GIT

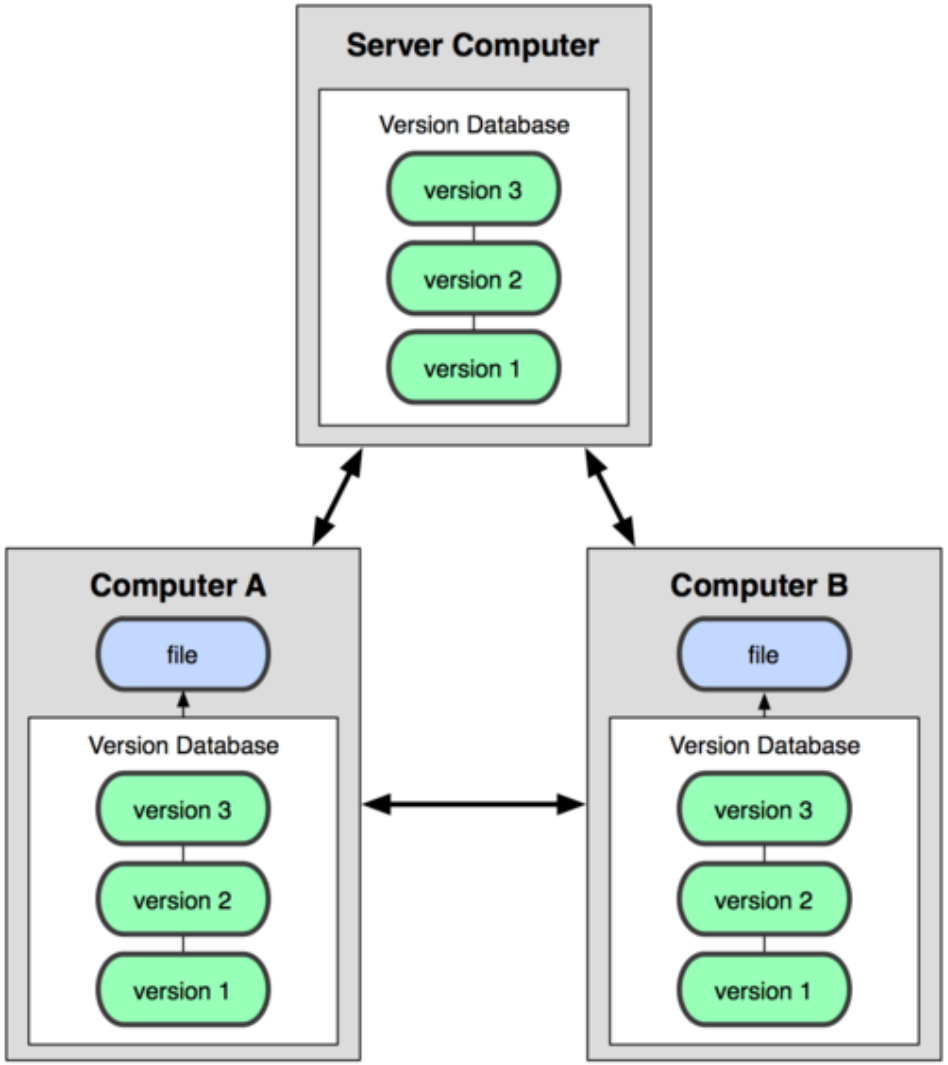
Topic

- Introduction to Git
- How Git works
- Git Commands
- What is GitHub
- summary

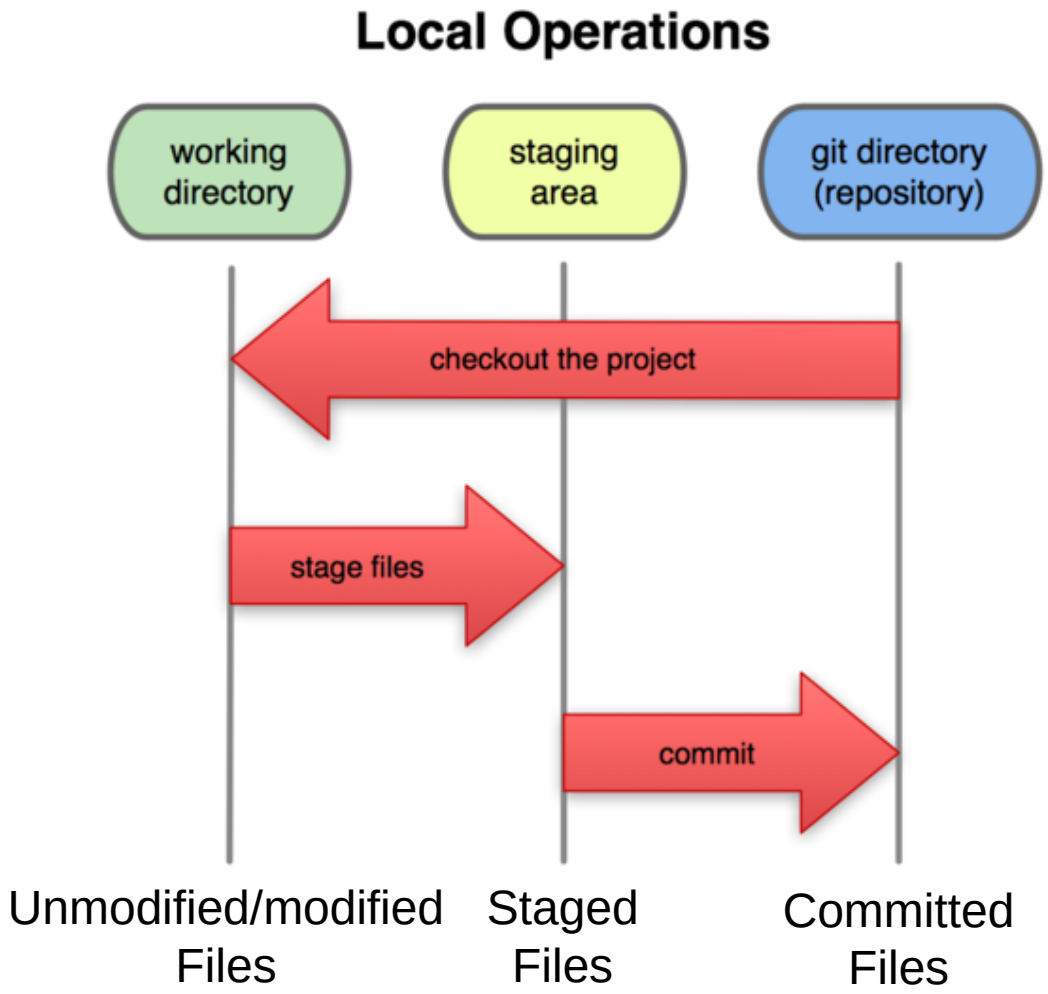
Git History

- 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



A Local Git project has three areas



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

Basic Workflow

Basic Git workflow:

1. **Modify** files in your working directory.
2. **Stage** files, adding snapshots of them to your staging area.
3. Do a **commit**, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

- Notes:
  - If a particular version of a file is in the **git directory**, it's considered **committed**.
  - If it's modified but has been added to the **staging area**, it is **staged**.
  - If it was **changed** since it was checked out but has not been staged, it is **modified**.

What is github?

- GitHub.com is a site for online storage of Git repositories.
- 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

Git Key Concepts

- Repository (Repo)A Git repository is a folder that Git is tracking. It contains your project files and a hidden .git folder with all the version history.
  - Local repository – on your computer.
  - Remote repository – hosted online (like GitHub or GitLab).

- A **commit** is a snapshot of your changes. Think of it as "saving your work" with a message explaining what you did.
  - **git commit -m "Add login form"**

Branch

- A **branch** allows you to work on different features or fixes without affecting the main codebase.
  - main or master – the default branch.You can create other branches like feature/login or bugfix/header.
  - git branch new-feature

Merge

- Merge combines changes from one branch into another (usually into main).
  - git merge new-feature
  - Clone means making a local copy of a remote repository.bashCopyEdit
  - git clone https://github.com/user/repo.git

Push and Pull command

- **Push** uploads your local commits to a remote repository.
  - git push origin main

- **Pull**

- **Pull** downloads changes from the remote repository into your local copy.
  - git pull origin main

Git Staged

- Shows the current state of the working directory and staged files.
    - git status

Get ready to use Git!

1. Set the name and email for Git to use when you commit:

```
$ git config --global user.name “mphasis”
$ git config --global user.email mphasis@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`	

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.txt
  ```

  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
```

After editing a file…

```
[rea@attu1 superstar]$ emacs rea.txt
[rea@attu1 superstar]$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   rea.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
[rea@attu1 superstar]$ git status -s
 M rea.txt                                          ← Note: M is in second column = “working  tree”
[rea@attu1 superstar]$ git diff                     ← Shows modifications that have not been staged.
diff --git a/rea.txt b/rea.txt
index 66b293d..90b65fd 100644
--- a/rea.txt
+++ b/rea.txt
@@ -1,2 +1,4 @@
 Here is rea's file.
+
+One new line added.
[rea@attu1 superstar]$ git diff --cached            ← Shows nothing, no modifications have been staged yet.
[rea@attu1 superstar]$
```

After adding file to staging area…

```
[rea@attu1 superstar]$ git add rea.txt
[rea@attu1 superstar]$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   rea.txt
#
[rea@attu1 superstar]$ git status -s
M  rea.txt                                          ← Note: M is in first column = “staging area”
[rea@attu1 superstar]$ git diff                     ← Note: Shows nothing, no modifications that have not been staged.
[rea@attu1 superstar]$ git diff --cached            ← Note: Shows staged modifications.
diff --git a/rea.txt b/rea.txt
index 66b293d..90b65fd 100644
--- a/rea.txt
+++ b/rea.txt
@@ -1,2 +1,4 @@
 Here is rea's file.
+
+One new line added.
```

Viewing logs

To see a log of all changes in your local repo:

- `$ git log` or
- `$ git log --oneline` (to show a 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.)

Note: changes will be listed by commitID #, (SHA-1 hash)

Note: changes made to the remote repo before the last time you cloned/pulled from it will also be included here

Pulling and Pushing

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)

Note: On attu you will get a Gtk-warning, you can ignore this.

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*!

1. **$ git config --global user.name “Your Name”**
2. **$ git config --global user.email youremail@whatever.com**
3. **$ git clone https://github.com/username/myrepo.git**

Then try:

1. **$ git log, $ git log --oneline**
2. Create a file named *userID*.txt (e.g. rea.txt)
3. **$ git status, $ git status -s**
4. Add the file: **$ git add *userID*.txt**
5. **$ git status, $ git status -s**
6. Commit the file to your local repo:
   $ git commit -m “added rea.txt file”
7. **$ git status, $ git status -s, $ git log --oneline**

***WAIT, DO NOT GO ON TO THE NEXT STEPS UNTIL YOU ARE TOLD TO!!**

1. Pull from remote repo: **$git pull origin master**
2. Push to remote repo: **$git push origin master**

Do This: