Version 2013-1
as of 31 May 2013

Scientific and Technical Computing

Git Tutorial:

Distributed Source Control Management

BY NC

THE UNIVERSITY OF TEXAS AT AUSTIN
Texas Advanced Computing Center

Outline

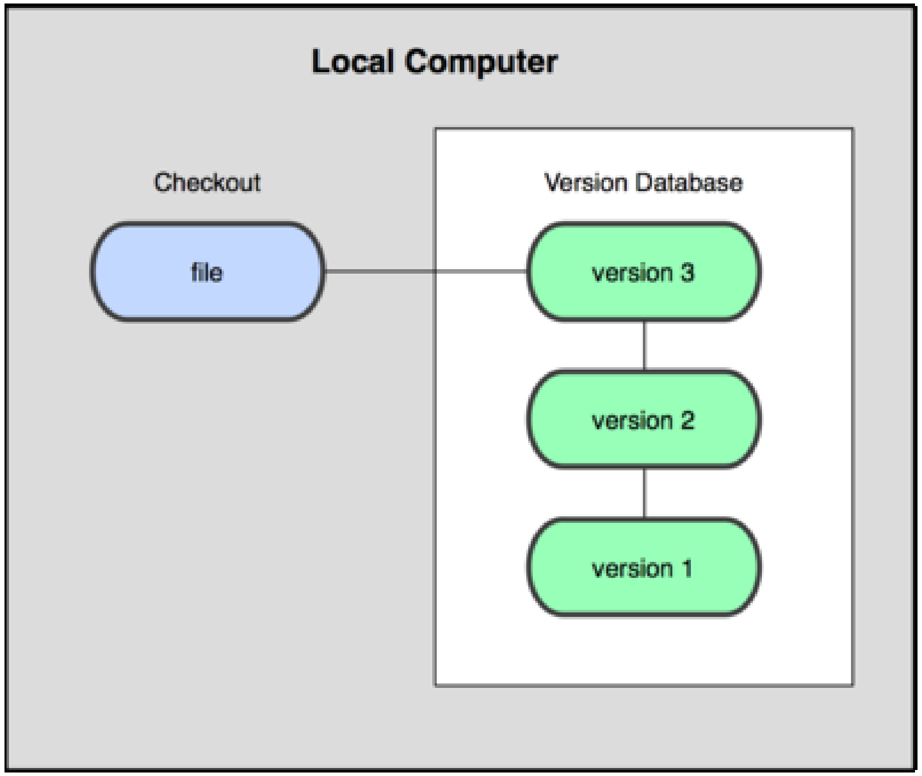
- Source Control Management
- Basic Git Usage
- Branches, Forks, and more

See:
http://git-scm.com
https://bitbucket.com

Why use source control?

• **Reproducibility** –	Versions maintained with comments– has history.
• **Traceability** -	Records author and timestamp.
• **Collaboration** -	Sequential or parallel (branched) development updates. Allow contributions without risking code breakage.
• Organization	Enforces a method of organization

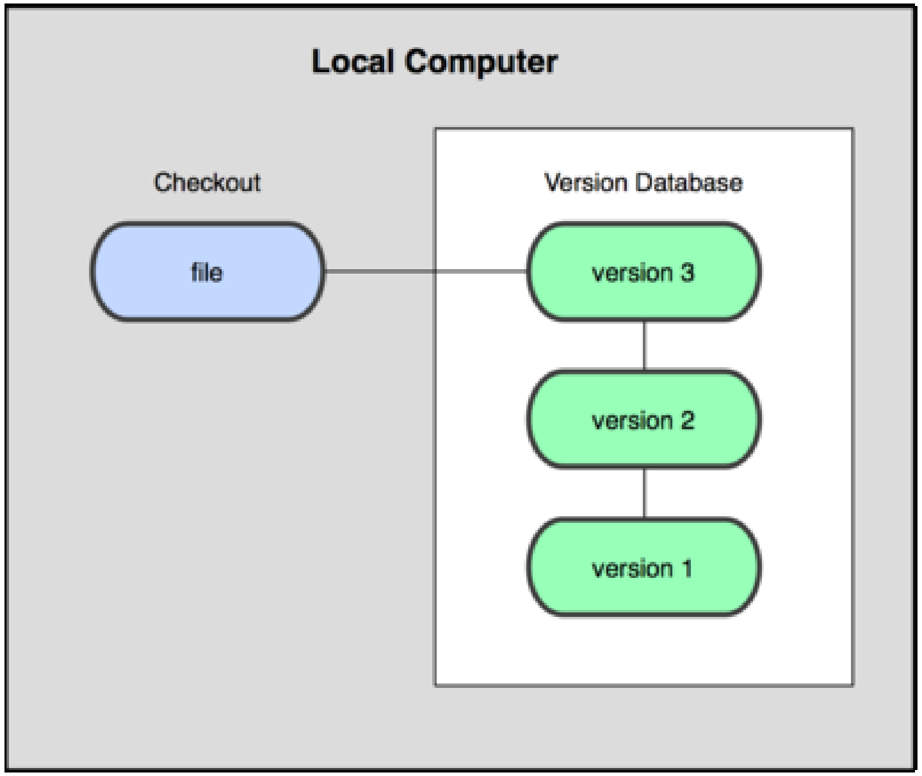
Local Source Control Management



[Image credit: http://git-scm.com]*

- "Database" keeps versions of the file that can be "checked out"
- Edit and revise local files
- Use smart tools to see differences in the files

Local Source Control Management



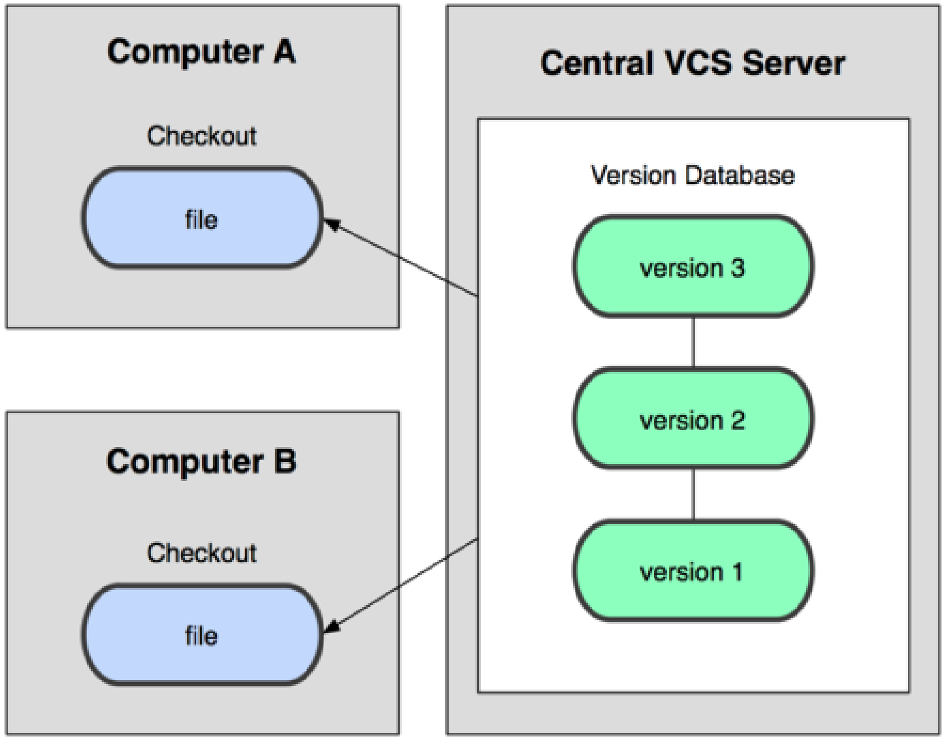
[Image credit: http://git-scm.com]*

Examples

- SCCS (1972)
- RCS (1982)

- Locking mechanism gives exclusive rights to a user.

Centralized Source Control Management

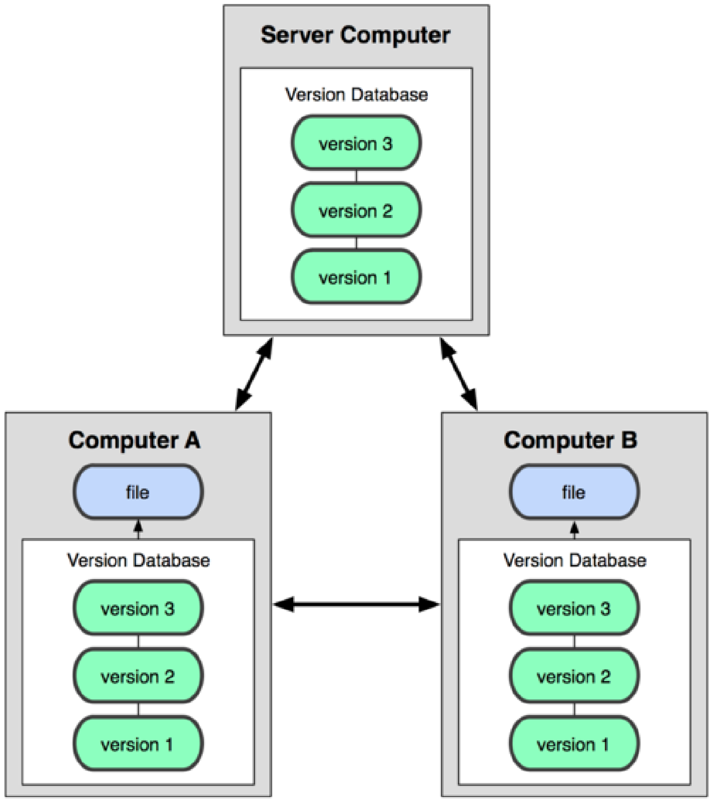


[Image credit: http://git-scm.com]*

Examples

- CVS (1989)
- SVN (2000)
- ClearCase
- Perforce

Distributed Source Control Management



[Image credit: http://git-scm.com]*

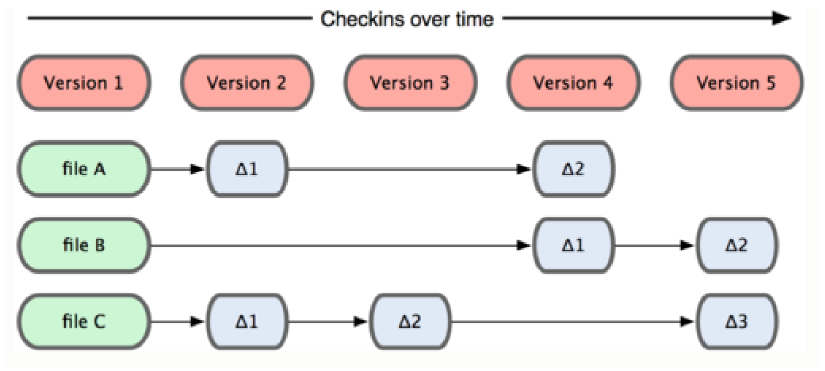
Examples

- Bitkeeper (2000)
- Darcs (2003)
- Git (2005)
- Bazaar (2005)
- Mercurial (2005)

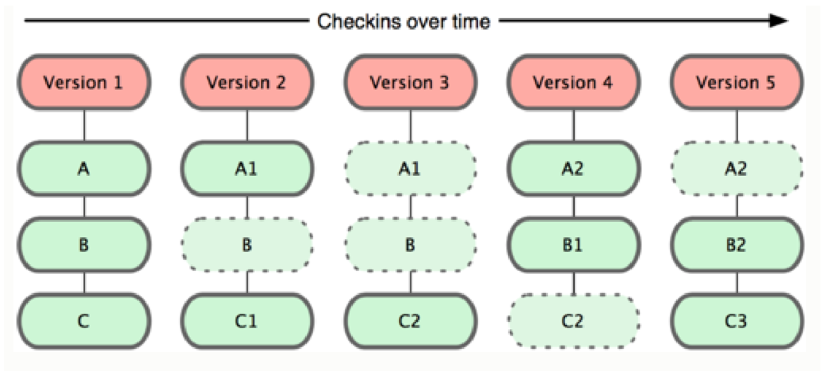
Often called a DVCS, distributed version control system.

Git is different

- A commit creates a version for your file changes (common to most SCMs).
- A commit of your project or file is a snapshot at that moment which has a reference to it. (There is a separate copy, not just a “delta”.)



Change- (delta) based System



Git, Snapshot-based System

[Image credit: http://git-scm.com]*

Outline

- Source Control Management
- Basic Git Usage
- Branches, Forks, and more

Local Commands

Use lab machine, lonestar/stampede or laptop (Macies: download binaries in a dmg. Windows: use stampede.)

git help gives list of commands. You can use man pages, too.

git help <command> gives details of a command.

```
$ ssh stampede.tacc.utexas.edu  or lonestar.tacc.utexas.edu
$ module load git              #if on stampede or lonestar
$ git help
usage: git ...
$ git help init
GIT-INIT(1)                                            Git Manual
...
```

Local Commands

`git init`: Create an empty git repository or reinitialize and existing one.

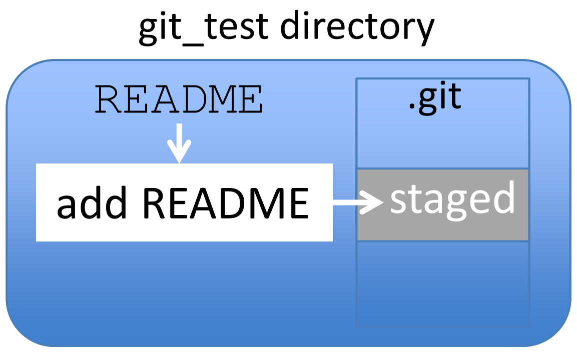
```
$ mkdir git_test
$ cd git_test
$ git init
Initialized empty Git repository in
/home1/01392/aterrel/git_test/.git/
```

`.`git is your local repository.

Local Commands

`git add`: Add file contents to the index (of files) and stages present copy for commitment.

```
$ echo "Hello Git World" >> README
$ git add README
```



Local Commands

`git status`: Show the working tree status

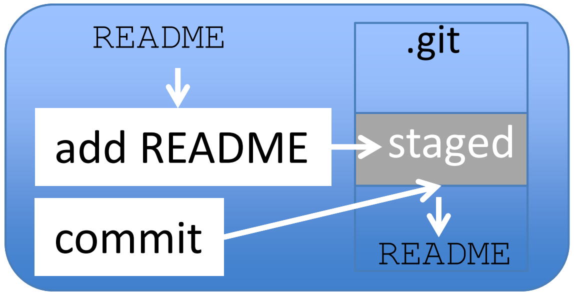
```
$ git status
# On branch master
#
# Initial commit
# 1
# Changes to be committed:
# (use "git rm --cached <file>..." to unstage)
#  2
# new file: README
```

Shows no commitments (1)and a staged file (2) .

Local Commands

`git commit`: Record changes to the repository

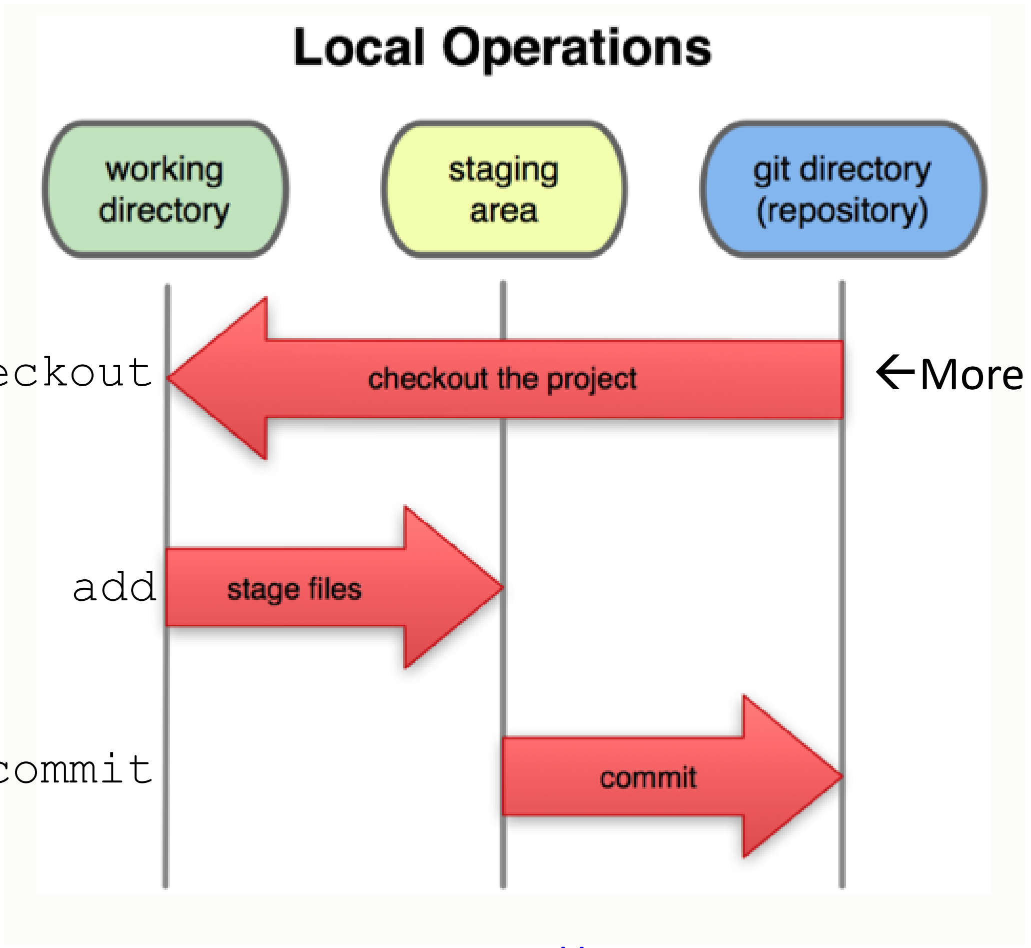
```
$ git commit -m "Adding README"
[master (root-commit) 774c810] Adding README
1 file changed, 1 insertion(+)
create mode 100644 README
```



May get message to set your
user name and email– so that it knows details of the author.

```
$ git config --global user.name "Your Name"
```

Add and Commit



←More about this later on.

[Image credit: http://git-scm.com]*

Local Commands

`git log`: Show the commit logs

```
$ git log
commit 774c81087d052e43a630db7f676cfd9a6b006772
Author: Andy R. Terrel <andy.terrel@gmail.com>
Date: Tue Jul 24 17:53:04 2012 -0500


 1  Adding README
```

Note comment from commit –m" option 1 Adding README).
Make your comments (history) meaningful.

TACC

Local Commands

```
$ echo "Line 2" >> README
$ git add README
$ git commit -m "Adding Line 2"
$ echo "Line 3" >> README
$ git add README
$ git commit -m "Adding Line 3"
$ echo "Clear file" > README

$ git commit -am "Clear file"
```

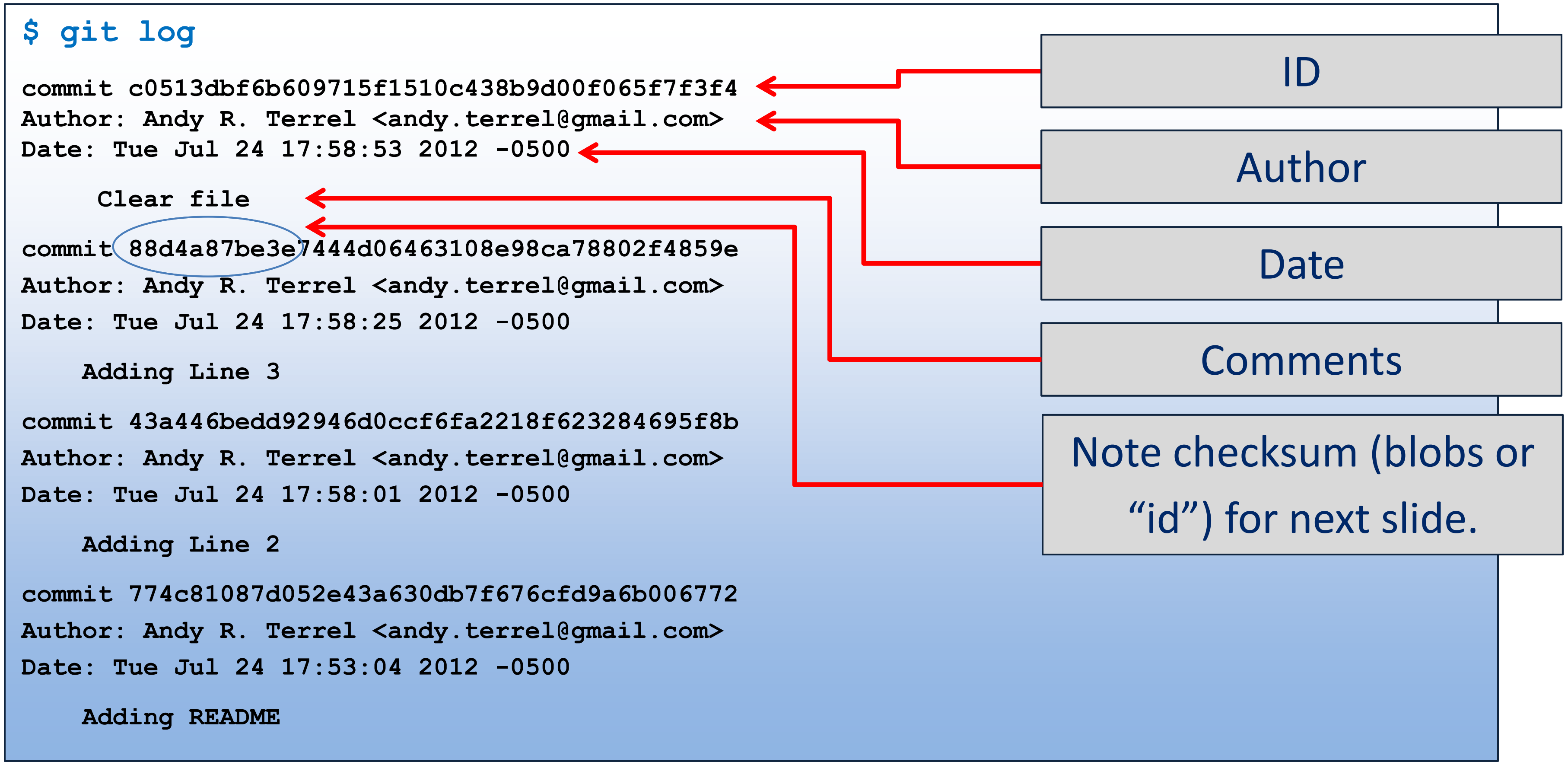
">" deletes previous contents of README

add & commit combined, all modified and indexed files

```
$ git commit      –m "Clear file" README     #add/commit a file
$ git commit –p –m "Clear file"              # query add/commit files
```

alternate forms

Local Commands



```
$ git log

commit c0513dbf6b609715f1510c438b9d00f065f7f3f4
Author: Andy R. Terrel <andy.terrel@gmail.com>
Date: Tue Jul 24 17:58:53 2012 -0500

     Clear file

commit 88d4a87be3e7444d06463108e98ca78802f4859e
Author: Andy R. Terrel <andy.terrel@gmail.com>
Date: Tue Jul 24 17:58:25 2012 -0500

    Adding Line 3

commit 43a446bedd92946d0ccf6fa2218f623284695f8b
Author: Andy R. Terrel <andy.terrel@gmail.com>
Date: Tue Jul 24 17:58:01 2012 -0500

    Adding Line 2

commit 774c81087d052e43a630db7f676cfd9a6b006772
Author: Andy R. Terrel <andy.terrel@gmail.com>
Date: Tue Jul 24 17:53:04 2012 -0500

    Adding README
```

Note checksum (blobs or "id") for next slide.

```
$ git log README          #can view log for individual files.
```

Local Commands

`git diff`: Show changes between commits, commit and working tree, etc.

```
$ git diff README              #--- staged  +++ modified
diff --git a/README b/README
index d5c15a2..fcb6062 100644
--- a/README
+++ b/README
@@ -1,3 +1 @@
-Hello Git World
-Line 2
-Line 3
+Clear file
```

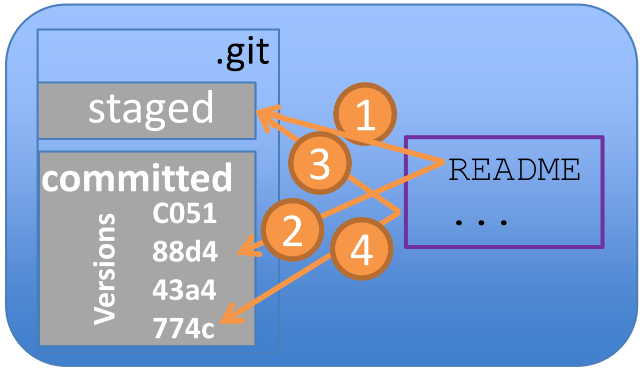
Local Commands

Types of differences:

Comparison

```
$ git diff          README      1 staged     file  with modified README
$ git diff 88d4 README          2 ver 88d4… file  with modified README
$ git diff                      3 staged     files with modified files
$ git diff 774c                 4 ver 88d4… files with modified files
```

If there are no staged files,
diff occurs on latest version.



Local Commands

`git checkout`: Checkout a branch or paths to the working tree

```
$ git checkout 88d4a87be3e7
Note: checking out '88d4a87be3e7'.

You are in 'detached HEAD' state. You can look around, make experimental changes and
commit them, and you can discard any commits you make in this state without impacting
any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may do so (now or
later) by using -b with the checkout command again. Example:

    git checkout -b new_branch_name

HEAD is now at 88d4a87... Adding Line 3

$ git checkout master
Previous HEAD position was 88d4a87... Adding Line 3
Switched to branch 'master'
```

Reverts (files) to snapshot 88..

More on branches (-b) later.

Revert back to master snapshot (path)

These are the comments!

Remote Commands

`git clone`: Clone a repository into a new directory

`git pull`: Fetch from and merge with another repository or a local branch

`git push`: Update remote refs along with associated objects

Remote Commands

Bitbucket Repository:

Supports git and other protocols

After creating empty repository:

Import at bitbucket or push files from local system.

For convenience name local directory of repository and remote repository the same name.

```
$ #@bitbucket create repository STC
$ mkdir STC; cd STC  #create local repo
$ date > README
$ git commit -am "new README"
```

Push to a Server Repository

```
$ git remote add origin \
  ssh://git@bitbucket.org/milfeld/STC.git
```

<protocol://<site>/<user>/<repo_name>.<repo_type>

```
$ git push -u origin -all
```

First time: push ALL up to site, declare local as upstream

```
$ echo '// No line return' >>p.c
$ git commit -am '2nd commit'
$ git push origin master
```

Subsequent pushes: from local master to origin.

Summary of Useful Commands

`git status`	Show the working tree status
`git log`	Show commit logs
`git tag`	Create, list, delete, or verify a tag object signed with GPG
`.gitignore`	include *.o *.a .gitignore (1 line each)
`git diff`	Show changes between commits, commit, and working tree, etc.
`git branch -a`	Lists all branches.
`git remote add <rem_nam> <sit>`	add a remote branch

Outline

- Source Control Management
- Basic Git Usage
- Branches, Forks, and more

Branch Commands

`git branch`	Lists, creates, or deletes branches.
`git merge`	Joins branches together.
`git rebase`	Another form of merge that serializes changes into an easy to follow history.

Why Branches?

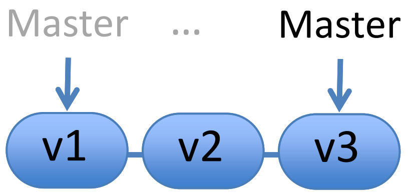
Branches act as "silos" for a package:

master	production quality
proposed	ready for master (early users)
development	on-going work
topic	early development

Context switching

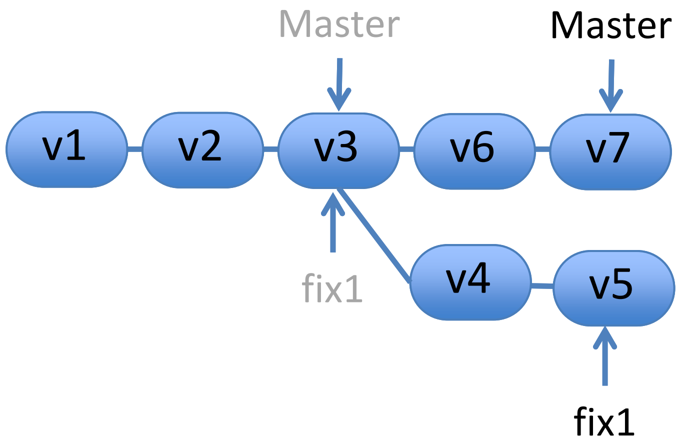
Container for historical information

Git Branch



```
$...
$ git commit -am "2nd change"
$ …
$ commit -am "3rd change"
```

Master follows subsequent commits.



```
$ git branch fix1
$ git checkout fix1
$ <changes with 2 commits>
```

New fix1 branch points also to Master.

Checkout makes copy & fix1 follows commits.

Use git checkout master or fix1 to switch back and forth (same directory) between branches.

```
$ git checkout master
$ <changes with 2 commits>
```

Example 1 Workflow

```
$ git checkout -b "opt1"        # -b = branch; optimizing code….
$ git commit  -am "opt1 branch"
                                # create new branch for immediate fix
$ git checkout master           # get back to master
$ git checkout -b "fix2"        # then edit, compile, test & commit
$ git commit -am "quick fix 2"  # and next merge it back to master.
$ git checkout master   # You have the Master as the reference
$ git merge fix2        # Merging fix2 INTO Master.
                        # If single revision, then no conflicts.
$ git branch -d fix2    # Remove the fix2 branch (it has been merged).
$ git checkout opt1     # Go back to working on opt1
$ <work, commit, finish> #optimization (opt1)  done
$ git checkout master
$ git merge opt1
    The directory will contains files show conflicts.
    Fix noted differences in files between chevrons:
        <<<<<<< HEAD
        something changed in Master....
       =======
        things changed in opt1
       >>>>>>> opt1
$ git commit
$ git branch -d opt1
```

Setting up rsa keys

- Key generation:

```
$ ssh-keygen  -f  $HOME/.ssh/rsa_id_bb
```

(this makes rsa_id_bb and rsa_id_bb.pub files– private & public keys in $HOME/.ssh)

- Cat the contents of rsa_id_bb.pub and put it into bitbucket.

```
$ cat $HOME/.ssh/rsa_id_bb.pub
```

in browser: bitbucket.org→avatar→Manage Accounts→SSH keys→Add

- Make the file $HOME/.ssh/config with the following:

```
Host bitbucket.org
 User git
 Hostname bitbucket.org
 IdentityFile ~/.ssh/rsa_id_bb
```

Hostname, can use aliases on this line

You login as user git!!!

Actual hostname

Where to find the rsa key for host.

- Make sure it works:

```
$ ssh -T git@bitbucket.org
logged in as milfeld.
You can use git …. Shell access is disabled.
```

Access, what can go wrong?

- Access is denied if you get this result:

```
$ ssh -T git@bitbucket.org
Permission denied (publickey).
```

- Make sure you cut and paste key as a single line. (There is a space after ssh-rsa.)

```
ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAIEA4xVc9fj1lLzynrXYeZcAyMG0d5NJSx9ZkAnwUavtLDXOl6XtkMHym6v/G32A20k9OHMfSuzmp
1kMBmyGjErdYJNxTj3M8WC/EHYS51dkGwzfUH5Irvb49nvY6NH8UVrYIn7bgyELzP0VZPnYgKSbUpkKLT0pH5yryKy2/GTaM8s=
milfeld@login1.ls4.tacc.utexas.edu
```

- Use the verbose form of ssh to see more details, make sure ssh is using the right key for bitbucket. `$ ssh -Tv git@bitbucket.org`

```
$ ssh -vT git@bitbucket.org
OpenSSH_4.3p2, OpenSSL 0.9.8e-fips-rhel5 01 Jul 2008
debug1: Reading configuration data /home1/00770/milfeld/.ssh/config
...
debug1: identity file /home1/00770/milfeld/.ssh/rsa_id type -1
...
debug1: Trying private key: /home1/00770/milfeld/.ssh/rsa_id
debug1: No more authentication methods to try.
Permission denied (publickey).
```

Tips and Tricks

- You must add a new file, and then commit it. git commit -a will not work for a new file.

- www.gitguys.com/topics

We gratefully acknowledge the sponsorship of Chevron Corporation, whose generous support of TACC has made possible this Scientific Computing Curriculum and other student-focused initiatives.

License