Git and GitHub

Idaho State University | Computer Science

Dr. Isaac Griffith

CS 2263
Department of Computer Science
Idaho State University

ROAR

Outcomes

After today's lecture you will be able to:

- Understand the basic git workflow and GitHub
- To use the basic git commands

GitHub

CS 2263

ROAR

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.

For This Class

- You will need to create a GitHub account using your isu.edu email
- Setup your account with an SSH Key
- Use Git and GitHub for your projects and homeworks

SSH

- SSH provides the ability to securely connect to remote services
  - Using this you can connnect to GitHub without a username and personal access token
- To Use SSH you will need to do the following
  1. Generate a new SSH Key
  2. Add the key to the ssh-agent
  3. Add the key to GitHub
  4. Test your SSH Connection

Generating a new SSH Key

1. Open a terminal (or Git Bash for windows)
2. Execute the following command (use your isu.edu email for the email address)

   ```
   $ ssh-keygen -t ed25519 -C "your_email@isu.edu"
   ```

   If you are using an older system that does not support the ed25519 algorithm use the following

   ```
   $ ssh-keygen -t rsa -b 4096 -C "your_email@isu.edu"
   ```

3. When prompted to enter a file to save the key in, just use the default
4. At the prompt type a secure passphrase (this encrypts your private key)

   ```
   > Enter passphrase (empty for no passphrase): [Type a passphrase]
   > Enter same passphrase again: [Type passphrase again]
   ```

Adding your SSH Key to ssh-agent

1. Open a terminal (or Git Bash for windows) if not already open
2. Run ssh-agent in the background using the following command

   ```
   $ eval "$(ssh-agent -s)"
   > Agent pid 59566
   ```

3. Modify the `~/.ssh/config` file to automatically load keys into ssh-agent

   `$ open ~/.ssh/config` (if it exists)

   `$ touch ~/.ssh/config` (if it does not exist, then open it)

   Modify it to contain the following lines

   ```
   Host *
     AddKeysToAgent yes
     UseKeychain yes
     IdentifyFile ~/.ssh/id_ed25519
   ```

4. Add your ssh private key to the ssh-agent

   ```
   $ ssh-add -K ~/.ssh/id_ed25519
   ```

5. Add your public ssh key to your GitHub account

Adding your SSH Key to GitHub

1. From the Git Bash terminal (or terminal in Mac/Linux) copy the SSH public key to the clipboard

   `$ clip < ~/.ssh/id_ed25519.pub` (windows)

   `$ cat ~/.ssh/id_ed25519.pub` (linux/mac)

2. On GitHub (make sure you logged in), click your profile photo, then click **Settings**
3. In the user settings sidebar, click **SSH and GPG keys**
4. Click **New SSH key** or **Add SSH key**
5. In the “Title” field, add a descriptive title
6. Paste your key into the “Key” field
7. Click **Add SSH key**
8. If prompted, confirm your GitHub password

Testing your SSH connection

1. Open the terminal or Git Bash terminal
2. Enter the following:

   ```
   $ ssh -T git@github.com
   ```

   You may see a warning similar to the following:

   ```
   > The authenticity of host 'github.com (IP ADDRESS)' can't be established.
   > RSA key fingerprint is SHA256:nThbg6kXUpJWGl7E1IGOCspRomTxdCARLviKw6E5SY8.
   > Are you sure you want to continue connecting (yes/no)?
   ```

3. Type “yes”

   If it worked you should see the following:

   ```
   > Hi username! You've successfully authenticated, but GitHub does not
   > provide shell access.
   ```

   If it doesn’t work you would see something like the following:

   ```
   ...
   Agent admitted failure to sign using the key.
   debug1: No more authentication methods to try.
   Permission denied (publickey).
   ```

4. Verify that the resulting message contains your username

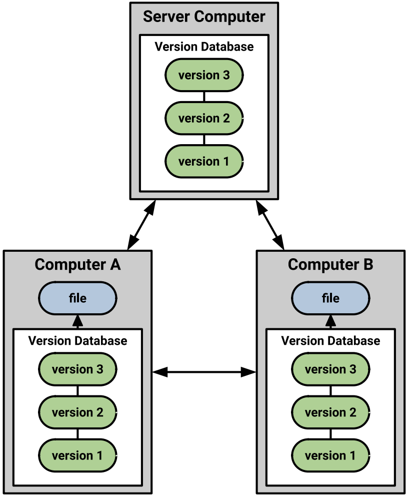
Git

CS 2263

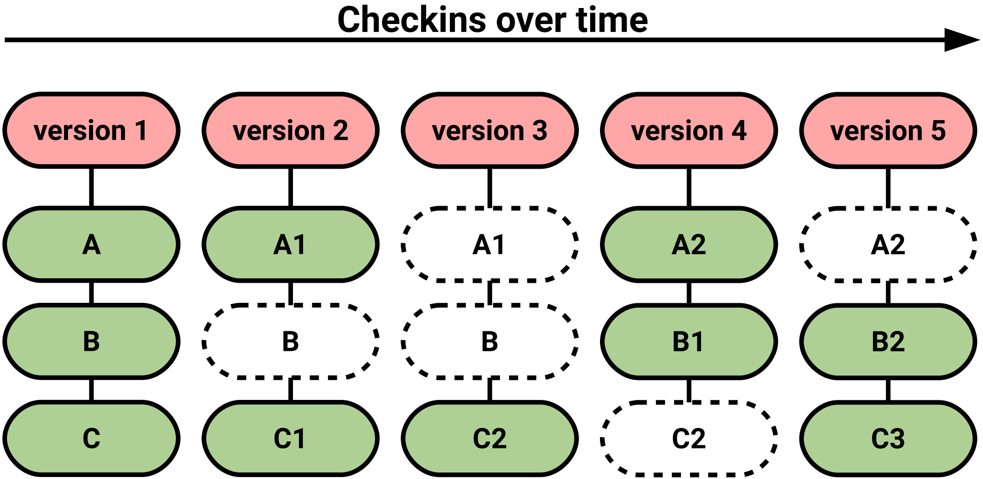
Git History

- 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



Git Takes Snapshots



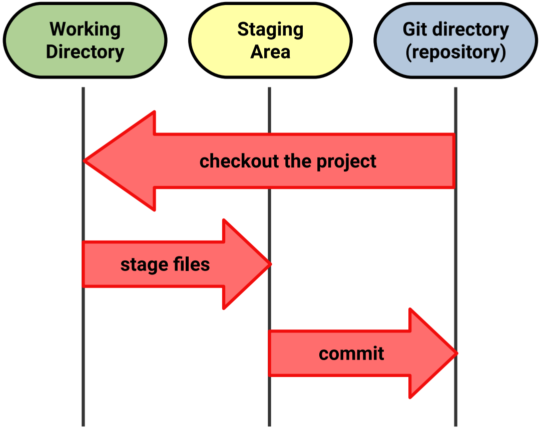
Checkins over time

version 1	version 2	version 3	version 4	version 5
A	A1	A1	A2	A2
B	B	B	B1	B2
C	C1	C2	C2	C3

Git Uses Checksums

- Git generates a unique SHA-1 hash for every commit
  - 40 character string of hex digits
- 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`

Local Projects



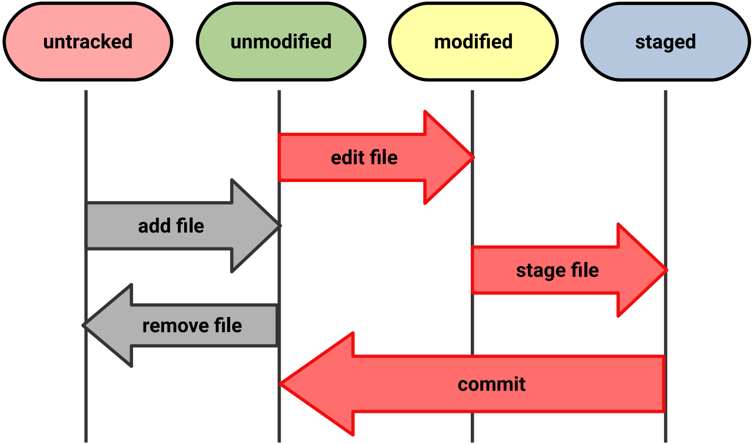
Unmodified/modified Files

Staged Files

Committed Files

Git File Lifecycle

File Status Lifecycle



Basic Workflow

Basic Git workflow:

❶ **Modify** files in your working directory.

❷ **Stage** files, adding snapshots of them to your staging area.

❸ 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**

Using Git

CS 2263

Get Ready to Use Git!

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

```
$ git config --global user.name "Your Name"
$ git config --global user.email youremail@whatever.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

❷ Two common scenarios: (only do one of these)

- To **clone an already existing repo** to your curent 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).
- 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 vesion"
```

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.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)

- To see what is modified but unstaged:

```
$ git diff
```

- To see staged changes:

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

Viewing Logs

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

- `$ git log`
- `$ 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

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*!

Resources

- Pro Git - **Highly recommended reading**. Chapters 1-5 should teach you most of what you need to use Git proficiently.
- Oh Shit, Git!?! - short guide on how to recover from common git mistakes.
- Git for Computer Scientists - short explaination of git's data model
- Git from the Bottom Up - detailed explanation of git's implementation, for the curious
- How to explain git in simple words
- Learn Git Branching - a browser-based game that teaches you git.

For Next Time

- Review the Git Book Ch. 2
- Review this Lecture
- Come to Class
- Continue working on Homework 02
- Read the Git Flow Articles

For Class make sure you have:

- Installed the most recent version of git on your laptop and can run it from the command line
- Created a GitHub account and setup your ssh keys

Are there any questions?

ROAR