
Version control with Git

<http://git-scm.com/book/en/>

Basic Intro to Git

- We will:
 - Discuss how Git differs from Subversion
 - Discuss the basic Git model
 - Pull/clone files from a repository on github
 - Edit files in your own local Git repo
 - Push files to a repo on github

Git Resources

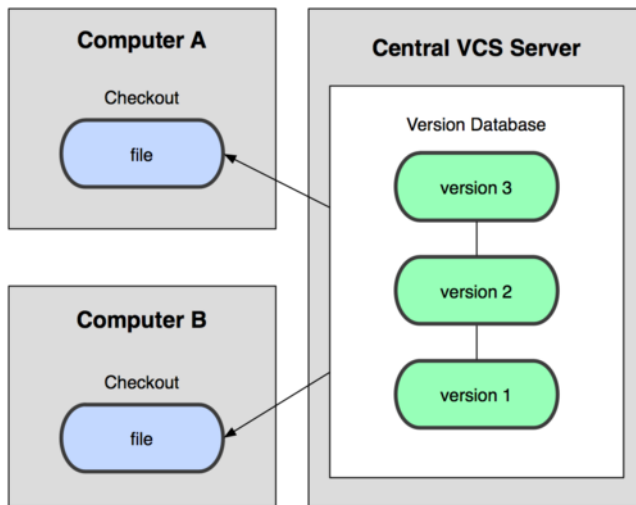
- At the command line: (where verb = config, add, commit, etc.)
\$ git help <verb>
\$ git <verb> --help
\$ man git-<verb>
- Free on-line book: <http://git-scm.com/book>
- Git tutorial: <http://schacon.github.com/git/gittutorial.html>
- Reference page for Git: <http://gitref.org/index.html>
- Git website: <http://git-scm.com/>
- Git for Computer Scientists (<http://eagain.net/articles/git-for-computer-scientists/>)

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

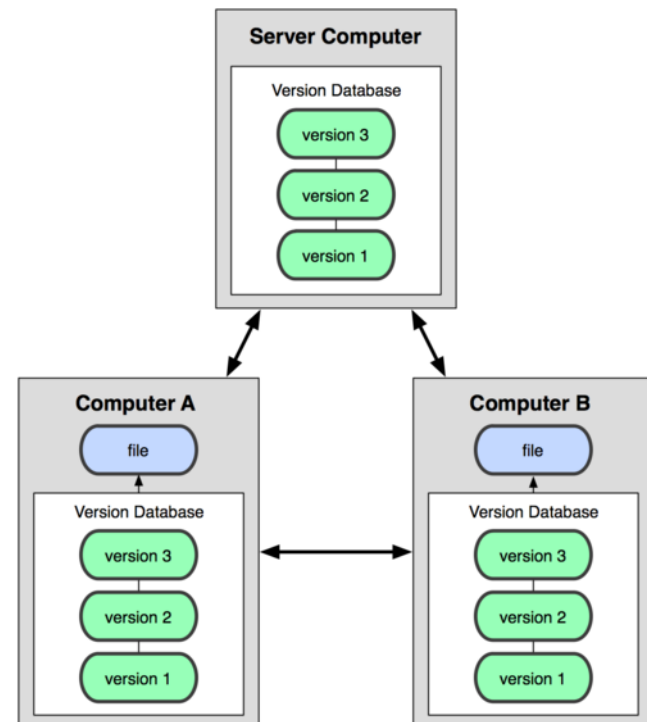
Git uses a distributed model

Centralized Model



(CVS, Subversion, Perforce)

Distributed Model

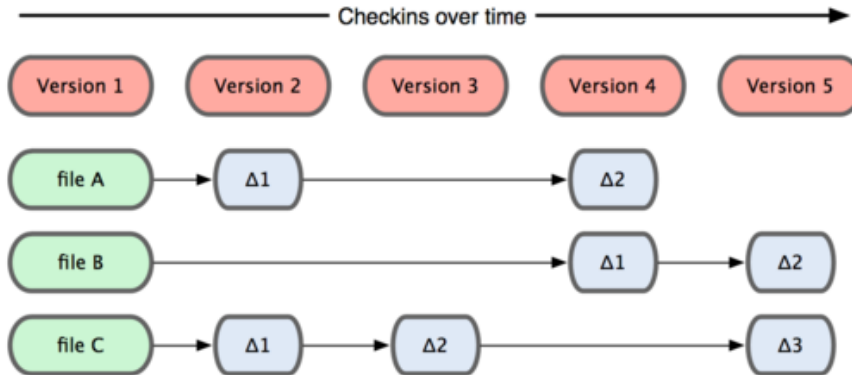


(Git, Mercurial)

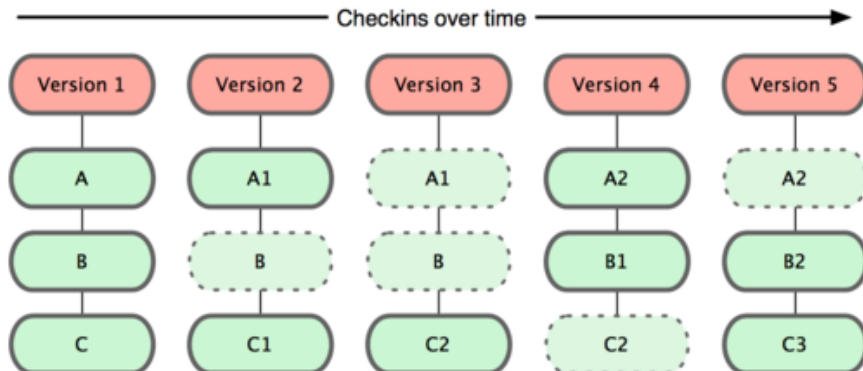
Result: Many operations are local

Git takes snapshots

Subversion



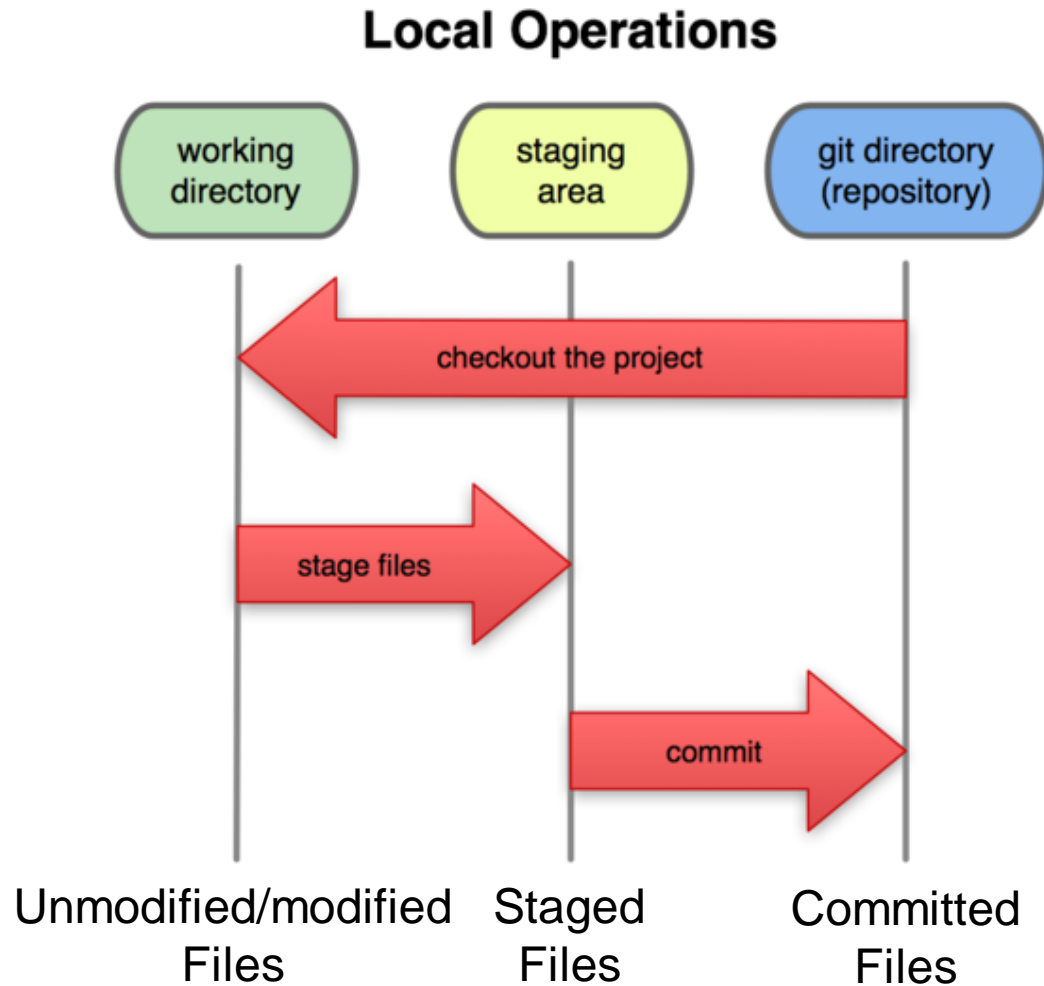
Git



Git uses checksums

- In Subversion each modification to the central repo incremented the version # of the overall repo.
- How will this numbering scheme work **when each user has their own copy of the repo**, and commits changes to their local copy of the repo before pushing to the central server????
- Instead, Git generates a unique SHA-1 hash – 40 character string of hex digits, for every commit. 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

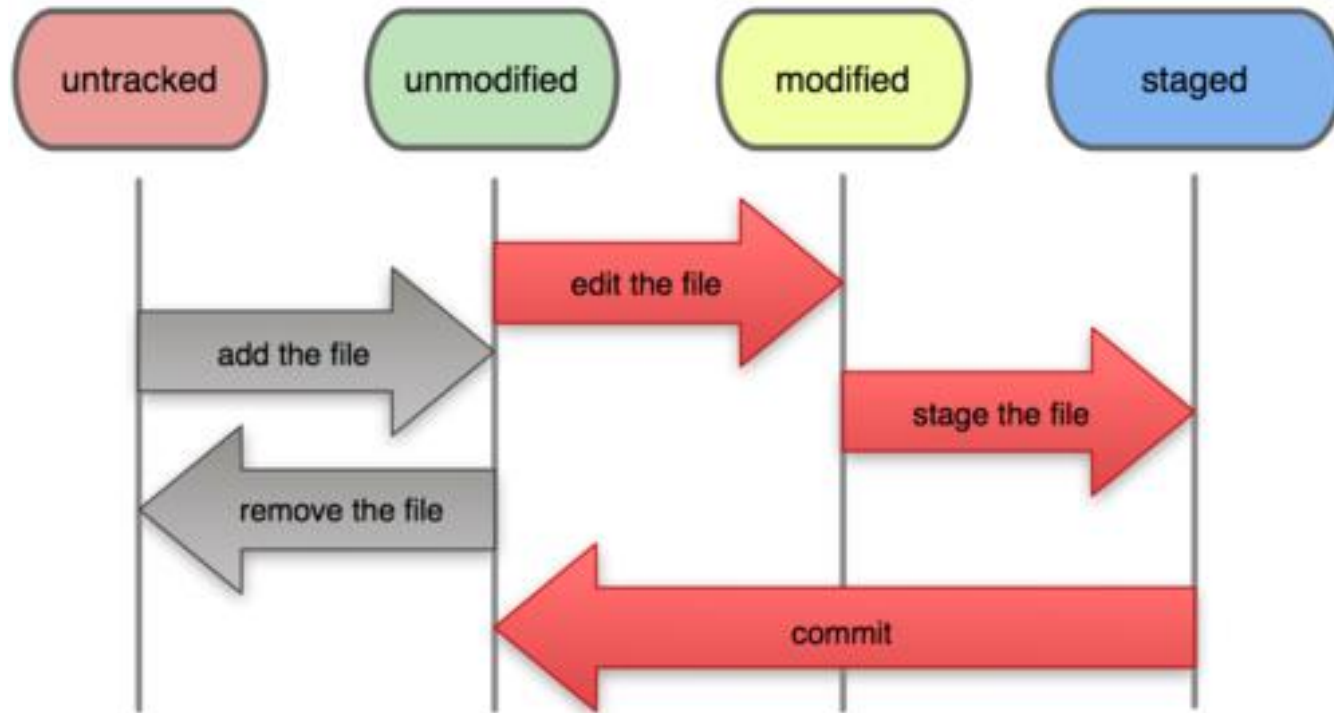
A Local Git project has three areas



Note: working directory sometimes called the “working tree”, staging area sometimes called the “index”.

Git file lifecycle

File Status Lifecycle



Basic Workflow

Basic Git workflow:

1. **Modify** files in your working directory.
2. **Stage** files, adding snapshots of them to your staging area.
3. 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**.

Aside: So what is github?

- [GitHub.com](https://github.com) is a site for online storage of Git repositories.
- Many open source projects use it, such as the [Linux kernel](https://www.kernel.org/).
- 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, such as we did for homework 9 (as long everyone has the needed file permissions).

Get ready to use Git!

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

```
$ git config --global user.name "Bugs Bunny"
```

```
$ git config --global user.email bugs@gmail.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 of a repo

2. Two common scenarios: (only do one of these)

a) To clone an already existing repo to your current 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)

b) 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 version"
```

Git commands

command	description
<code>git clone <i>url</i> [<i>dir</i>]</code>	copy a git repository so you can add to it
<code>git add <i>files</i></code>	adds file contents to the staging area
<code>git commit</code>	records a snapshot of the staging area
<code>git status</code>	view the status of your files in the working directory and staging area
<code>git diff</code>	shows diff of what is staged and what is modified but unstaged
<code>git help [<i>command</i>]</code>	get help info about a particular command
<code>git pull</code>	fetch from a remote repo and try to merge into the current branch
<code>git push</code>	push your new branches and data to a remote repository
others: <code>init</code> , <code>reset</code> , <code>branch</code> , <code>checkout</code> , <code>merge</code> , <code>log</code> , <code>tag</code>	

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 similar to svn)

- To see what is modified but unstaged:

```
$ git diff
```

- To see staged changes:

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


After editing a file...

```
[rea@attu1 superstar]$ emacs rea.txt
```

```
[rea@attu1 superstar]$ 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:   rea.txt
```

```
#
```

```
no changes added to commit (use "git add" and/or "git commit -a")
```

```
[rea@attu1 superstar]$ git status -s
```

```
M rea.txt
```

← Note: M is in second column = "working tree"

```
[rea@attu1 superstar]$ git diff
```

← Shows modifications that have not been staged.

```
diff --git a/rea.txt b/rea.txt
```

```
index 66b293d..90b65fd 100644
```

```
--- a/rea.txt
```

```
+++ b/rea.txt
```

```
@@ -1,2 +1,4 @@
```

```
Here is rea's file.
```

```
+
```

```
+One new line added.
```

```
[rea@attu1 superstar]$ git diff --cached
```

← Shows nothing, no modifications have been staged yet.

```
[rea@attu1 superstar]$
```

After adding file to staging area...

```
[rea@attu1 superstar]$ git add rea.txt
```

```
[rea@attu1 superstar]$ git status
```

```
# On branch master
```

```
# Changes to be committed:
```

```
# (use "git reset HEAD <file>..." to unstage)
```

```
#
```

```
#    modified:   rea.txt
```

```
#
```

```
[rea@attu1 superstar]$ git status -s
```

```
M rea.txt
```

```
[rea@attu1 superstar]$ git diff
```

```
[rea@attu1 superstar]$ git diff --cached
```

```
diff --git a/rea.txt b/rea.txt
```

```
index 66b293d..90b65fd 100644
```

```
--- a/rea.txt
```

```
+++ b/rea.txt
```

```
@@ -1,2 +1,4 @@
```

```
Here is rea's file.
```

```
+
```

```
+One new line added.
```

← Note: M is in first column = "staging area"

← Note: Shows nothing, no modifications that have not been staged.

← Note: Shows staged modifications.

Viewing logs

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

- `$ git log` or
- `$ 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)

Note: On attu you will get a Gtk-warning, you can ignore this.

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*!

SVN vs. Git

- SVN:
 - central repository approach – the main repository is the only “true” source, only the main repository has the complete file history
 - Users check out local copies of the current version
- Git:
 - Distributed repository approach – every checkout of the repository is a full fledged repository, complete with history
 - Greater redundancy and speed
 - Branching and merging repositories is more heavily used as a result

Do This:

1. `$ git config --global user.name "Your Name"`
2. `$ git config --global user.email youremail@whatever.com`
3. `$ git clone https://github.com/rea2000/santalist.git`

Then try:

1. `$ git log, $ git log --oneline`
 2. Create a file named `userID.txt` (e.g. `rea.txt`)
 3. `$ git status, $ git status -s`
 4. Add the file: `$ git add userID.txt`
 5. `$ git status, $ git status -s`
 6. Commit the file to your local repo:
`$ git commit -m "added rea.txt file"`
 7. `$ git status, $ git status -s, $ git log --oneline`
- *WAIT, DO NOT GO ON TO THE NEXT STEPS UNTIL YOU ARE TOLD TO!!**
1. Pull from remote repo: `$git pull origin master`
 2. Push to remote repo: `$git push origin master`