cis112

Version Control Systems (Git)

BBBF

Yeditepe University

V2025-02-17

Agenda

- Why use Version (Source) Control Systems?
- What are Git and GitHub?
- Basic Git Commands

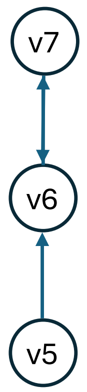
Motivation

Motivation
for the
Single Programmer

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?*

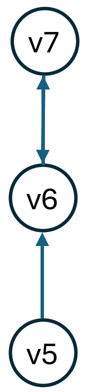
- Has this ever happened to you?



Why version control? (part 2)

- 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*

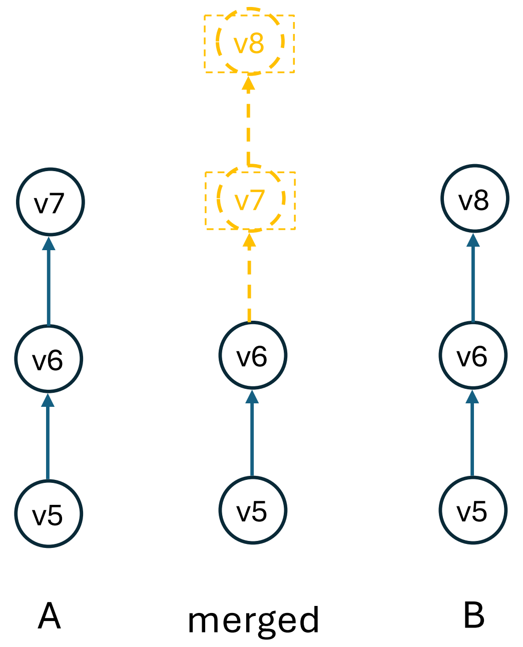
- Has this ever happened to you?



Motivation
for a
Team of Programmers

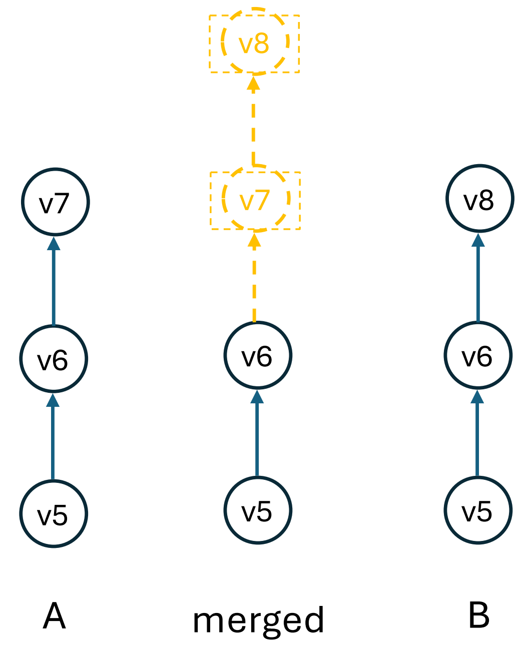
Version control for teams

- 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?



Teams (part 2)

- 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?



Version Control Systems

Version control systems

- A version control system (often called a source code control system) does these things:
  - Keeps multiple versions (older and newer) of everything (not just source code)
  - Requests comments regarding every change
  - Allows “check in” and “check out” of files so you know which files someone else is working on
  - Displays differences between versions

Benefits 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

Git and GitHub

What are Git and GitHub?

- Git is a free and open source, distributed **version control system** designed to handle everything from small to very large projects with speed and efficiency.
- GitHub is a **web-based** Git repository **hosting service**, which offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features. GitHub is also free (as in free beer).

GitHub

- https://github.com/
- When looking for a software developer position, host substantial projects you have developed on GitHub and give a link in your resume.

About Git

Created by Linus Torvalds, who is
the creator of Linux kernel, 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

(A "git" is a cranky old man. Linus meant himself.)

Centralized VCS

In Subversion, CVS, Perforce, etc., a central server repository (repo) holds the "official copy" of the code

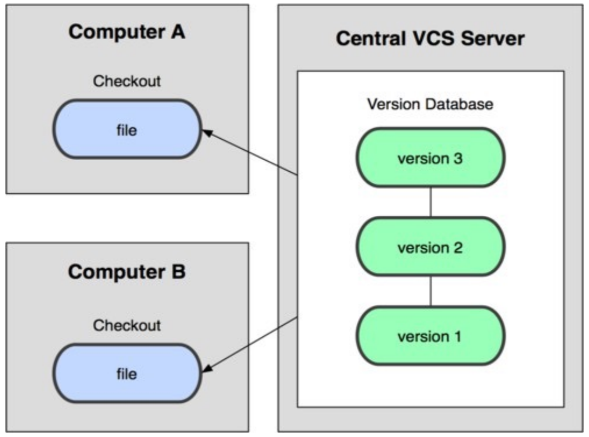
- the server maintains the sole version history of the repo

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

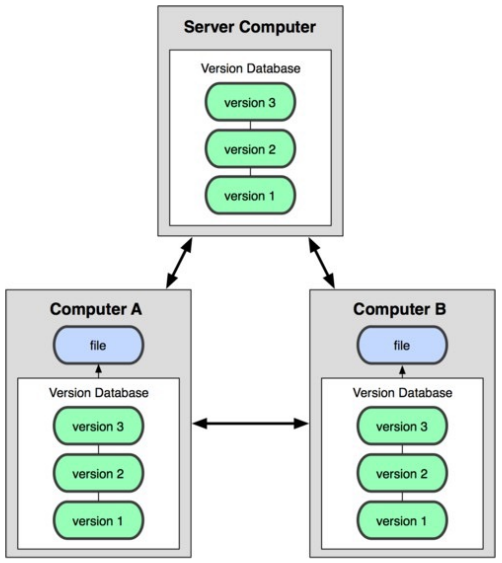
Your local repo is a complete copy of everything on the remote server

- 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



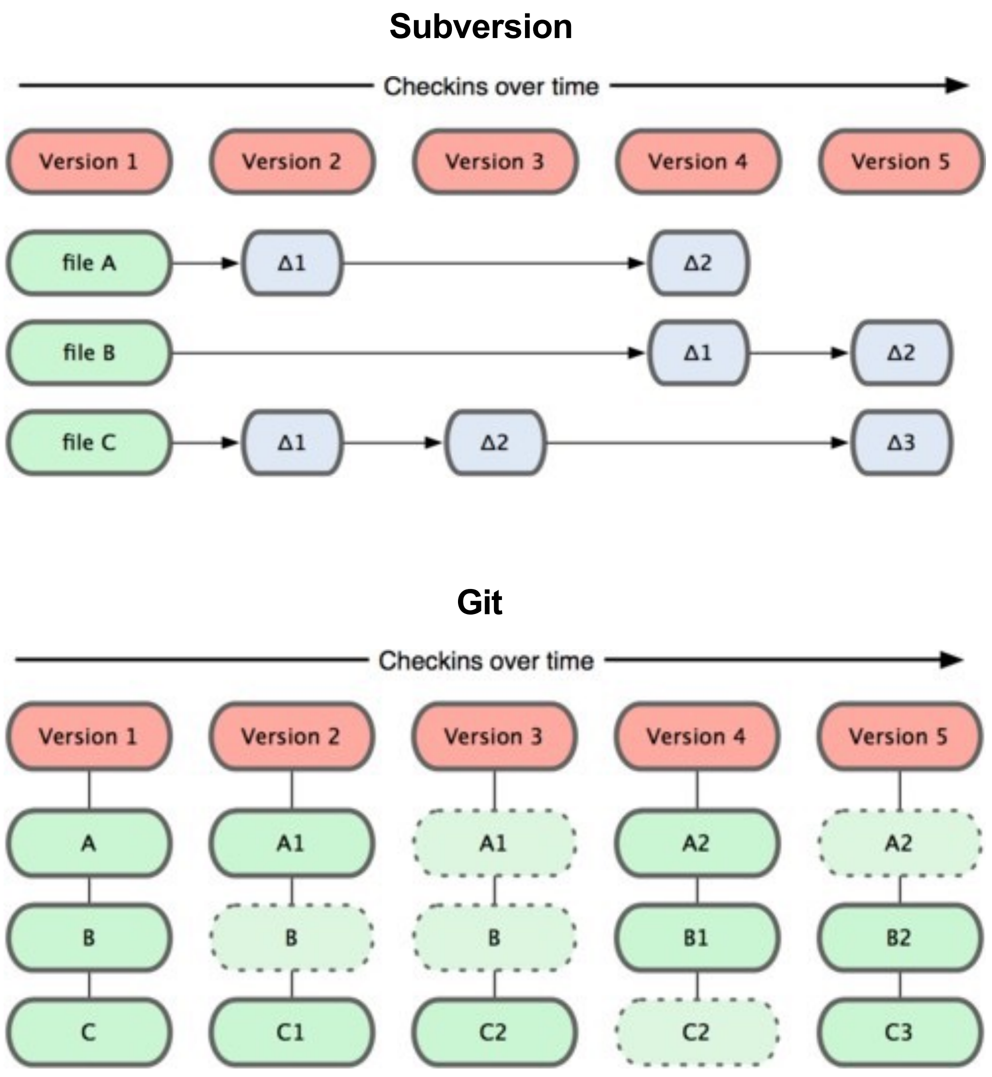
Git

Git snapshots

Centralized VCS like Subversion track version data on each individual file.

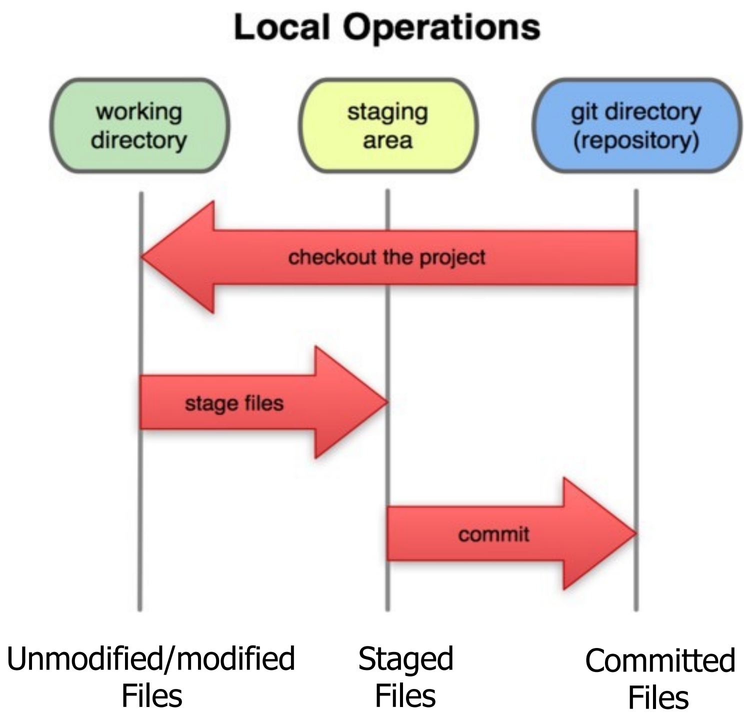
Git keeps "snapshots" of the entire state of the project.

- Each check-in version of the overall code has a copy of each file in it.
- Some files change on a given check-in, 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.

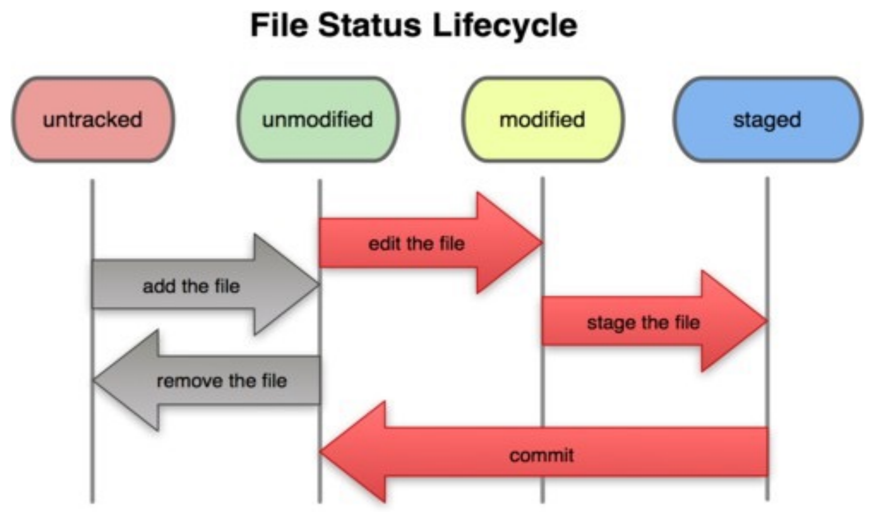


Basic Git workflow

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 hex digits) for every commit, commit hash

  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
```

Git Repository

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 editor by default)

The Repository

- Your top-level working directory contains everything about your project
  - The working directory probably contains many subdirectories—source code, binaries, documentation, data files, etc.
  - One of these subdirectories, named .git, is your repository
- At any time, you can take a “snapshot” of everything (or selected things) in your project directory, and put it in your repository
  - This “snapshot” is called a commit object
  - The commit object contains (1) a set of files, (2) references to the “parents” of the commit object, and (3) a unique “SHA1” name
  - Commit objects do *not* require huge amounts of memory
- You can work as much as you like in your working directory, but the repository isn’t updated until you commit something

`init` and the `.git` repository

- When you said `git init` in your project directory, or when you cloned an existing project, you create a repository
  - The repository is a subdirectory named `.git` containing various files
  - The dot indicates a "hidden" directory
  - You do *not* work directly with the contents of `.git` ; various git commands do that for you

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
`git clone` ***`url [dir]`***	copy a Git repository so you can add to it
`git add` ***`file`***	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`	

Local Git Repository

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.

Making Commits

- You do your work in your project directory, as usual
- If you create new files and/or folders, they are *not tracked* by Git unless you ask it to do so

  ```
  git add newFile1  newFolder1  newFolder2  newFile2
  ```

- Committing makes a “snapshot” of everything being tracked into your repository

  A message telling what you have done is required

  ```
  git commit -m "Uncrevulated the conundrum bar"
  git commit
  ```

  - This version opens an editor for you the enter the message
  - To finish, save and quit the editor
- Format of the commit message

  One line containing the complete summary

  If more than one line, the second line must be blank

Commits and graphs

- A commit is when you tell git that a change (or addition) you have made is ready to be included in the project
- When you commit your change to git, it creates a commit object
  - A commit object represents the complete state of the project, including all the files in the project
  - The *very first* commit object has no “parents”
  - Usually, you take some commit object, make some changes, and create a new commit object; the original commit object is the parent of the new commit object
    - Hence, most commit objects have a single parent
  - You can also merge two commit objects to form a new one
    - The new commit object has two parents
- Hence, commit objects forms a directed acyclic graph (DAG)
  - Git is all about using and manipulating this graph

Commit messages

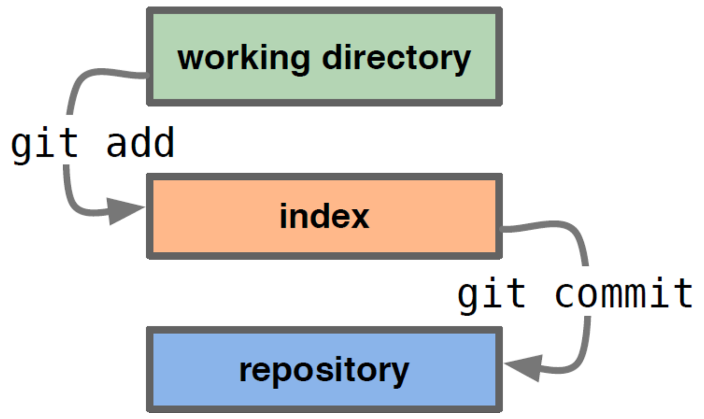
- In git, “commits are cheap.” Do them often.
- When you commit, you must provide a one-line message communicating what you have done (to team members or you future self):
  - Terrible message: “Fixed a bunch of things”
  - Better message: “Corrected the calculation of median scores”
- You can’t say much in one line, so commit often

Typical workflow

- `git status`

  See what Git thinks is going on

  Use this frequently!
- Work on your files
- `git add yourEditedFiles`
- `git commit -m "What did"`



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.

An   example workflow

```
$ git init
$ vi sample.txt
$ git status
$ git status -s
   ?? sample.txt
$ git add sample.txt
$ git status -s
   A  sample.txt

(update sample.txt in editor)
$ git diff

diff --git a/sample.txt b/sample.txt
index 6231f5a..3e32a85 100644
--- a/sample.txt
+++ b/sample.txt
@@ -1,3 +1,3 @@
 12345
 67890
-
+abcde
$ git status -s
M  sample.txt
$ git log
```

Using GitHub with Eclipse

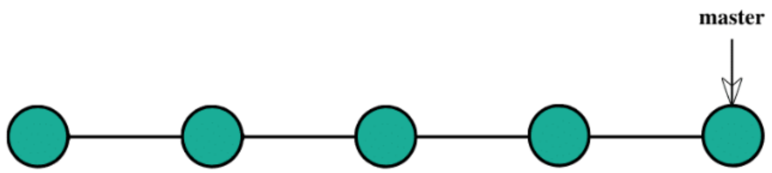
- Open an account on GitHub
- And also create a new repository.
- Also create a token (https://github.com/settings/tokens). You will use this token to access your repository on GitHub.
- On the Eclipse side:
  - Click on your project (in Package Explorer)
  - On the menu choose *Team -> Share Project*

  which creates a *local* git repository for you.
- Then, again from the menu *Team -> Commit*
- Click on the ++ icon to commit staged files (also enter a commit message)
- As the third and final step, “push” committed changes to the repository you created in GitHub account.

Branch in Git

What is a Branch in Git?

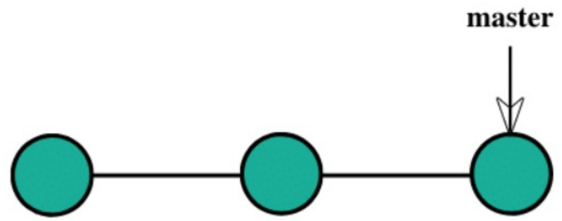
- Branch in Git is a way to keep developing and coding a new feature or modification to the software and still not affecting the main part of the project.



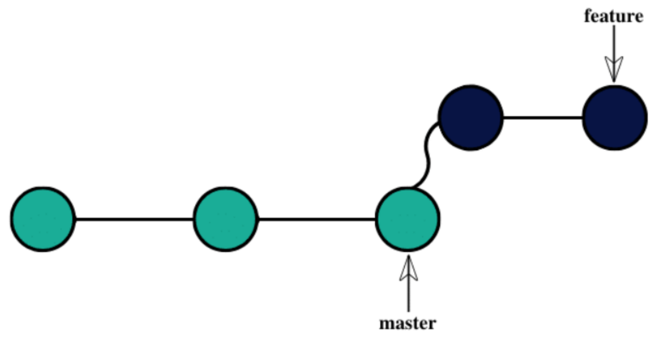
- ***The primary or default branch in Git is the master branch (similar to a trunk of the tree)***. As soon as the repository creates, so does the master branch (*or the default branch*).

Branching in Git

- You have been working on a project with the client being happy until this point.

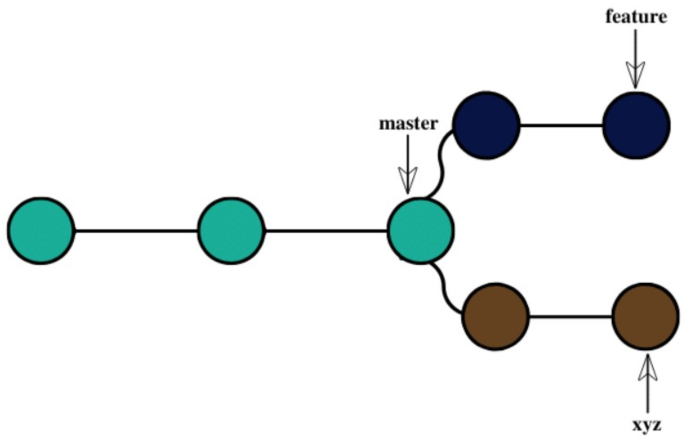


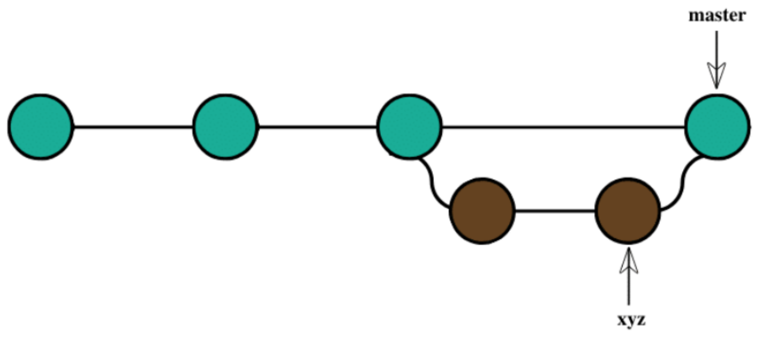
- After that, you decide to develop a feature and create a new branch called feature for the same purpose and start working on it.



Branching in Git

- In the meantime, you decide to develop another feature and wait for the client's approval.
- Now, since you were following the branched strategy, you need to remove the branch, and all the remaining code remains as it is.
- The new feature can be easily added to the master branch to achieve the following.





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`*

Resources

Installing/learning Git

Git website: http://git-scm.com/
- Free on-line book: http://git-scm.com/book
- Git for Computer Scientists:
  - http://eagain.net/articles/git-for-computer-scientists/

At command line: (where verb = config, add, commit, etc.)

```
git help  verb
```