git

Git History

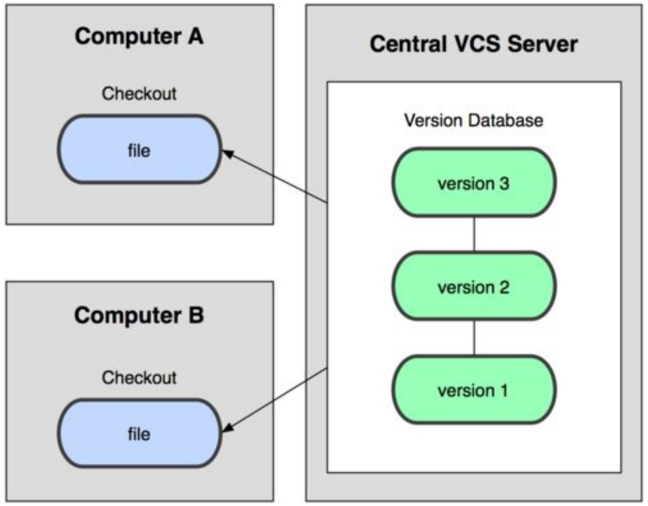
- Came out of Linux development community
- Linus Torvalds, 2005
- Initial goals:
  - Speed
  - Support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects like Linux efficiently

SCM Terminologies

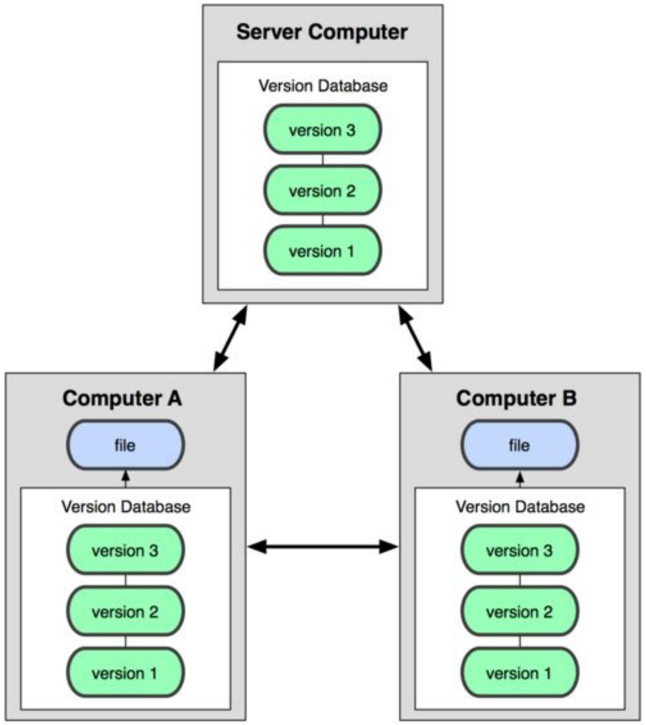
- Server/Client
- Repositorys
- Workspace
- Branch
- Checkin/Checkout
- Revision
- Baseline

Distributed version control system

Centralized Model



Distributed Model



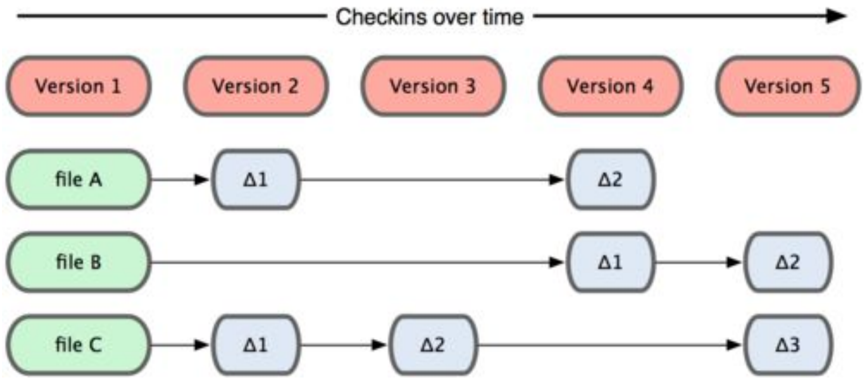
(CVS, Subversion, Perforce)

(Git)

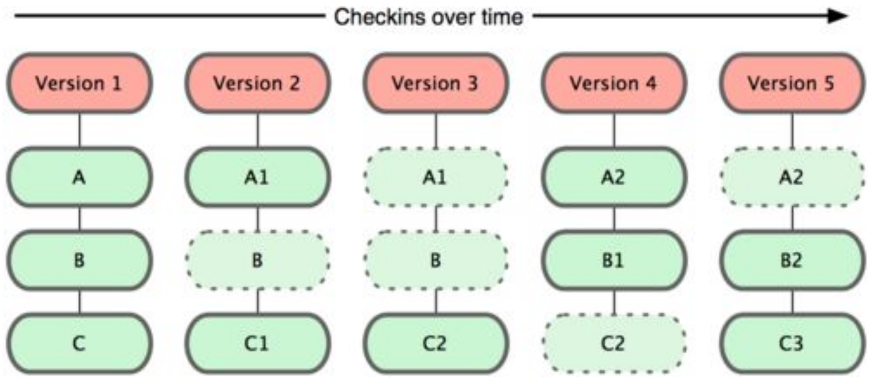
Result: Many operations are local

Git takes snapshots

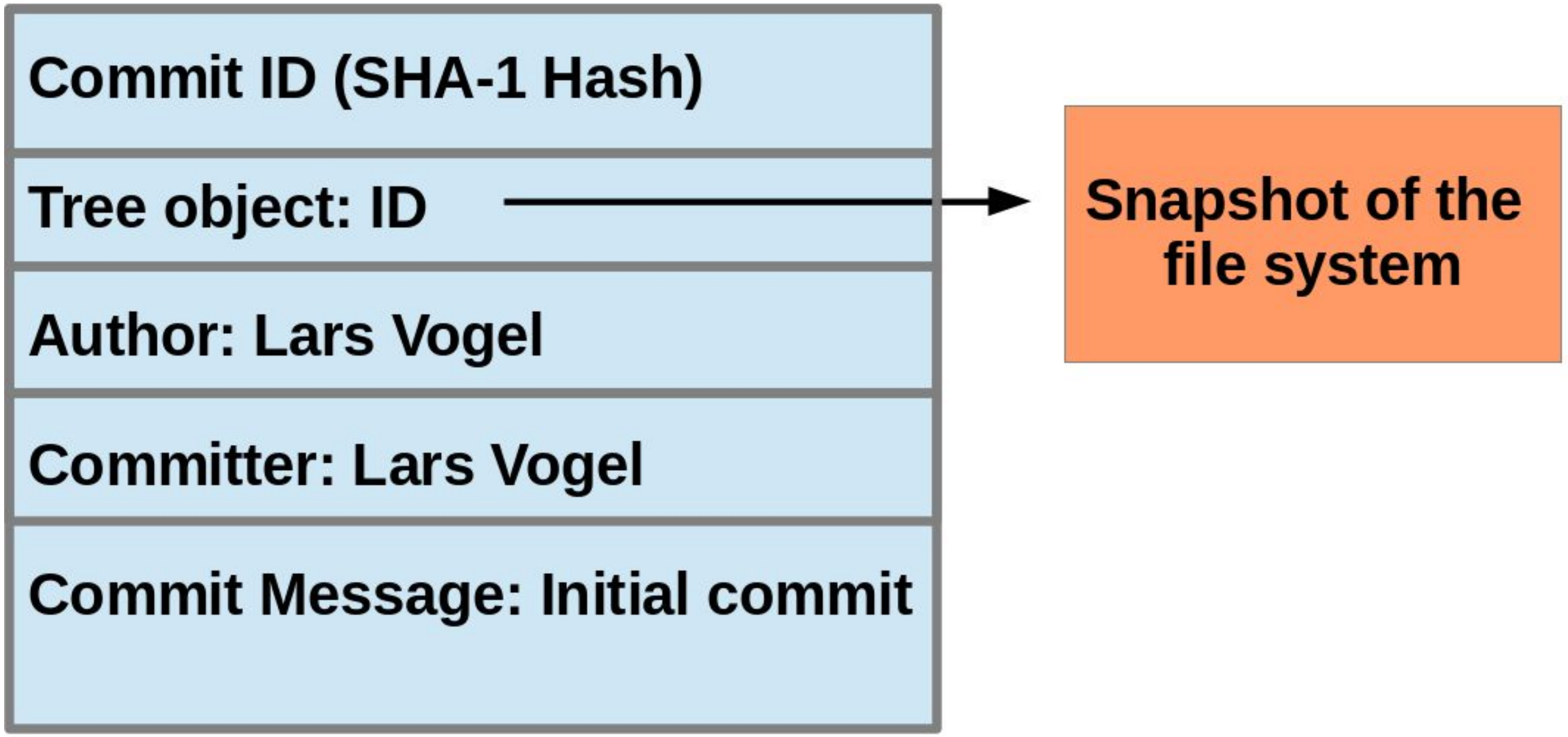
Subversion



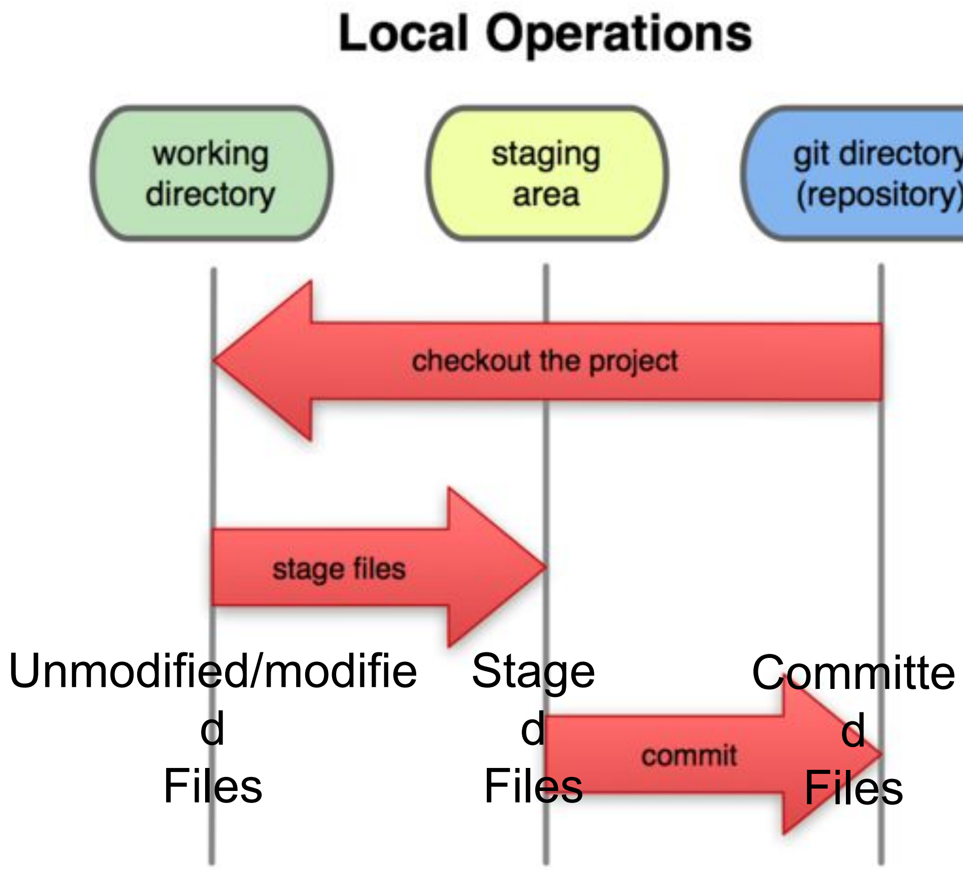
Git



Git uses checksums

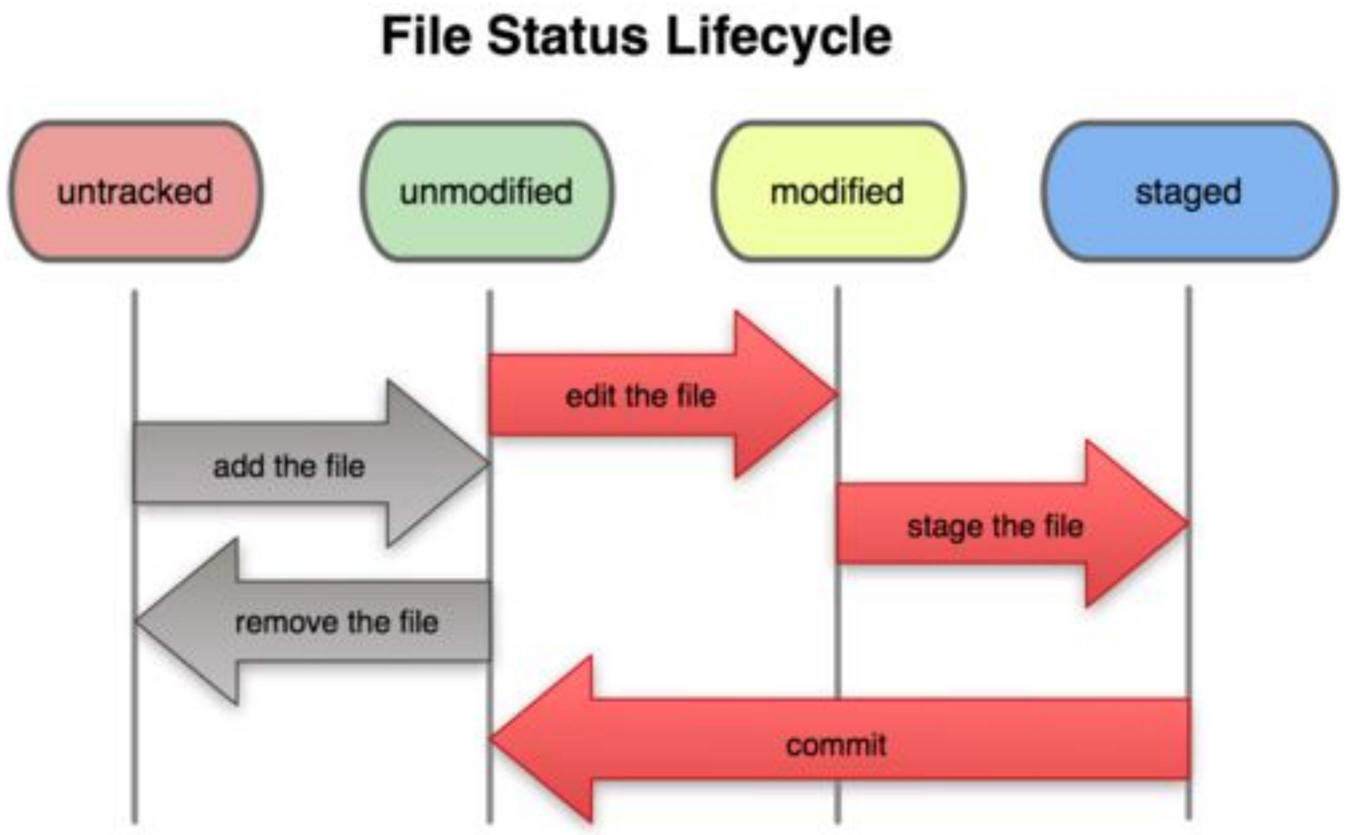


A Local Git project has three areas



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

Git file lifecycle



Aside: So what is github?

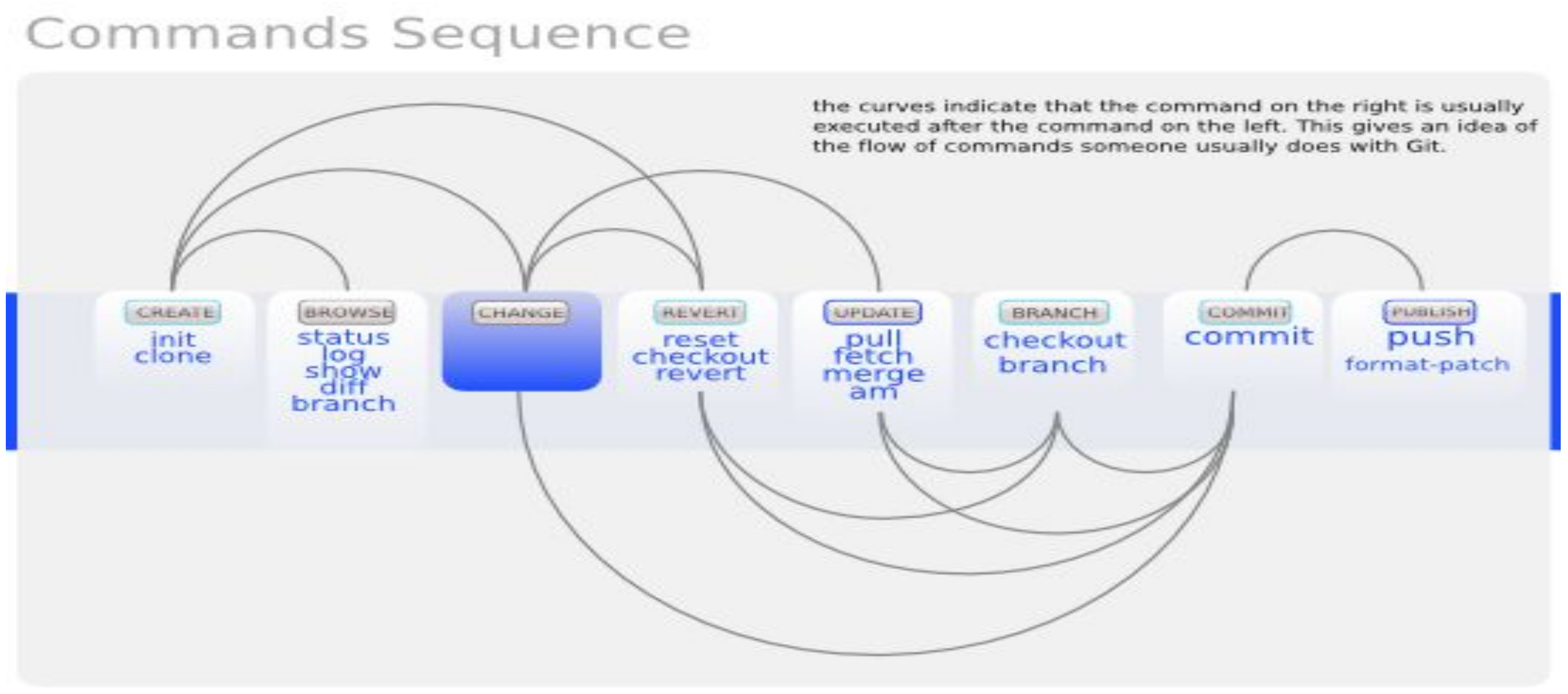
- GitHub.com is a site for online storage of Git repositories.
- Many open source projects use it, such as the Linux kernel.
- You can get free space for open source projects or you can pay for private projects.

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).

Basic Workflow


Commands Sequence
the curves indicate that the command on the right is usually executed after the command on the left. This gives an idea of the flow of commands someone usually does with Git.
CREATE
init
clone
BROWSE
status
log
show
diff
branch
CHANGE
REVERT
reset
checkout
revert
UPDATE
pull
fetch
merge
am
BRANCH
checkout
branch
COMMIT
commit
PUBLISH
push
format-patch

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
$ git config --global push.default simple
```

1. You can call **`git config –list`** to verify these are set.
2. These will be set globally for all Git projects you work with.
3. You can also set variables on a project-only basis by not using the **`--global`** flag.
4. You can also set the editor that is used for writing commit messages:
   $ git config --global core.editor emacs (it is vim by default)

Repositories

- bare repository
  - contains the version control information and no working files
  - by convention the name of a bare repository should end with the .git extension

  $ git init --bare
- Non-bare repository
  - These are regular user repository which has the working files & .git dir

- Creating a new reposiotory

  $ git init

  $ git clone <remote> <local>

Git commands

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

Add, Status and Diff

- To add changes to the staging area :

```
$ git add <file>
```

- 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
```

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:

```
$ 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)

Ignoring files & Viewing logs

Ignoring certain files and directories

- **.gitignore**
- Git never ignores files which are already tracked, so changes in the .gitignore file only affect new files
- Commit the .gitignore to the Git repository

File version

```
$ git log
$ git log --oneline --grep "workspace"
$ git shortlog
$ git log -1
```

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`

Difference between branches:

- `$ git diff master <your_branch>`

`Merge branches :`

- `$ git merge <source_branch> <destnation_branch>`

Stashing commited changes

creates stash, remove changes from working dir:

- `$ git stash`

To list all stash available for the repository :

- `$ git stash list`

Reapply the changes, remove stash:

- `$ git stash pop`

Apply a specific stash from repo:

- `$ git stash apply stash@{num}`

`Remove stash from repo`

- `$ git stash clear`

Reverting changes

Revert uncommited changes:

- `$ git reset <file>`

Move only to HEAD pointer:

- `$ git reset --soft`

Move the HEAD pointer & reset the staging area (default):

- `$ git reset --mixed`

Move the HEAD pointer, resets staging area & working tree to the new HEAD:

- `$ git reset --hard`

Reverting changes …

Revert a commit:

- `$ git revert <commit>`

Checkout specific commit:

- `$ git checkout <commit_id>`

Deleting a file:

- `$ git rm <file>`

Removing untracked file:

- `$ git clean -n (-n is for dry run)`
- `$ git clean -f (force delete)`

`Discard changes in working directory`

- `$ git checkout -- <file>`

Tags

Git has the option to *tag* a commit in the repository history so that you find it easier at a later point in time

Apply tag to a commit:

- **`$ git tag –a <pattern> -m ‘comment’ <commitid>`**

Contents of the tag:

- **`$ git show <pattern>`**

Display list of tags available:

- **`$ git tag`**

Delete a tag:

- $ git tag –d <tag>

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**