GENERAL ASSEMBLY

Introduction to git

APRIL 16, 2014

Part 0: The Problem

When editing project files this has a tendency to happen:

```
$ ls
ls -lt
total 0
-rw-r--r--  1 dstevens  staff  0 Apr 16 09:52 file.py
-rw-r--r--  1 dstevens  staff  0 Apr 15 10:00 file.py-1
-rw-r--r--  1 dstevens  staff  0 Apr 14 08:00 file.py-2
-rw-r--r--  1 dstevens  staff  0 Apr 10 08:00 file.py-2014-04-10
-rw-r--r--  1 dstevens  staff  0 Dec 10 08:00 file.py-OLD
-rw-r--r--  1 dstevens  staff  0 Oct 10  2012 file.py-OLD.older
```

The revision control benefits

- stores revisions of a file on demand
- stores comments on each revision
- allows retrieval of any previous revision
- displays the differences between any pair of revisions
- manages multiple lines of development

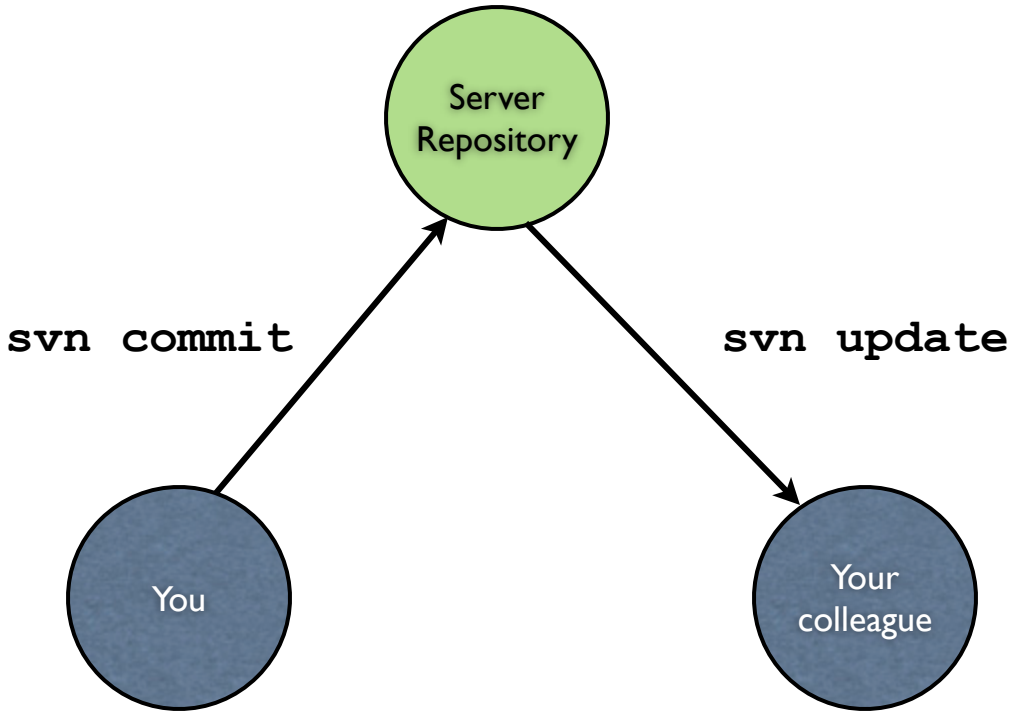
Part 1: Basic Theory

git is really intelligent, and beautiful.

But it takes some time to fully grasp the way it works.

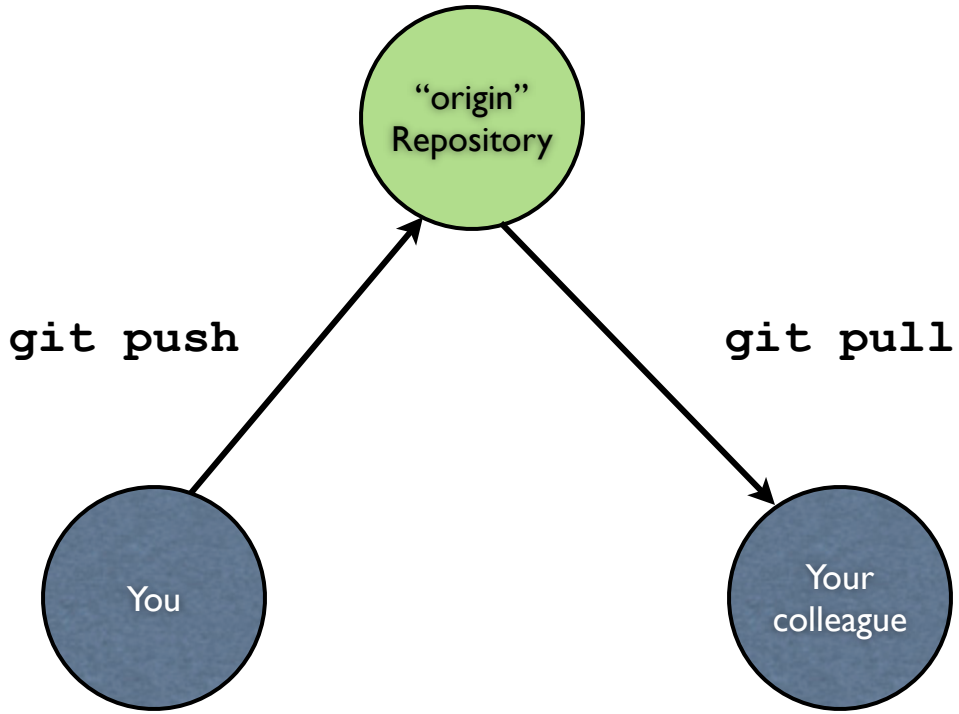
Just ask if you need help with your understanding.

The SVN model



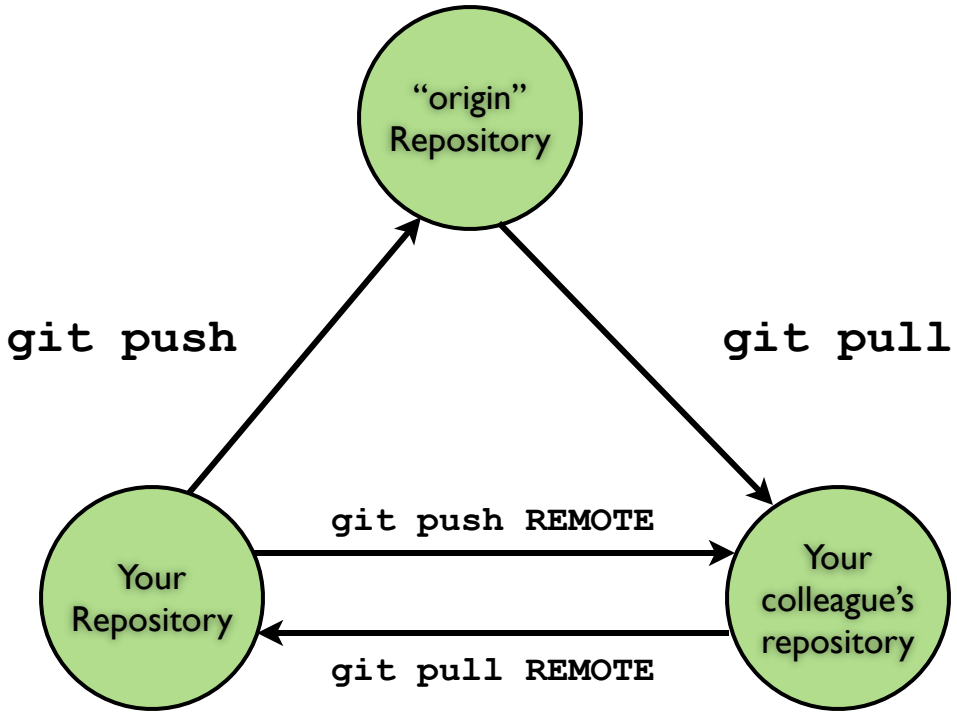
Commits are only shared through the server.

The Git model



May at first look like a traditional Revision Control System

The Git model



You can push/pull commits to any remote repository, there is no difference between server and client.

So how does that work?

First of all, definitions:

- **Working tree**
  A directory in your filesystem that is associated with a repository, containing files & sub-directories.
- **Repository**
  A collection of commits & branches, saved in the `.git` directory.
- **Commit**
  A snapshot of your working tree at a certain point in time, identified by a revision number.
- **HEAD**
  The name for the commit thats currently checked out in the working tree.

Revision Numbering

- Every commit gets a globally unique identifier, not a simple revision number

```
git log 00de993ae4a12f286da8bdf24b041c2e8dfc4e3b
```

vs

```
svn log -r3129
```

- For commands you can also cut off the end of the identifier, as long as that is still unique within your repository

```
git log 00de993ae4
```

`git clone`

is your starting point for working with existing code

It creates a local repository for you, copying & tracking the master branch from the specified location.

```
git clone https://github.com/numpy/numpy.git
```

Remotes

- By default you'll only have the "origin" remote repository, which is the repository you did `git clone` from.
- List existing remotes using `git remote`
- Show details with
  `git remote show NAME`
- Add new remotes using
  `git remote add NAME URL`

Commits = local

- `git commit` only affects your repository, not the origin or any other remote repository
- `git push` in order to share your commits
- Commits are cheap & fast
- Commit as often as possible!

The index

- When you edit/add/remove files, only your working tree changes
- To commit changes, you first save them in the index with `git add` (or `git rm`)
- `git status` shows the current index
- `git commit` commits only the changes saved in the index, and clears the index afterwards

But there's also `git commit -a` to commit all unsaved working tree changes without adding them to the index.

But use this option carefully.

* You still have to use `git rm` for removing files

Also of import:

`git push`

push our changes to a
remote repository

Additionally:

```
git log
git diff
git annotate
```

And they do what you expect.

```
man git-annotate
...
```

is also your friend.

Branching
&
Merging

Branches are cheap.

Merging just works.*

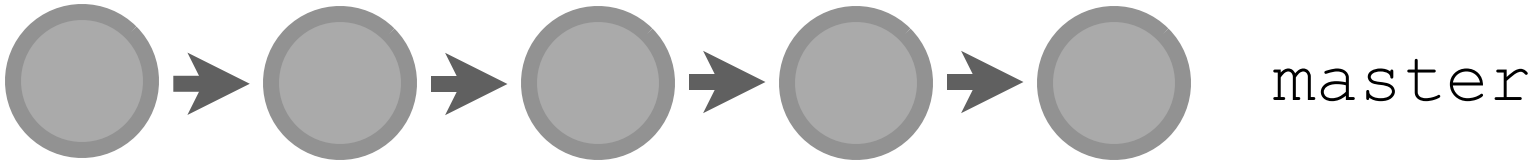
* ...or “merging works very well”; merge conflicts can still be very complex to remedy

Branch

=

Alternate commit path

Per default you always work on the `master` branch.



`git branch`

shows you the branch you are currently on
(marked with *) & lists the available branches

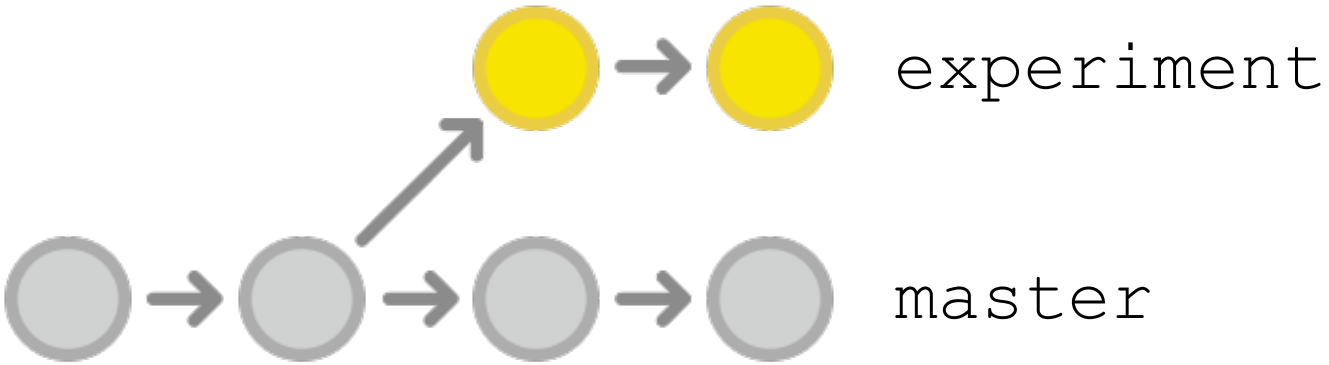
A branch is local to your repository,
create as many or as little as you like.

Create a new one:

```
git branch BRANCH
```

Checkout an existing one:

```
git checkout BRANCH
```



You can also push all commits of your current branch to a remote branch:

```
git push REMOTE BRANCH

git push origin master
    git push
```

Or checkout someone else's branch, work on it, and then share your changes:

```
git branch -r # list remotes
git checkout REMOTE_BRANCH
git branch BRANCH
...
git push
```

`git merge BRANCH`

Merges the specified branch into your current branch.

You must have a common ancestor.

If the merge fails, use `git status` to see the conflicts, edit the files, `git add` them and then `git commit`.

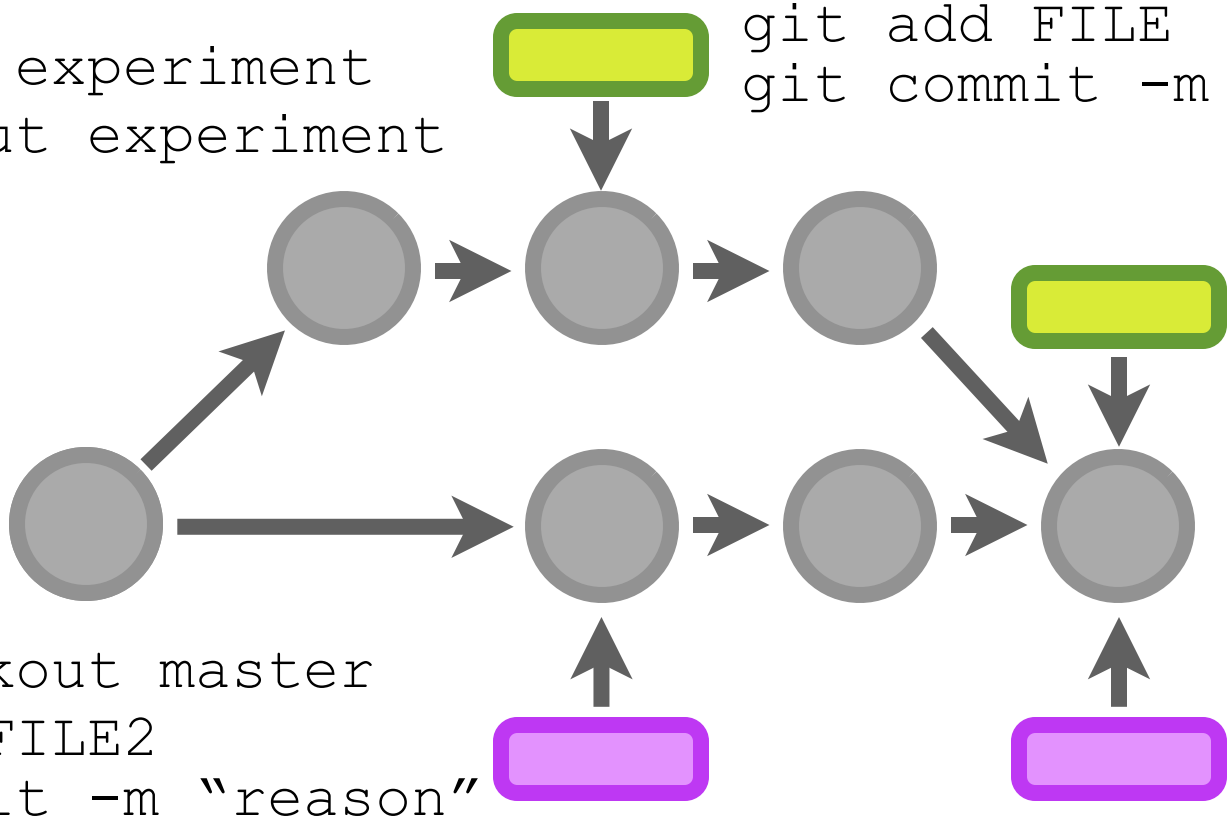
Visualizing the tree

1.

```
git branch experiment
git checkout experiment
```

2.

```
git add FILE
git commit -m “comment”
```



3.

```
git checkout master
git add FILE2
git commit -m “reason”
```

4.

```
git merge experiment
```

You can delete the merged branch afterwards:

```
git branch -d BRANCH
```

git will complain if this would result in lost changes (use `-D` to override).

```
git push REMOTE :REMOTE_BRANCH
```

to delete a remote branch.

Careful!

This won't check whether the branch is already merged.

Tips & Tricks

`git gui`

is your friend, especially when you're confused.

`git checkout FILENAME`

to revert a file changed in the working tree back to HEAD

`git checkout BRANCH`

to revert a all files to the HEAD of that BRANCH

Only if you didn't commit yet.

`git revert REVISION`

to revert a complete revision.

Creates a new commit that removes the changes.

`.gitignore`

instead of

`svn propset svn:ignore`

Single `.git` directory in the root directory,
instead of multiple `.svn` directories

Most relevant configuration is stored in the

.git/config

remotes, author name/email, etc.

git stash

git submodule

git cherry-pick

One great resource:

https://www.atlassian.com/git/tutorial

Further information

- Git - SVN Crash Course
  http://git.or.cz/course/svn.html
- The Git Community Book
  http://book.git-scm.com/index.html
- John Wiegley: Git from the bottom up
  http://ftp.newartisans.com/pub/git.from.bottom.up.pdf
- Github: a social source-code sharing site
  http://github.com/

Part 2: Lab