Instructor: Nicolas Savva

March 2nd, 2015

---

1 based on slides by Hussam Abu-Libdeh, Bruno Abrahao and David Slater over the years (with additional material from software-carpentry)
Announcements

- The final project is coming out later today
- Up to 2 slip days for the final project
cal - a calendar (Assignment 3)

ascii calendar

**cal [options] [[month] year]**

- **A 2**: display two months after the current month
- **B 5**: display five months after the current month
- **N**: toggle layout
- **w**: show week number
- **j**: Julian days (from start of year)

**Example:** cal -A 2 -B 1 -Nw May 2015

**calendar**

Reminder service using a calendar file.

**man calendar for more info**
GNU Make (recap)

GNU Make

- manage compilation of programs written in languages like C
- used to automatically update any set of files that depend on another set of files
- Makefile describes how files depend on each other, and how to update out-of-date files
- makes use of patterns, rules, and variables to eliminate redundancy
- uses macros to control operation
Version control is the lab notebook of the digital world: it’s what professionals use to keep track of what they’ve done and to collaborate with other people. Every large software development project relies on it, and most programmers use it for their small jobs as well. And it isn’t just for software: books, papers, small data sets, and anything that changes over time or needs to be shared can and should be stored in a version control system.
Why version control?

- Keep revision history
- Easy way to share files between machines
- More effective collaboration
- (Remote) backup
Version Control Systems (VCS)

- hard (not impossible) to accidentally overlook or overwrite someone’s changes to files (highlighted by VCS)
- no disputes about which copy is the most up to date. It is the master copy
- nothing that is committed to version control is lost (means we have an ”infinite undo” for the committed states of the files)
- version control system never overwrites the master copy in the repository
- every time someone commits a new version it is saved as an addition on top of the previous one
- possible to go back in time to see exactly who wrote what on a particular day
Version Control Systems: flavors

- Distributed
  - Git
  - Mercurial (Hg)
  - Bazaar
  - ...

- client-server
  - Subversion (SVN)
  - CVS
  - ...

Version Control Systems: Subversion

- Subversion `svn` : xclient-server VCS model

**svn**

- `svn checkout https://myrepo.org/myproj ~/placetoputworkcp/` makes a local working copy
- `svn log` : display history of the project (comments, changes and details)
- `svn add mynewfile` : add a new file to svn version control
- `svn commit -m "comment goes here"` : commit changes to repository
- `svn update` : retrieve latest changes from remote repository
- `svn diff -r HEAD` : compare working copy against HEAD revision from repository
- `svn revert myfile.txt` : undo uncommitted changes (roll back to HEAD)
- `svn merge -r HEAD:num myfile.txt` : where num is the revision we want to roll back to
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>svn co</code></td>
<td>Check out repository</td>
</tr>
<tr>
<td><code>svn log</code></td>
<td>View change history</td>
</tr>
<tr>
<td><code>svn add</code></td>
<td>Add file</td>
</tr>
<tr>
<td><code>svn commit</code></td>
<td>Commit file</td>
</tr>
<tr>
<td><code>svn update</code></td>
<td>Update to most recent revision</td>
</tr>
<tr>
<td><code>svn diff</code></td>
<td>View differences between revisions</td>
</tr>
<tr>
<td><code>svn merge</code></td>
<td>Merge revisions</td>
</tr>
</tbody>
</table>
Version Control Systems: Git

- **git**: distributed VCS model

```bash
# configure username
$ git config --global user.name "Your Name"
# configure email
$ git config --global user.email "you@some.domain"
# initialize working dir as repo
$ git init
# clone a remote repository
$ git clone http://mygiturl.edu
```
Version Control Systems: More Git

- **git**: distributed VCS model

**git**

- `git status`: show state of working dir
- `git add myfile1 myfile2`: add files to staging area
- `git add -A .`: add all modified files recursively
- `git commit -m "Some message"`: commit changes to local repo
- `git log`: view repo commit history
- `git diff`: show difference between current and last saved state
- `git diff --staged`: difference between last saved and staged things
- `git diff HEAD 42 myfile`: compare file to an earlier version
Version Control Systems: Even More Git

- `git`: distributed VCS model

```bash

- `git reset --hard HEAD`: erase all local changes since last commit
- `git checkout 424242 path/to/file`: restore file to a state from a prior revision
- `git remote add nickname remotespecurl`: add a remote repo to the git repo
- `git remote -v`: list associated remote git repos
- `git push nickname master`: commit local state to master branch of remote repo
- `git push -u nickname master`: push to remote repo if master does not exist
- `git pull nickname master`: get changes from remote repository
```
Repositories Host projects online

- Bitbucket (Mercurial or Git)
- Github (Git)
- Launchpad (Bazaar)
- Google Code (SVN, Mercurial or Git)
- Microsoft CodePlex (SVN, Mercurial or Git)
- Sourceforge (CVS, SVN, Bazaar, Git or Mercurial)
- Cornell Forge (SVN) - http://forge.cornell.edu
Further resources (Git)

- Git tutorial
  https://www.atlassian.com/git/tutorial
- Pro Git book
  http://git-scm.com/documentation
- Git (Interactive) Cheat Sheet
  http://ndpsoftware.com/git-cheatsheet.html
- Another Cheatsheet
Package Management System

- Parts of a Linux distribution and most of the addon software are installed via the Package Management System
- A package contains the files and other instructions to setup a piece of software
- Many packages depend on each other
- High-level managers download packages, figure out dependencies and deal with groups of packages
- Low-level managers unpack individual packages, run scripts, and get the software installed correctly
- Two main families exist: dpkg and rpm (low level)
  - apt-get (Debian), zypper (SUSE), yum (Fedora) at the high level

We briefly touch how to install, remove, or search for packages using the different families of package management tools.
apt aka Advanced Package Tool
(apt-get and apt-cache commands)

Debian:

```
sudo dpkg -l: list installed packages
sudo apt-cache search myproogname: search for a package
sudo apt-get install myproogname: install a package
sudo apt-get upgrade: update installed packages
sudo apt-get remove myproogname: remove a package
sudo apt-cache policy myproogname: check package status
```

Check https://wiki.debian.org/AptCLI for more info
• `yum` aka Yellowdog Updater Modified (rpm compatible)

Debian:

```bash
sudo yum list installed: list installed packages
sudo yum search myprogname: search for a package
sudo yum install myprogname: install a package
sudo yum update: update installed packages
sudo yum remove myprogname: remove a package
sudo yum info myprogname: check package status
```
zypper similar to yum (rpm based)

Debian:

- `sudo rpm -qa`: list installed packages
- `sudo zypper search myprogname`: search for a package
- `sudo zypper install myprogname`: install a package
- `sudo zypper update`: update installed packages
- `sudo zypper remove myprogname`: remove a package
- `sudo zypper info myprogname`: check package status
Check https://guide.macports.org/ and install Xcode and MacPorts

* port command

Also check Homebrew (`brew`): http://brew.sh/

**Darwin:**

```
sudo port installed: list installed packages
sudo port search myprogname: search for a package
sudo port install myprogname: install a package
sudo port upgrade: update installed packages
sudo port uninstall myprogname: remove a package
sudo port info myprogname: check package status
```
Next Time