Announcements

• Assignment 0 was released on Wednesday (Piazza).
  • Due this coming Tuesday night
  • Solo, written assignment
Outline

• Working with git
• Tarring a directory
• Transferring files
Working with \texttt{git}

\textbf{Last time:}
We cloned the existing course repo \texttt{cs120sp17-public} into the \texttt{cs120} directory, and then copied its contents to \texttt{cs120-personal}.
Working with git

This time:
We will create a repo from scratch and copy its contents between machines.

Repo (why)?
• A copy on a remote git server
• Supports collaboration / sharing
• Version control
  • Stores the history as a collection of concise, semi-automatically annotated, “snapshots” of the repo
Working with **git**

This time:
We will create a repo from scratch and copy its contents between machines.

**Repo (what)?**
- A copy of the entire history of the files/project
  - A *local* copy is stored on your (client) machine.
  - A *remote* copy is stored on the server (e.g. bitbucket.org or github.org)
Working with **git**

This time:
We will create a repo from scratch and copy its contents between machines.

**Repo (how)?**
- The repo directory contains:
  - **Local Repo**: A copy of the entire repository (in the `.git` subdirectory)
  - **Working copy**: A replica of all the “latest” files.
Working with **git**

This time:

We will create a repo from scratch and copy its contents between machines:

- Create an account on bitbucket.org
- Create an empty repository
- Clone the repository to our cs120 folder
- Add / modify local files
- Push the edited files in the repo.
Working with \texttt{git}

Before we start:

• We need to tell the \texttt{git} system (on \texttt{ugradx}) a bit about ourselves:

  • What name do we want the system to know us by:
    \begin{verbatim}
    git config --global user.name "<your name>"
    \end{verbatim}
    (quotes are required if there is a space in \texttt{<your name>})

  • What email do we want the system to know us by:
    \begin{verbatim}
    git config --global user.email <your email>
    \end{verbatim}
Create an account / log in
Create a new repository
Clone the repository
Clone the repository
Working with **git**

One you have a repository, you can:

- **commit** changes you’ve made locally (to create a restore point)
- **push** those back up to the git server (to synchronize your local repo)

**Note:**

- **git** does not know what changes you want to commit.
- You will have to let it know explicitly.
Working with **git**

To run a git command you type “git”, followed by the command, followed by the parameters:

- **git clone** `<address>`:
  - clones a repository into the local directory
- **git pull**:
  - fetches the most recent version of the repo from the server and merges it with the repo (and working copy) on the local machine
- **git status**:
  - display the current status of the repository
- **git log**:
  - display the history of changes (commits)
Working with **git**

To run a git command you type “git”, followed by the command, followed by the parameters:

- **git add <file name>:**
  - add the specified file to the list of files that you will be committing
    (do this to add a file to the repo or if you’ve changed an existing file)

- **git commit –m “<commit message>”:**
  - commit the changes (locally), including a brief message describing the changes.

- **git commit –am “<commit message>”:**
  - like git commit –m “<commit message>” but automatically adds any files that have been modified (not added) since the last commit

- **git push:**
  - push your local repo back to the server
Working with **git**

Note that you cannot modify a repo directly using standard `mv` or `rm`; all interactions are via the `git` command:

- **`git mv <source file> <target file>`:**
  - rename a file

- **`git rm <file>`:**
  - remove a file (delete it)

For a more complete list, see:
Working with **git**

- Files in your working copy (you **git add**'ed them) are *tracked* and can be in one of several states:
  - *Unmodified* (same as copy in local repo)
  - *Modified* (different from copy in local repo but not yet staged)
  - *Staged* (next **git commit** will update the repo)
Working with **git**

- The `git push` / `pull` / `clone` commands synchronize the local repository with the remote one.
- These commands synchronize the local/global repos, so be sure that your working copy matches the local repo before calling them (i.e. all files are unmodified)
Working with **git**

- Files that are *not* yet part of your project are *untracked*
  - When you create a new file; it’s *unstaged* until you `git add` it
    - This is true even if the file is in the directory containing your working copy
    - However, `git` will notice it, and it will appear as unstaged if you `git status`

- Some untracked files are files that we want `git` to ignore because we’ll never want to include them in the remote repo
  - Tell `git` to ignore a file by adding it to .gitignore file
  - Good candidates for ignoring might be a .out, gitlog.txt
Working with **git**

**Workflow Suggestions:**

- Start each session with `git pull`, to ensure your local copy is up-to-date.
- After you complete work on a small task, `git commit` it.
- Include a message with every commit to explain what changes you committed (use `-m`, or you might be forced into vim editor to create one!)
- Make sure you `git commit` and `git push` at the end of each work session.
Working with git

• Don’t be discouraged if git concepts are elusive at first
  • You can get by with just a few key ideas
  • Tutorials and explanations linked from Resources section of Piazza (go to General Resources area, then click on Tools Reference)
  • Lots of help available from TAs/CAs, instructors, Google
Working with **git**

There are also GUIs for **git**, like **SourceTree**.
Tarring a directory

The `tar` command lets us package everything in a directory (and all its sub-directories) into a single file.

**Why?**
- It is easier to move a single file
Tarring a directory

Basic operations:

• Create a “tar-ball”, filtered through gzip, and put it in a file.
  ```
  tar -c -z -f <tar-ball name>.tgz <directory name>
  tar -czf <tar-ball name>.tgz <directory name>
  ```

• Extract the contents of a tar-ball file that was filtered with gzip
  ```
  tar -xz <tar-ball name>.tgz
  ```

• List the contents of a tar-ball file that was filtered with gzip
  ```
  tar -tz <tar-ball name>.tgz
  ```
Transferring files

While we will do our work on the ugradx machines, we will sometimes want to copy our files to other machines.

• Audio
• Video
• Printing
• etc.

Or we may want to move content from our machines to ugradx...

Since the content on our ugradx is ours, we need to do this securely (using authentication to validate that the files are ours).
Transferring files

To perform local/local copying, we use the copy command, cp.
To perform local/remote or remote/local copying we use the secure copy command, [p]scp.

```
[p]scp misha@ugradx.cs.jhu.edu:<source> <target>
```

moves from ugradx to local machine.

```
[p]scp <source> misha@ugradx.cs.jhu.edu:<target>
```

moves to ugradx from local machine.

On windows, you will be using Putty, hence the [p]
Transferring files

Example:
On Linux, if the source file is cs120/foo.c on ugradx and we want to copy it to the current directory on our local machine:

```
scp misha@ugradx.cs.jhu.edu:cs120/foo.c .
```

On Windows, if we want to copy the file to directory foo:

```
pscp misha@ugradx.cs.jhu.edu:cs120/foo.c foo\n```
Transferring files

There are also GUIs for transferring, like WinSCP on Windows machines.
In-Class Exercises

• On Piazza, find Resources section, then click Resources tab
• Scroll down to section for this course section
• Find link for Exercise 1-2 and follow it
• Follow the instructions; raise your hand if you get stuck
• Make sure you check in with a course staff member sometime during this session