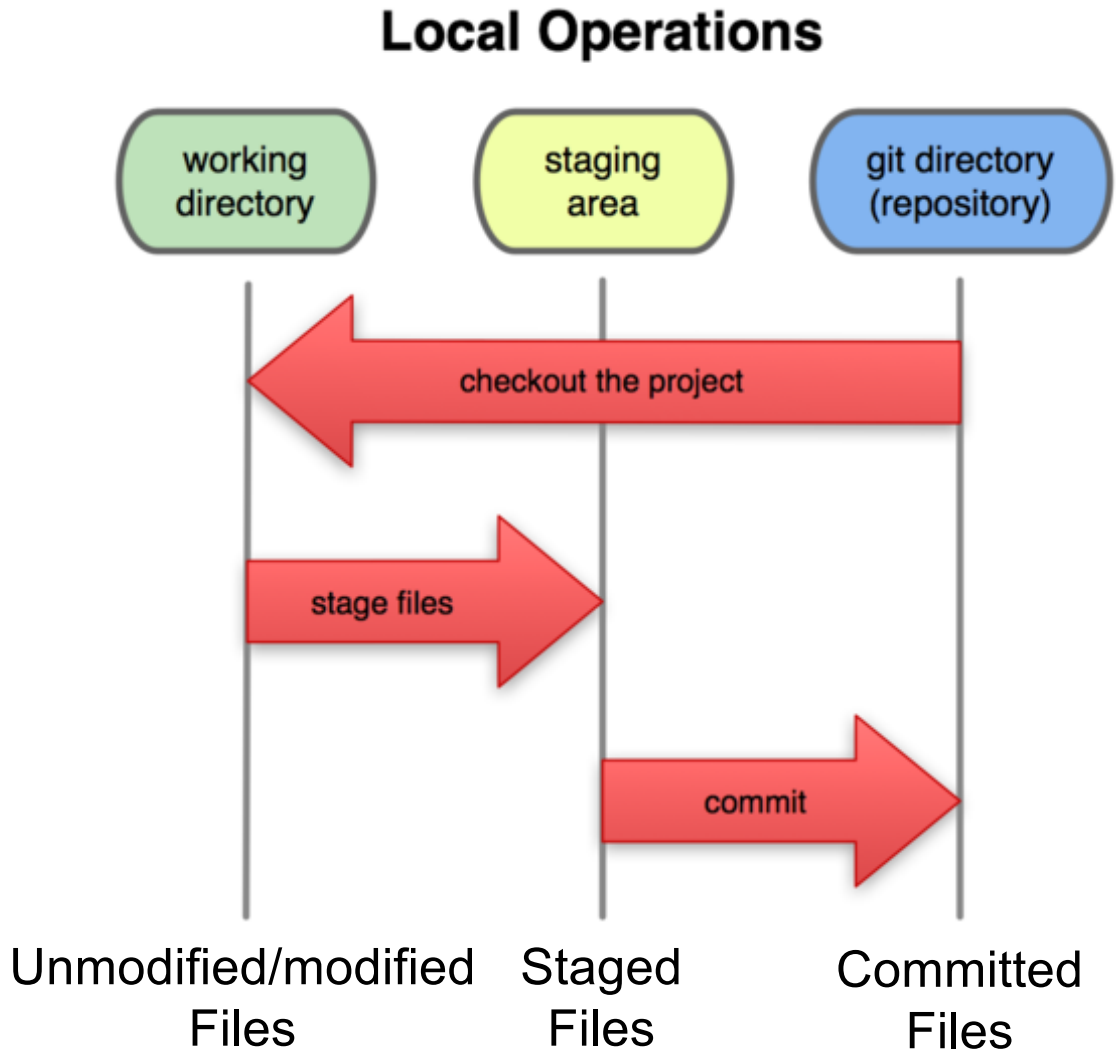
- Instead, Git generates a unique SHA-1 hash – 40 character string of hex digits, for every commit. Refer to commits by this ID rather than a version number. Often we only see the first 7 characters:

  1677b2d Edited first line of readme

  258efa7 Added line to readme

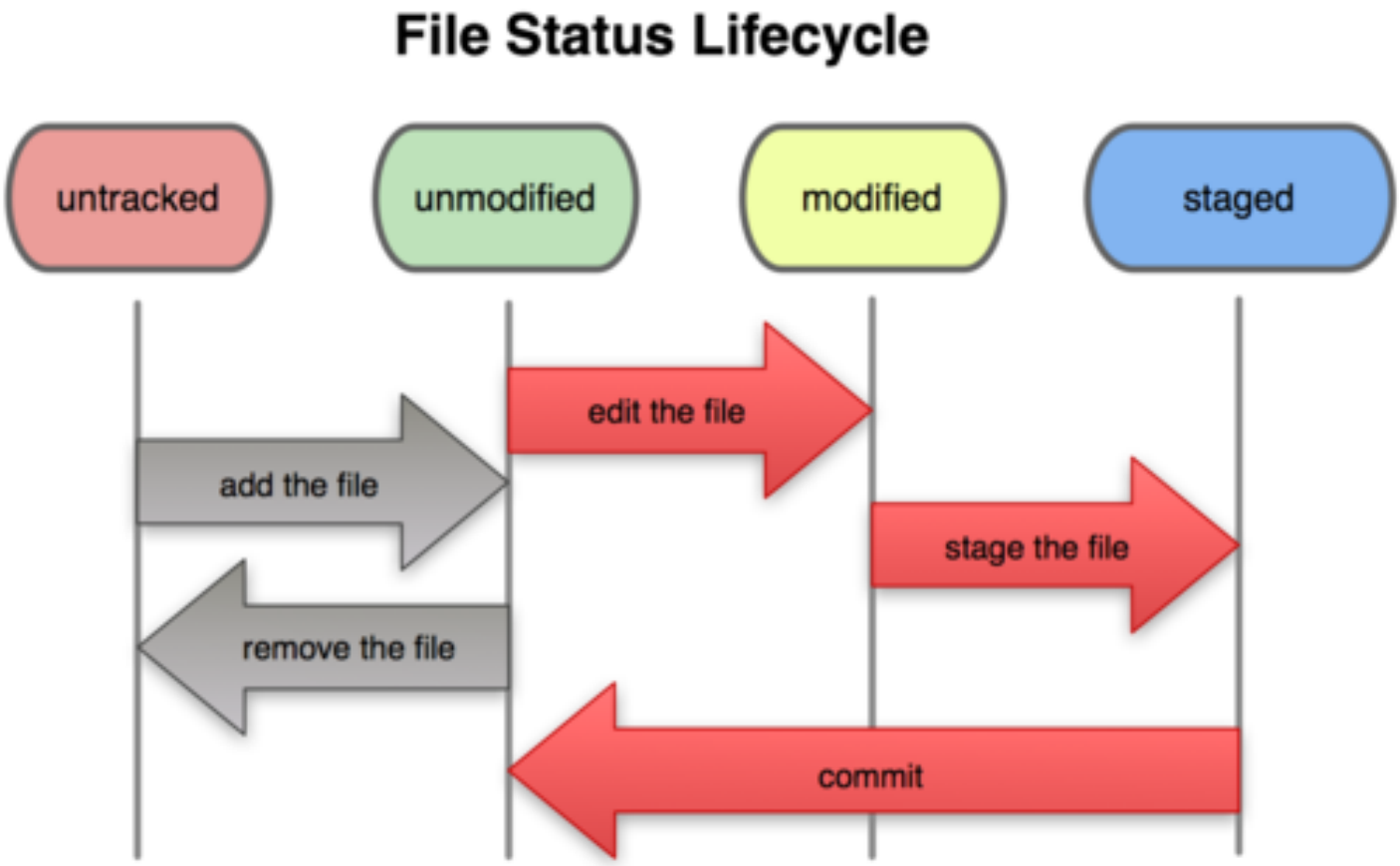
  0e52da7 Initial commit

A Local Git project has three areas



Note: working directory sometimes called the “working tree”, staging area sometimes called the “index”.

A Git file lifecycle



Aside: So what is github?

- GitHub.com is a site for online storage of Git repositories.
- Many open source projects use it, such as the Linux kernel.
- You can get free space for open source projects or you can pay for private projects.

Question: Do I have to use github to use Git?

Answer: No!

- you can use Git completely locally for your own purposes, or
- you or someone else could set up a server to share files, or
- you could share a repo with users on the same file system

Get ready to use Git!

1. 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.
- These will be set globally for all Git projects you work with.
- You can also set variables on a project-only basis by not using the `--global` flag.
- You can also set the editor that is used for writing commit messages:
  $ git config --global core.editor emacs (it is vim by default)

Create a local copy of a repo

2. Two common scenarios: (only do one of these)
   a) To **clone an already existing repo** to your current directory:

```
$ git clone <url> [local dir name]
```

This will create a directory named *local dir name*, containing a working copy of the files from the repo, and a **`.git`** directory (used to hold the staging area and your actual repo)

   b) To **create a 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 file1.java
$ git commit –m “initial project version”
```

Git commands

command	description
`git clone` ***`url [dir]`***	copy a git repository so you can add to it
`git add` ***`files`***	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`	

Git Interview Questions

- What is the difference between GIT and a centralized code version system, e.g., SVN?
- What is “Staging Area” or “Index” in GIT?
- What is the function of git clone?
- What is ‘git status’ is used for?
- Explain what is commit message? How can you change it?

More Git Reference Slides

Committing files

- 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 README.txt hello.java
```

This takes a snapshot of these files at this point in time and adds it to the staging area.

- To move staged changes into the repo we commit:

```
$ git commit –m “Fixing bug #22”
```

Note: To unstage a change on a file before you have committed it:

```
$ git reset HEAD  --  filename
```

Note: To unmodify a modified file:

```
$ git checkout  --  filename
```

Note: These commands are just acting on **your local version of repo**.

Status and Diff

- To view the **status** of your files in the working directory and staging area:

```
$ git status
```
or
```
$ git status -s
```
(`-s` shows a short one line version similar to svn)

- To see what is modified but unstaged:

```
$ git diff
```

- To see staged changes:

```
$ git diff --cached
```

Pulling and Pushing

Good practice:

1. **Add** and **Commit** your changes to your local repo
2. **Pull** from remote repo to get most recent changes (fix conflicts if necessary, add and commit them to your local repo)
3. **Push** your changes to the remote 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 push your changes from your local repo to the remote repo:

```
$ git push origin master
```

Notes: `origin` = an alias for the URL you cloned from
`master` = the remote branch you are pulling from/pushing to,
(the local branch you are pulling to/pushing from is your current branch)

Branching

To create a branch called experimental:

- `$ git branch experimental`

To list all branches: (* shows which one you are currently on)

- `$ git branch`

To switch to the experimental branch:

- `$ git checkout experimental`

Later on, changes between the two branches differ, to merge changes from experimental into the master:

- `$ git checkout master`
- `$ git merge experimental`

Note: `git log --graph` can be useful for showing branches.

Note: These branches are in *your local repo*!