EECS 470 Lab 4
Version Control System

Department of Electrical Engineering and Computer Science
College of Engineering
University of Michigan

Friday, 31st January, 2014
Overview

Administrivia

Git

VCS Basics
Distributed VCS
Git Preliminaries
Git Basics by Example

Assignment
Homework

- Homework 2 is due Monday, 3rd February at the beginning of lecture
- If you haven’t, you need to get started now

Projects

- Project 3 is due Sunday, 9th February at 9:00PM

We are available to answer questions on anything here. Office hours can be found in the course google calendar.
Version Control Systems

What is a version control system?

- Stores text files
- Keeps old versions around
- Allows parallel work

Why do I care?

- Prevents/helps prevent loss of data (nothing is foolproof)
- Great for group work
- Required for Project 3
History of VCS

Motivation
Avoid having

- isr.v.old
- isr.v.old2
- isr.v.working
- isr.v.REALLY_working

A Short History

<table>
<thead>
<tr>
<th>Year</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1950</td>
<td>Manual file naming</td>
<td>Local</td>
</tr>
<tr>
<td>1982</td>
<td>Revision Control System</td>
<td>Local</td>
</tr>
<tr>
<td>1986</td>
<td>Concurrent Versions System</td>
<td>Central</td>
</tr>
<tr>
<td>2000</td>
<td>Subversion</td>
<td>Central</td>
</tr>
<tr>
<td>2005</td>
<td>Git</td>
<td>Distributed</td>
</tr>
</tbody>
</table>
Centralized VCS

What does it mean to be centralized?

- Clients talk to a server, which is the one, true version
- Server copy keeps history
- Clients have a(n) (incomplete) *working copy*
Distributed VCS

VCS Structure – Centralized
What does it mean to be distributed?

- Every copy is created equal (can all act as the server)
- No one, true version
Distributed VCS

VCS Structure – Distributed

![Diagram of VCS structure](image-url)
Hybrid VCS

The Hybrid Approach

- Use a DVCS
- Set up a server to be the synchronization point
- Possibly still connect directly to colleagues, but generally not
Subsection 3

Git Preliminaries
Secure Shell

What is ssh?

- Secured remote connection to a server, e.g.
  - Remote shell
  - How git (should) communicate(s) with other machines

What do I need to know?

- Requires that you identify yourself:
  - Key (RSA or DSA)
  - Password (keyboard-interactive)
  - Kerberos (gssapi-with-mic)
- Necessary for Bitbucket (coming up)
Secure Shell: Keys

SSH Keys

- Public-private key pair authentication
- Can also be password protected

`ssh-keygen`

- Creates `~/.ssh/id_rsa` (your private key) and `~/.ssh/id_rsa.pub` (your public key)
Example: `ssh-keygen`

```
[wcunning@dahak ~] $ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/wcunning/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/wcunning/.ssh/id_rsa.
Your public key has been saved in /home/wcunning/.ssh/id_rsa.pub.
The key fingerprint is:
The key's randomart image is:
+--[ RSA 4096]----+
|       . E |
|       . o |
|       . + |
|       B = |
|       S B + ...
|       . + ...
|       o .+ ..
|       .o.o +|
|       o=+
+-------------------+
```
Git Configuration

Follow Along

- Run the following commands:
  - `git config --global user.name “Your Name”`
  - `git config --global user.email “uniqname@umich.edu”`

- Optionally:
  - Download `wcunning-dotfiles.tar.gz`
  - Open it up
  - Copy `.gitconfig` and `.gitignore` to your home directory
  - Optionally copy other configs/google those programs
Bitbucket

What is Bitbucket?
- Free, online git repositories
- More friendly to private repos than Github

What do I need to know?
- Used for Project 3 and the final project
- You need to make an account, right now
- Once you have an account, add it to this spreadsheet

How to make a Bitbucket account...
Example Execution

```
[wcunning@dahak ~] $ git clone git@bitbucket.org:eecs470staff/course_material.git
Cloning into 'course_material'...
Warning: Permanently added the RSA host key for IP address '131.103.20.168' to the list of known hosts.
remote: Counting objects: 183, done.
remote: Compressing objects: 100% (166/166), done.
remote: Total 183 (delta 39), reused 0 (delta 0)
Receiving objects: 100% (183/183), 141.58 KiB | 0 bytes/s, done.
Resolving deltas: 100% (39/39), done.
Checking connectivity... done
[wcunning@dahak Documents] $ cd course_material/
[wcunning@dahak project1] [master] $ vim projects/project1/testD.v
[wcunning@dahak project1] [master *] $ git add projects/project1/testD.v
[wcunning@dahak project1] [master +] $ git commit
[master 4caae96] Updated Project 1 to use @(posedge clock);
  1 file changed, 11 insertions(+), 11 deletions(-)
[wcunning@dahak project1] [master] $ git push
Counting objects: 10, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 544 bytes | 0 bytes/s, done.
Total 5 (delta 3), reused 0 (delta 0)
To git@bitbucket.org:eecs470staff/course_material.git
  da0cf7a..4caae96  master -> master
```
Git Basics Introduction

Structure
For each command you need to know, we will

1. Occasionally, an aside with a Git Concept we need to describe a command
2. Describe the command, along with any useful flags/options
3. Show how the command affects the repositories
4. Show an example with output
Git Commands: clone

What does `git clone` do?

- Makes a copy of a repository

Syntax and Options

- Example: `git clone <protocol>:/<repo> <directory>`
  Clones the repo at `<repo>` into `<directory>`
- Repos can be accessed through several different protocols
  - Git: Unsecured, do not use
  - HTTP(S): Used for freely available things, but push should use the secured version
  - SSH: The right answer, always secure, likely passwordless
Git Diagram

Git Repo Structure

[origin]
HEAD=9a12490

Mark
HEAD=9a12490

Jon
HEAD=9a12490

Will
HEAD=9a12490

Commit Tree

9a12490
Git Commands by Example: clone

[jbbeau@dahak ~] $ git clone git@bitbucket.org:eecs470staff/course_material.git
Cloning into '...'
remote: Counting objects: 79, done.
remote: Compressing objects: 100% (68/68), done.
remote: Total 79 (delta 13), reused 0 (delta 0)
Receiving objects: 100% (79/79), 79.99 KiB, done.
Resolving deltas: 100% (13/13), done.
Git Concepts: Working Copy vs. Local Repo

What is the *working copy*?
- The files/folders you actually operate on
  - e.g. group1/

What is the *local repo*?
- The hidden folder containing the stored history for the repo
  - e.g. group1/.git/

Consequences
- .git/ directory is a full repo
  - Useful in the event that you needed a backup
  - Commits get stored here
  - Synchronization with the *remote* is explicit
Git Concepts: Remotes

What is a remote?

- Remotes are bookmarked repositories to synchronize with

Consequences

- Clone automatically creates a remote named origin, which is the default for all remote operations
Git Concepts: Tracked Files

What does it mean for a file to be *tracked*?

- Files that are tracked are stored in the repository

Consequences

- Untracked files have no history
- Tracked files can be compared for changes
- Files can be ignored (.gitignore)
Git Commands: status

What does `git status` do?

- Shows the contents of the staging area
- Shows the files with differences to the most recent commit
- Shows the files untracked by git

Syntax and Options

- Example: `git status`
Git Commands by Example: status

[jbbeau@dahak slides] [master *] $ 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:  eecs470lab4slides.tex
Git Commands: add

What does `git add` do?

- If a file is untracked it becomes tracked
- Otherwise, the current version is taken as a `snapshot` and added to the `staging area`

Syntax and Options

- Example: `git add <files>`
- `<files>` can be any normal shell file structure (globs, wildcards, directories, etc.)
- if `<files>` has a directory, it is added recursively
- `git add -interactive`:
  
  Lets you add files and parts of files from the status output interactively; possibly good for commit building after a complex set of changes
Git Diagram

Git Repo Structure

[origin]
HEAD=9a12490

Mark
HEAD=9a12490

Jon+
HEAD=9a12490

Will
HEAD=9a12490

Commit Tree

9a12490
Git Commands by Example: add

```
[jbbeau@dahak slides] [master *] $ git add eecs470lab4slides.tex
[jbbeau@dahak slides] [master +] $ git status
# On branch master
# Changes to be committed:
#  (use git reset HEAD <file>..." to unstage
#
#       modified:       eecs470lab4slides.tex
#
```
Git Concepts: Snapshots

What is a *snapshot*?

- `git add` doesn’t follow a file in perpetuity
- Instead, it saves a snapshot (the state of the file at the time the command was called)

Consequences

- Files need to be added every time they are changed
- Can add only certain files to a commit
What is the *staging area*?

- Contains the snapshots to be used in creating the next commit

**Consequences**

- Commits can be built thematically
- Commits can be relatively small, even if changes to be committed are large (useful for `git bisect`)
Git Commands: commit

What does `git commit` do?

- Combines everything in the staging area into a commit, described by your log message
- Adds it all to the local repo
- Points at it with a *commit hash*
- Changes HEAD to point at that commit hash

Syntax and Options

- Example: `git commit`
- Interesting things are possible with *commit hooks*
Git Ettiquette: Commits

What is a good commit?

- Informative message – short one liner followed by thorough paragraphs describing all of the changes
- Thematic – all of the changes should go together logically (e.g. added a module and integrated it)
- Small – keeping your changes small makes it easier to find what broke everything by binary searching the commit history (bisect)

Why?

- All of these rules make it easier to find bugs and understand the progression of a project
- Most are mandatory at companies
Git Diagram

Git Repo Structure

[origin]
HEAD=9a12490

Mark
HEAD=9a12490

Jon
HEAD=f2ea0f5

Will
HEAD=9a12490

Commit Tree

f2ea0f5

9a12490
Git Commands by Example: commit

[jbbeau@dahak slides] [master +] $ git commit
[master f2ea0f5] Lab 4 diagram change
1 file changed, 1 insertion(+), 1 deletion(-)
Git Commands: push

What does `git push` do?

- Puts all of your local commits on some remote

Syntax and Options

- Example: `git push`
- Defaults to pushing the master branch to the origin repo
- Interesting things are possible with `push hooks`
Git Diagram

Git Repo Structure

[origin]
HEAD=f2ea0f5

Mark
HEAD=9a12490

Jon
HEAD=f2ea0f5

Will
HEAD=9a12490

Commit Tree

f2ea0f5

9a12490
Git Commands by Example: push

[jbbeau@dahak slides] [master] $ git push
Counting objects: 11, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 475 bytes | 0 bytes/s, done.
Total 6 (delta 3), reused 0 (delta 0)
To git@bitbucket.org:eecs470staff/course_material.git
  9a12490..f2ea0f5  master -> master
Git Commands: pull

What does git pull do?

- Gets all commits from some remote
- This is actually a combination of git fetch followed by git merge

Syntax and Options

- Example: git pull
- Defaults to pulling the master branch from the origin repo
Git Basics by Example

Git Diagram

Git Repo Structure

- [origin]
  - HEAD=f2ea0f5

- Mark
  - HEAD=f2ea0f5

- Jon
  - HEAD=f2ea0f5

- Will
  - HEAD=f2ea0f5

Commit Tree

- f2ea0f5
- 9a12490
Git Commands by Example: pull

```
[wcunning@mycroft-holmes course_material] [master] $ git pull
remote: Counting objects: 13, done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 8 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (8/8), done.
From bitbucket.org:eecs470staff/course_material
  9a12490..f2ea0f5  master    -> origin/master
```
Git Commands: reset

What does `git reset` do?

- Replaces modified files in the working directory with the versions in the local repo (at some commit hash, generally)

Syntax and Options

- Example: `git reset <hash> <file>`
- Replaces the contents of `<file>` with the version in `<hash>`
- `<hash>` can be specified in relation the HEAD (e.g. HEAD~3)
Git Diagram

Git Repo Structure

- **[origin]**
  - HEAD=f2ea0f5

  - Mark
    - HEAD=f2ea0f5

  - Jon
    - HEAD=f2ea0f5

  - Will
    - HEAD=f2ea0f5

Commit Tree

- 9a12490
- f2ea0f5
Git Commands by Example: reset

```
[wcunning@mycroft-holmes slides] [master *] $ git status
# On branch master
# Changes not staged for commit:
#   (use "git add/rm <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   deleted:   beamerthemeLab.sty
#
no changes added to commit (use "git add" and/or "git commit -a")
[wcunning@mycroft-holmes slides] [master *] $ git reset --hard
HEAD is now at f2ea0f5 Lab 4 diagram change
```
Git Commands: branch

What does git branch do?

- Creates a pointer to a particular commit hash, future commits update this pointer

Syntax and Options

- Example: git branch <branchname>
  - Creates a branch named <branchname> starting at the current HEAD
Git Commands: checkout

What does `git checkout` do?

- Moves the working directory to the specified commit hash or pointer to a commit hash

Syntax and Options

- Example: `git checkout <hash>`
- `<hash>` can be
  - an older commit (e.g. `HEAD~2`)
  - a branch, or other pointer, (e.g. `lab4`)
**Git Diagram**

**Git Repo Structure**

- **[origin]**
  - HEAD=f2ea0f5

  - Mark
    - HEAD=f2ea0f5
  - Jon
    - HEAD=f2ea0f5
  - Will
    - HEAD=f2ea0f5

**Commit Tree**

- 9a12490
- f2ea0f5

---

(University of Michigan)  
Lab 4: VCS  
Friday, 31st January, 2014  
52 / 62
Git Commands by Example: branch, checkout

```
[wcunning@mycroft-holmes slides] [master] $ git branch lab4
[wcunning@mycroft-holmes slides] [master] $ git status
On branch master
Your branch is up-to-date with 'origin/master'.
[wcunning@mycroft-holmes slides] [master] $ git checkout lab4
Switched to branch 'lab4'
```
Git Diagram

Git Repo Structure

[origin]
HEAD=f2ea0f5

Mark
HEAD=f2ea0f5

Jon
HEAD=f2ea0f5

Will*
HEAD=f2ea0f5

Commit Tree

9a12490

f2ea0f5
Git Diagram

Git Repo Structure

- [origin]
  - HEAD=f2ea0f5
- Mark
  - HEAD=f2ea0f5
- Jon
  - HEAD=f2ea0f5
- Will
  - HEAD=8e4bab6

Commit Tree

- 8e4bab6
- f2ea0f5
- 9a12490
Git Diagram

Git Repo Structure

[origin]
HEAD=f2ea0f5

Mark*
HEAD=f2ea0f5

Jon
HEAD=f2ea0f5

Will
HEAD=8e4bab6

Commit Tree

8e4bab6

f2ea0f5

9a12490

(University of Michigan)
Git Diagram

Git Repo Structure

[origin]
HEAD=f2ea0f5

Mark
HEAD=b73416d

Jon
HEAD=f2ea0f5

Will
HEAD=8e4bab6

Commit Tree

8e4bab6
b73416d
f2ea0f5
9a12490
Git Basics by Example

Git Diagram

Git Repo Structure

[origin]
HEAD=f2ea0f5

Mark*
HEAD=b73416d

Jon
HEAD=f2ea0f5

Will
HEAD=8e4bab6

Commit Tree

8e4bab6
b73416d
f2ea0f5
9a12490

(University of Michigan)
Lab 4: VCS
Friday, 31st January, 2014
54 / 62
Git Diagram

Git Repo Structure

[origin]
HEAD=f2ea0f5

Mark+
HEAD=b73416d

Jon
HEAD=f2ea0f5

Will
HEAD=8e4bab6

Commit Tree

8e4bab6
9a12490
b73416d
f2ea0f5

(Unciversity of Michigan)
Git Diagram

**Git Repo Structure**

- [origin]
  - HEAD=4b3d3cb
- Mark
  - HEAD=4b3d3cb
- Jon
  - HEAD=f2ea0f5
- Will
  - HEAD=8e4bab6

**Commit Tree**

- 4b3d3cb
- 8e4bab6
  - b73416d
    - f2ea0f5
      - 9a12490
Git Basics by Example

Git Diagram

Git Repo Structure

- [origin]
  - HEAD=4b3d3cb
- Mark
  - HEAD=4b3d3cb
- Jon
  - HEAD=f2ea0f5
- Will
  - HEAD=06505f6

Commit Tree

- 4b3d3cb
- b73416d
- 8e4bab6
- 06505f6
- 9a12490
- f2ea0f5
Git Repo Structure

[origin]
HEAD=4b3d3cb

Mark
HEAD=4b3d3cb

Jon
HEAD=f2ea0f5

Will
HEAD=06505f6

Commit Tree

06505f6
4b3d3cb

8e4bab6
b73416d
f2ea0f5
9a12490
Git Commands: merge

What does `git merge` do?

- Combines two branches (when used manually)
- Can mangle a repository
- Can have *conflicts*

Syntax and Options

- Example: `git merge lab4`
- Merges the `lab4` branch into the current branch
- `git mergetool` helps you handle merge conflicts
- See me in office hours if you need to do this...
Git Diagram

Git Repo Structure

[origin]
HEAD=4b3d3cb

Mark
HEAD=4b3d3cb

Jon
HEAD=f2ea0f5

Will
HEAD=a292319

Commit Tree

a292319

06505f6

4b3d3cb

8e4bab6

b73416d

f2ea0f5

9a12490
[wcunning@mycroft-holmes slides] [master $]$ git merge lab4
Updating f2ea0f5..25cbbce
Fast-forward
.gitignore | 68 +++++++
labs/lab4/assignment/tex/eecs470lab4assignment.tex | 2 +- 
labs/lab4/slides/eecs470lab4slides.tex | 613 ++++++++++++++++++++ 
labs/lab4/slides/script.txt | 148 +++++++++++++++++++ 
labs/lab4/slides/ssh-keygen.txt | 21 ++
5 files changed, 841 insertions(+), 11 deletions(-)
create mode 100644 .gitignore
create mode 100644 labs/lab4/slides/script.txt
create mode 100644 labs/lab4/slides/ssh-keygen.txt
Git Commands: tag

What does git tag do?

- Creates an additional pointer to a particular commit hash

Syntax and Options

- Example: `git tag -a <tagname> <hash>`
- Creates a pointer to `<hash>` called `<tagname>`
- This pointer can be checked out just like a branch
- Tags are local by default, push with `git push <tagname>`
Git Repository Structure

[origin]
HEAD=4b3d3cb

- Mark
  HEAD=4b3d3cb
- Jon
  HEAD=f2ea0f5
- Will
  HEAD=a292319

Commit Tree

lab4 ➔ a292319

- 06505f6 ➔ 4b3d3cb
- 8e4bab6 ➔ b73416d
  - f2ea0f5
  - 9a12490
Git Commands by Example: merge

```
[wconomic@mycroft-holmes w14] [master] $ git tag -a lab4-release HEAD
[wconomic@mycroft-holmes w14] [master] $ git push origin lab4-release
Counting objects: 1, done.
Writing objects: 100% (1/1), 187 bytes | 0 bytes/s, done.
Total 1 (delta 0), reused 0 (delta 0)
To git@bitbucket.org:eecs470staff/course_material.git
  * [new tag] lab4-release -> lab4-release
[wconomic@mycroft-holmes w14] [master] $ git tag -l
lab4-release
```
Lab Assignment

- Assignment is posted to the course website as Lab 4 Assignment.
- If you get stuck...
  - Ask a neighbor, quietly
  - Put yourself in the help queue
- Note: Unlike most labs, to get checked off for this one, you will also need to have made your Bitbucket account and entered it in the spreadsheet.
- When you finish the assignment, sign up in the help queue and mark that you would like to be checked off.
- If you are unable to finish today, the assignment needs to be checked off by a GSI in office hours before the next lab session.
Bitbucket Account Setup

1. Go to Bitbucket Sign Up
2. Fill out the form.
   - Preferably, use your uniqname as your username
   - Use your @umich.edu email address
   - Choose “Personal Account”
3. Click on the little silhouette of a person in the upper right corner.
4. Select “Manage Account” from the list.
5. Click on “SSH Keys” in the list on the left.
6. Click on “Add Key.”
7. In the pop up, paste the contents of ~/.ssh/id_rsa.pub, and name the key whatever you would like.
8. Add your Bitbucket username to the spreadsheet.