



Trilinos Git Tutorial

Elijah Newren

Sandia National Laboratories

November 5, 2009

version: UNKNOWN

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.



Sandia
National
Laboratories

- 1 Preliminaries
 - Scope
 - Documentation
 - Easy Git
 - Installation
 - First Time Setup
 - Migrating Changes from Existing CVS Projects
- 2 When Things Go Well: The Basics
- 3 When Things Go Wrong: Common Error Messages

Git has a *lot* of cool features for developers.

Git has a *lot* of cool features for developers.

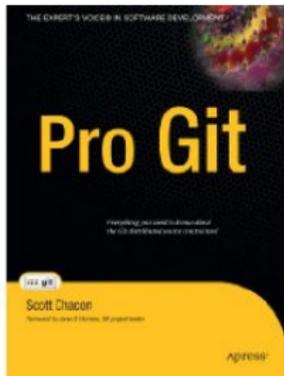
But I won't be covering them today.

Git has a *lot* of cool features for developers.

But I won't be covering them today.

I'm focusing enabling you to do it git what you now do in CVS.

- Book



Available free online: <http://progit.org/book>

Chapter 2 has all the basics.

- Built-in

```
$ eg help [command]
```

Easy Git

- Changes many defaults to be similar to CVS/SVN
- Single-file wrapper script for Git
- Designed to make Git easy to learn and use
- Focuses on documentation and examples

We'll only cover using `eg`

Easy Git

- Changes many defaults to be similar to CVS/SVN
- Single-file wrapper script for Git
- Designed to make Git easy to learn and use
- Focuses on documentation and examples

We'll only cover using `eg`

Assuming you already have a CVS checkout of Trilinos:

```
$ BASE_DIR/Trilinos/cmake/python/install-git.py \  
    -do-all -install-dir=SOME_DIR  
$ export PATH=SOME_DIR/bin:$PATH
```

Test it out:

```
$ eg clone software.sandia.gov:/space/git/temp/Trilinos
```

The *temp* part of the path will disappear after the real conversion.

WARNING: You need eg ≥ 0.995 . (Check via `eg --version`). Get a new copy from

<http://www.gnome.org/~newren/eg/download/latest/eg>
and stick it in your \$PATH.

Assuming you already have a CVS checkout of Trilinos:

```
$ BASE_DIR/Trilinos/cmake/python/install-git.py \  
    -do-all -install-dir=SOME_DIR  
$ export PATH=SOME_DIR/bin:$PATH
```

Test it out:

```
$ eg clone software.sandia.gov:/space/git/temp/Trilinos
```

The *temp* part of the path will disappear after the real conversion.

WARNING: You need eg ≥ 0.995 . (Check via `eg --version`). Get a new copy from

<http://www.gnome.org/~newren/eg/download/latest/eg>
and stick it in your \$PATH.

First Time Configuration (~/.gitconfig)

```
# Set your username
$ eg config --global user.name "Copy N. Paste"

# Set your email address
$ eg config --global user.email -----@sandia.gov

# Use colored output when it makes sense
$ eg config --global color.ui true

# Set your favorite diff & merge tool
# (Choices include tkdiff, vimdiff, meld, kdiff3, etc.)
$ eg config --global merge.tool meld
```

See `git config --help` for more...



Migrating Work from Existing CVS Projects

Download: `scp software.sandia.gov:/space/git/temp/merge-cvs-changes.py .`

From within a clone of the Trilinos git repository, run:

```
$ merge-cvs-changes.py /path/to/Trilinos/cvs/checkout
```

Pay close attention to any messages the script writes.

- If there are conflicts:
 - Edit the relevant files to remove conflict markers
 - Mark the file as ready to be committed by running `$ eg stage FILE`
- We'll cover committing these changes later in the presentation.



Migrating Work from Existing CVS Projects

Download: `scp software.sandia.gov:/space/git/temp/merge-cvs-changes.py .`

From within a clone of the Trilinos git repository, run:

```
$ merge-cvs-changes.py /path/to/Trilinos/cvs/checkout
```

Pay close attention to any messages the script writes.

- If there are conflicts:
 - Edit the relevant files to remove conflict markers
 - Mark the file as ready to be committed by running `$ eg stage FILE`
- We'll cover committing these changes later in the presentation.



Migrating Work from Existing CVS Projects

Download: `scp software.sandia.gov:/space/git/temp/merge-cvs-changes.py .`

From within a clone of the Trilinos git repository, run:

```
$ merge-cvs-changes.py /path/to/Trilinos/cvs/checkout
```

Pay close attention to any messages the script writes.

- If there are conflicts:
 - Edit the relevant files to remove conflict markers
 - Mark the file as ready to be committed by running `$ eg stage FILE`
- We'll cover committing these changes later in the presentation.



Migrating Work from Existing CVS Projects

Download: `scp software.sandia.gov:/space/git/temp/merge-cvs-changes.py .`

From within a clone of the Trilinos git repository, run:

```
$ merge-cvs-changes.py /path/to/Trilinos/cvs/checkout
```

Pay close attention to any messages the script writes.

- If there are conflicts:
 - Edit the relevant files to remove conflict markers
 - Mark the file as ready to be committed by running `$ eg stage FILE`
- We'll cover committing these changes later in the presentation.



Migrating Work from Existing CVS Projects

Download: `scp software.sandia.gov:/space/git/temp/merge-cvs-changes.py .`

From within a clone of the Trilinos git repository, run:

```
$ merge-cvs-changes.py /path/to/Trilinos/cvs/checkout
```

Pay close attention to any messages the script writes.

- If there are conflicts:
 - Edit the relevant files to remove conflict markers
 - Mark the file as ready to be committed by running `$ eg stage FILE`
- We'll cover committing these changes later in the presentation.

1 Preliminaries

2 When Things Go Well: The Basics

- Cheet Sheat: CVS → Git
- Clone — Getting your Copy
- Switch, Branch — Dealing with Branches
- Gitk — a local Bonsai replacement, and history viewer
- Add, Mv, Rm — Adding/Renaming/Deleting Files
- Revert — Reverting Uncommitted File Modifications
- Commit — Recording Changes *Locally*
- Status, Diff, Log — Getting info
- Squash
- Push, Pull

3 When Things Go Wrong: Common Error Messages

CVS Usage

Git Usage

\$ **cvs checkout -d**

:ext:USER@MACHINE:/PATH REPOSITORY → \$ **eg clone [USER@]MACHINE:/PATH**

\$ **cvs commit**

→ \$ **eg commit**
\$ **eg push**

\$ **cvs [-q] update [-dP]**

→ \$ **eg commit # To allow backout**
\$ **eg pull**

\$ **cvs update**
\$ **cvs commit**

→ \$ **eg commit # To allow backout**
\$ **eg pull**
\$ **eg squash # To cleanup**
\$ **eg push**

\$ **rm FILE**
\$ **cvs update FILE**

→ \$ **eg revert FILE**

\$ **cvs -H COMMAND**

→ \$ **eg help COMMAND**

\$ **cvs status**

→ \$ **eg status**

\$ **cvs -nq update**

→ \$ **eg status**

\$ **cvs diff -u**

→ \$ **eg diff**

\$ **cvs add FILE**

→ \$ **eg add FILE-OR-DIRECTORY**

\$ **cvs rm [-f] FILE**

→ \$ **eg rm FILE-OR-DIRECTORY**

You're kidding, right?

→ \$ **eg mv FILE-OR-DIR NEWPATH**

\$ **cvs log FILE**

→ \$ **eg log [FILE-OR-DIR]**

Cloning a Repository

```
$ eg clone software.sandia.gov:/space/git/temp/Trilinos
```

```
Initialized empty Git repository in /home/newren/Trilinos/.git/
```

```
remote: Counting objects: 442849, done.
```

```
remote: Compressing objects: 100% (93158/93158), done.
```

```
remote: Total 442849 (delta 340106), reused 442746 (delta 340033)
```

```
Receiving objects: 100% (442849/442849), 438.01 MiB | 11198 KiB/s, done.
```

```
Resolving deltas: 100% (340106/340106), done.
```

```
Checking out files: 100% (29405/29405), done.
```

```
$ eg clone software.sandia.gov:/space/git/temp/Trilinos
Initialized empty Git repository in /home/newren/Trilinos/.git/
remote: Counting objects: 442849, done.
remote: Compressing objects: 100% (93158/93158), done.
remote: Total 442849 (delta 340106), reused 442746 (delta 340033)
Receiving objects: 100% (442849/442849), 438.01 MiB | 11198 KiB/s, done.
Resolving deltas: 100% (340106/340106), done.
Checking out files: 100% (29405/29405), done.
```

*Each “clone” in Git is a **full** repository.*

Network operations: clone, fetch, pull, push

Local operations: everything else (status, diff, log, commit, merge, branch, tag, etc.)

```
$ eg clone software.sandia.gov:/space/git/temp/Trilinos
Initialized empty Git repository in /home/newren/Trilinos/.git/
remote: Counting objects: 442849, done.
remote: Compressing objects: 100% (93158/93158), done.
remote: Total 442849 (delta 340106), reused 442746 (delta 340033)
Receiving objects: 100% (442849/442849), 438.01 MiB | 11198 KiB/s, done.
Resolving deltas: 100% (340106/340106), done.
Checking out files: 100% (29405/29405), done.
```

*Each “clone” in Git is a **full** repository.*

Network operations: clone, fetch, pull, push

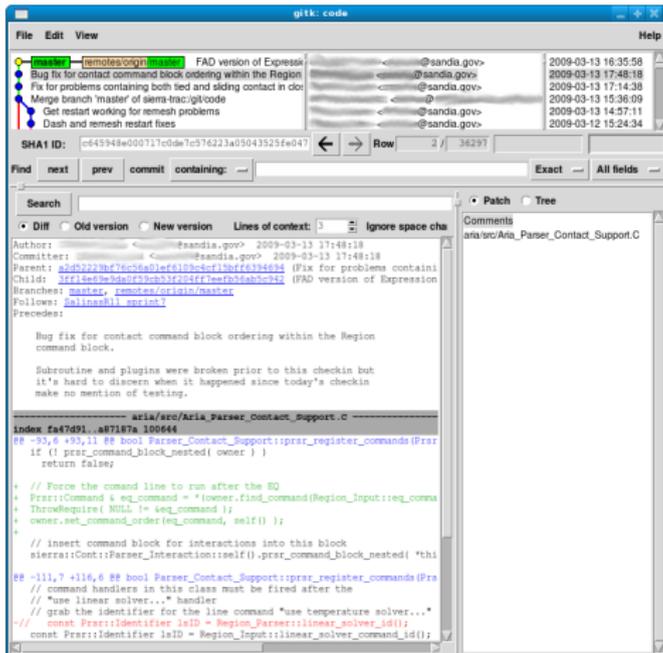
Local operations: everything else (status, diff, log, commit, merge, branch, tag, etc.)

Switching and Creating Branches

- List existing branches:
\$ **eg branch**
- Switch to another branch:
\$ **eg switch trilinos-release-10-0-branch**
- Create a new *local* branch, giving it a starting point of origin/master:
\$ **eg branch mybugfix *origin/master***

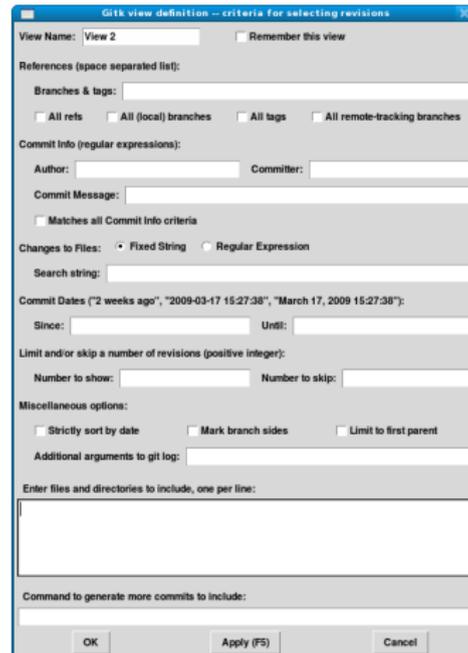
Gitk — viewing history, replacing Bonsai

\$ gitk &



The screenshot shows the Gitk application window titled "gitk: code". The top menu bar includes "File", "Edit", "View", and "Help". The main window is divided into several sections:

- Commit History:** A list of commits with columns for author, date, and time. The selected commit is from "remotes/origin/master" with a SHA1 ID of "e645948e00711e0dde7c576223a05043525fe047".
- Search:** A search bar with "Find" and "next" buttons.
- Diff View:** A view showing the differences between the selected commit and its parent. It includes a "Search" field and a "Comments" section.
- Code Editor:** A large text area displaying the diff of the file "aria/src/Aria_Parser_Contact_Support.C". The code shows changes to the "prsr_register_commands" function, including comments and code blocks.



The screenshot shows the "Gitk view definition -- criteria for selecting revisions" dialog box. It has a "View Name" field set to "View 2" and a "Remember this view" checkbox. The "References (space separated list):" section is empty. The "Branches & tags:" section has three checkboxes: "All refs" (checked), "All (local) branches", and "All tags". The "Commit Info (regular expressions):" section has fields for "Author:", "Committer:", and "Commit Message:". The "Matches all Commit Info criteria" checkbox is unchecked. The "Changes to files:" section has radio buttons for "Fixed String" (checked) and "Regular Expression", and a "Search string:" field. The "Commit Dates" section has a text input field with "2 weeks ago", "2009-03-17 15:27:38", and "March 17, 2009 15:27:38". The "Limit and/or skip a number of revisions (positive integer):" section has "Number to show:" and "Number to skip:" fields. The "Miscellaneous options:" section has three checkboxes: "Strictly sort by date", "Mark branch sides", and "Limit to first parent". The "Additional arguments to git log:" field is empty. The "Enter files and directories to include, one per line:" field is empty. The "Command to generate more commits to include:" field is empty. The dialog has "OK", "Apply (F5)", and "Cancel" buttons at the bottom.

Dialog on right from View→New View (or press F4).

Adding, Moving and Deleting Files

- Add a file

```
$ eg add foo.C
```

- Add directory of files (recursive)

```
$ eg add somedir/
```

- Rename (Move) a file

```
$ eg mv foo.C bar.C
```

- Remove a file

```
$ rm foo.C
```

OR

```
$ eg rm foo.C
```

Adding, Moving and Deleting Files

- Add a file

```
$ eg add foo.C
```

- Add directory of files (recursive)

```
$ eg add somedir/
```

- Rename (Move) a file

```
$ eg mv foo.C bar.C
```

- Remove a file

```
$ rm foo.C
```

OR

```
$ eg rm foo.C
```

Adding, Moving and Deleting Files

- Add a file

```
$ eg add foo.C
```

- Add directory of files (recursive)

```
$ eg add somedir/
```

- Rename (Move) a file

```
$ eg mv foo.C bar.C
```

- Remove a file

```
$ rm foo.C
```

OR

```
$ eg rm foo.C
```

Adding, Moving and Deleting Files

- Add a file

```
$ eg add foo.C
```

- Add directory of files (recursive)

```
$ eg add somedir/
```

- Rename (Move) a file

```
$ eg mv foo.C bar.C
```

- Remove a file

```
$ rm foo.C
```

OR

```
$ eg rm foo.C
```

Reverting File Modifications — `revert`

eg `revert [--since COMMIT] FILES-OR-DIRECTORIES`

- *COMMIT* — revision specifier, defaulting to last commit

Examples:

```
$ eg revert packages/PyTrilinos/CMakeLists.txt
```

```
$ eg revert --since origin/master packages/WebTrilinos
```

Note: This replaces the `rm <filename>; cvs update <filename>` sequence used in CVS.

Reverting File Modifications — `revert`

eg `revert [--since COMMIT] FILES-OR-DIRECTORIES`

- *COMMIT* — revision specifier, defaulting to last commit

Examples:

```
$ eg revert packages/PyTrilinos/CMakeLists.txt
```

```
$ eg revert --since origin/master packages/WebTrilinos
```

Note: This replaces the `rm <filename>; cvs update <filename>` sequence used in CVS.

Reverting File Modifications — `revert`

eg `revert [--since COMMIT] FILES-OR-DIRECTORIES`

- *COMMIT* — revision specifier, defaulting to last commit

Examples:

```
$ eg revert packages/PyTrilinos/CMakeLists.txt
```

```
$ eg revert --since origin/master packages/WebTrilinos
```

Note: This replaces the `rm <filename>; cvs update <filename>` sequence used in CVS.

\$ eg commit

- ***Follow the Git Convention***
 - Short, one-line summary
 - blank line
 - detailed message
- Several git commands work best with this format
- No templates, No test vouchers

```
Slightly improved Tpetra::CrsGraph import/export support.
```

```
Previously the Tpetra::CrsGraph methods which support import/export operations insisted that neither the source nor destination graph had been fillComplete'd. Now that has been slightly relaxed such that it is ok if the source graph has been fillComplete'd, but the destination graph must still be in the pre-fillComplete state.
```

\$ eg commit

- ***Follow the Git Convention***

- Short, one-line summary
- blank line
- detailed message

- Several git commands work best with this format

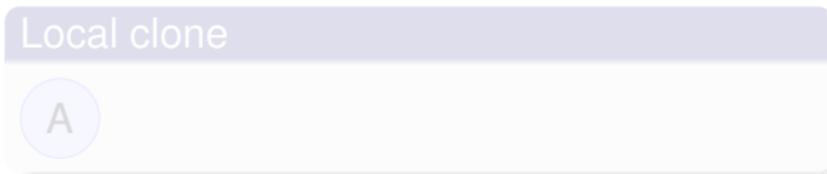
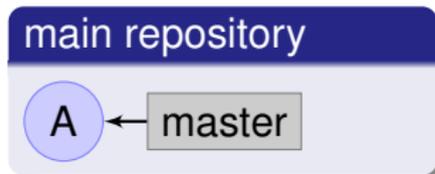
- No templates, No test vouchers

```
Slightly improved Tpetra::CrsGraph import/export support.
```

```
Previously the Tpetra::CrsGraph methods which support  
import/export operations insisted that neither the source nor  
destination graph had been fillComplete'd. Now that has been  
slightly relaxed such that it is ok if the source graph has been  
fillComplete'd, but the destination graph must still be in the  
pre-fillComplete state.
```

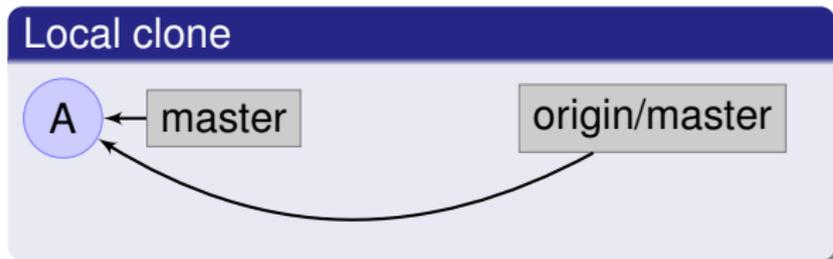
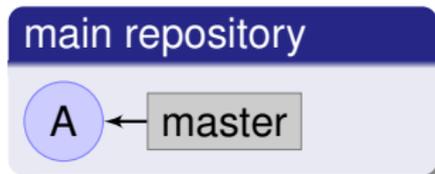
Recording Changes Locally

Initial State:



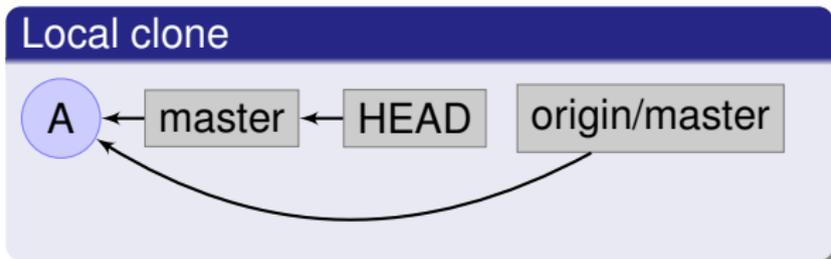
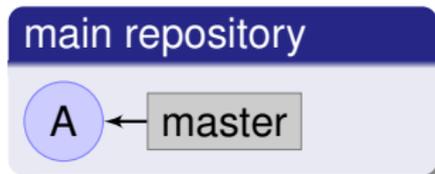
Recording Changes Locally

After cloning:



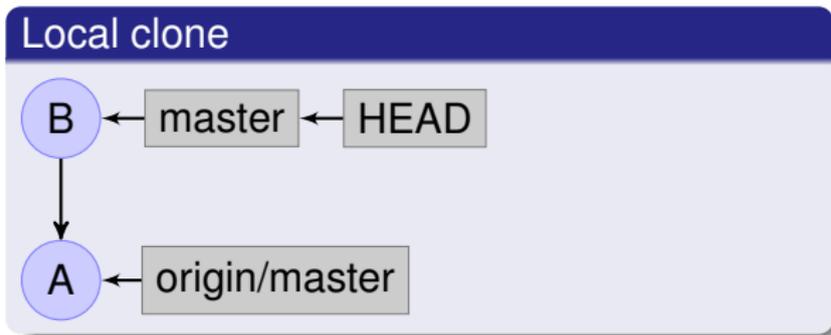
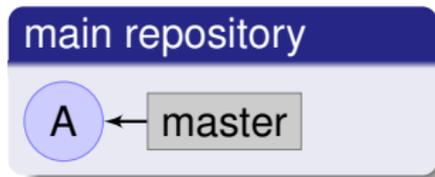
Recording Changes Locally

After cloning:



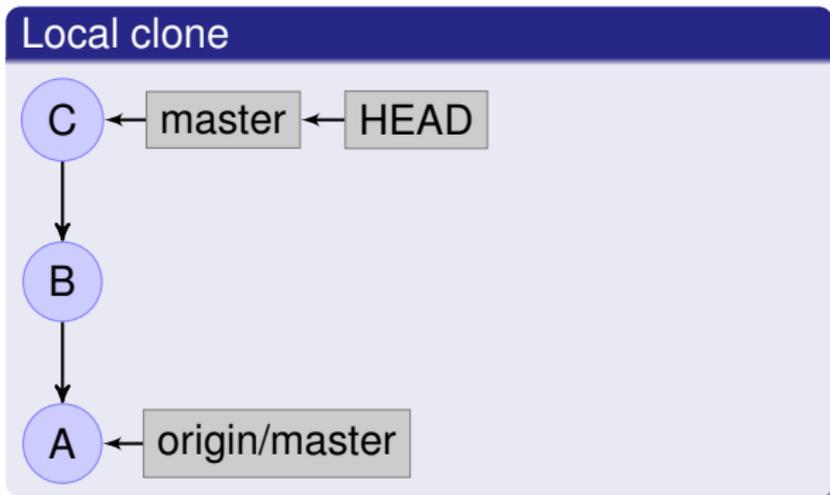
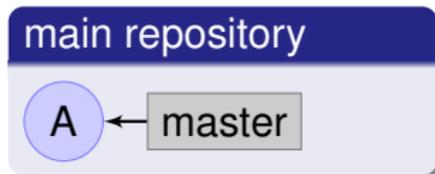
Recording Changes Locally

After making some changes & committing:



Recording Changes Locally

After making more changes & committing again:



Unlike cvs, the status command is brief, fast, and useful. You should get into the habit of running it.

```
$ eg status
```

```
(On branch master)
```

```
(Your branch is ahead of 'origin/master' by 1 commit.)
```

```
Changed but not updated ("unstaged"):
```

```
    modified:   README
```

```
    modified:   packages/aztecoo/src/AztecOO.cpp
```

```
    deleted:    packages/epetra/src/Epetra_Vector.h
```

```
Unknown files:
```

```
    notes.txt
```

`eg diff [options] FROM TO -- PATHS`

- *FROM* - revision specifier, defaulting to HEAD
- *TO* - revision specifier, or the working copy if not specified

Examples:

See the changes between HEAD and the working copy:

```
$ eg diff
```

See the changes since origin/master in the working copy:

```
$ eg diff origin/master
```

See the changes between master~1 and master:

```
$ eg diff master~1 master
```

See the changes to epetra since master~2:

```
$ eg diff master~2 -- packages/epetra
```

Diff has various forms of high level statistics:

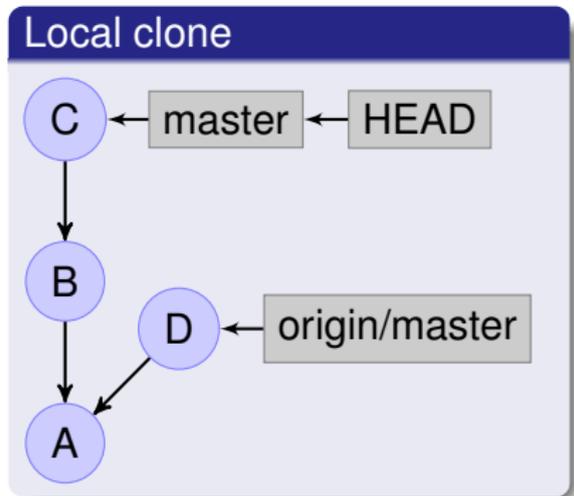
- eg diff --name-only *# Just list the names of the changed files*
- eg diff --name-status *# List files that changed and the type of change*
- eg diff --stat *# List files that changed and lines added and removed*
- eg diff --dirstat *# List directories by percentage of line changes*
- eg diff --shortstat *# List the overall line change count*

Diff has various forms of high level statistics:

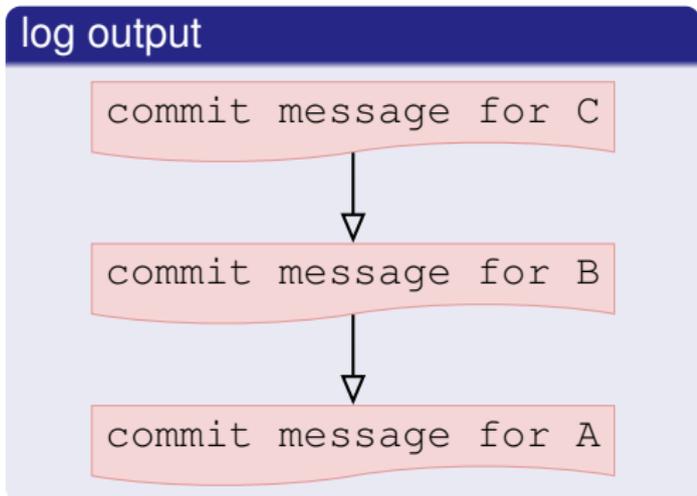
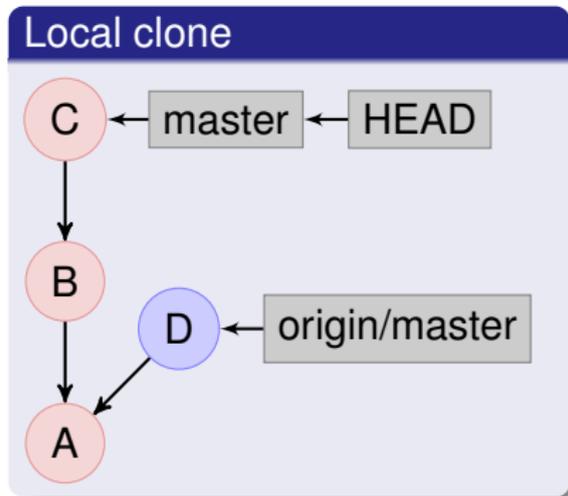
- eg `diff --name-only` *# Just list the names of the changed files*
- eg `diff --name-status` *# List files that changed and the type of change*
- eg `diff --stat` *# List files that changed and lines added and removed*
- eg `diff --dirstat` *# List directories by percentage of line changes*
- eg `diff --shortstat` *# List the overall line change count*

Sidenote: eg `log` also accepts these same arguments, plus `-p` (for showing patches with the commit message), in addition to having all the same search capabilities that `gitk` has.

Initial state after `eg fetch`

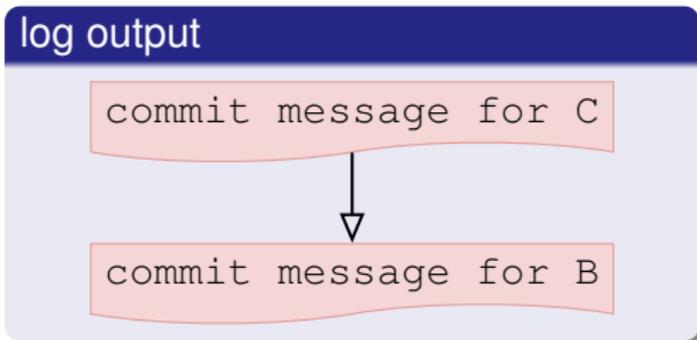
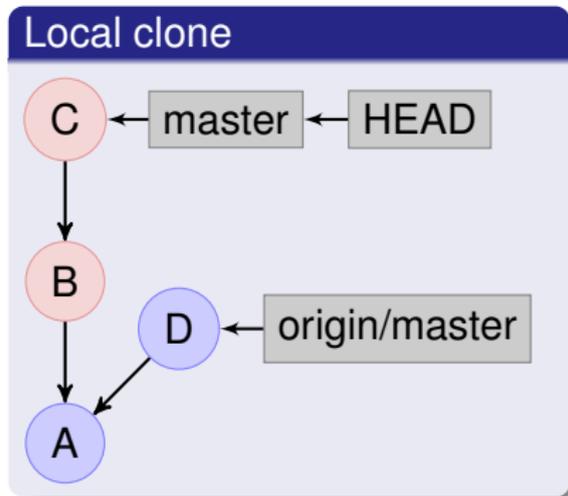


\$ eg log



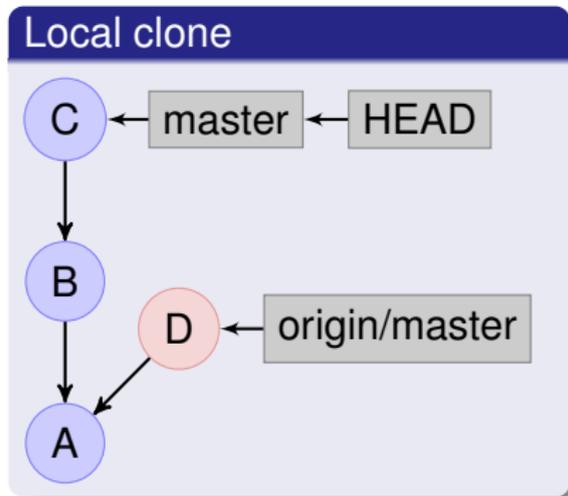
Shows all commits in **master**

```
$ eg log origin/master..master
```



Shows all commits in **master** that are not in **origin/master**

```
$ eg log master..origin/master
```

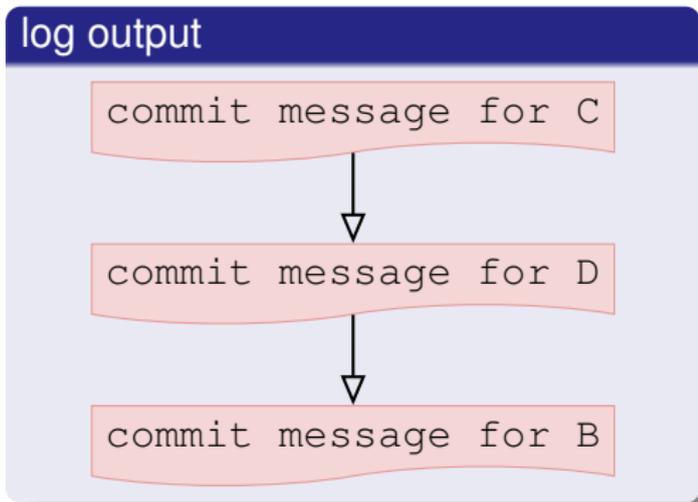
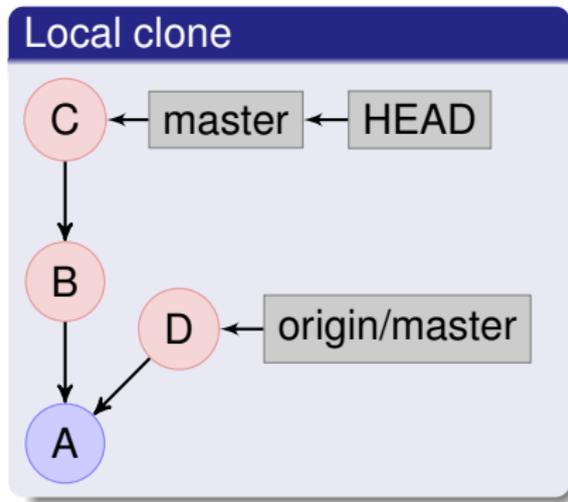


log output

```
commit message for D
```

Shows all commits in **origin/master** that are not in **master**

\$ **eg** `log origin/master...master`

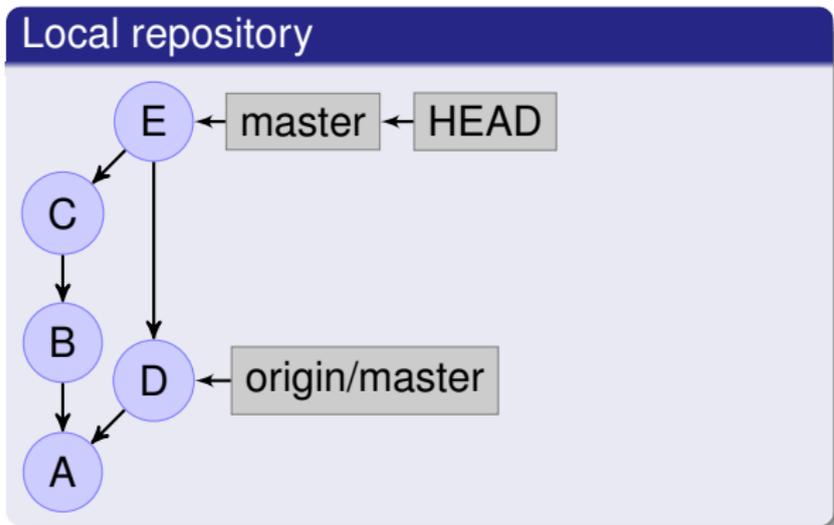


Shows all commits that are in exactly one of **origin/master** and **master**

(log output is ordered by date)

Squashing Commits together

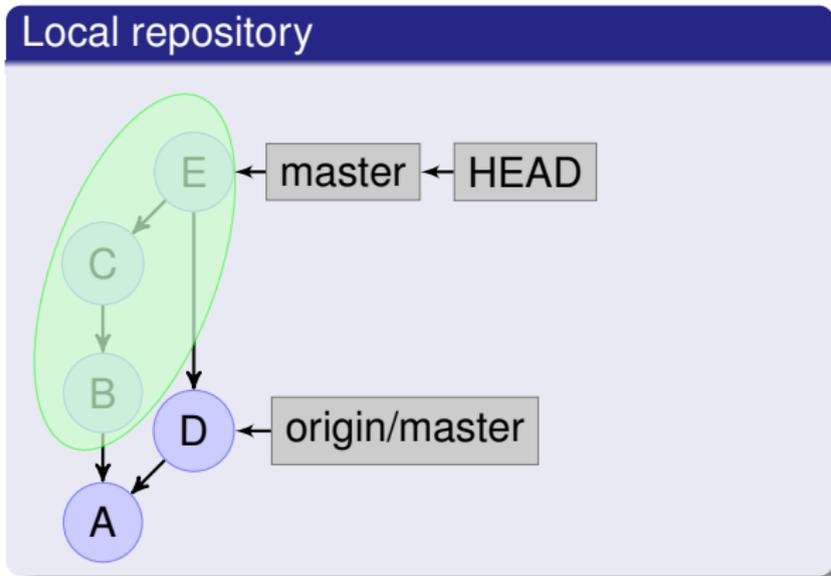
Commit early, commit often, may result in lots of “bad” commits:



You can use `eg rebase -i [--since origin/master]` to selectively combine, reorder, drop, split, or edit commits.

Squashing Commits together

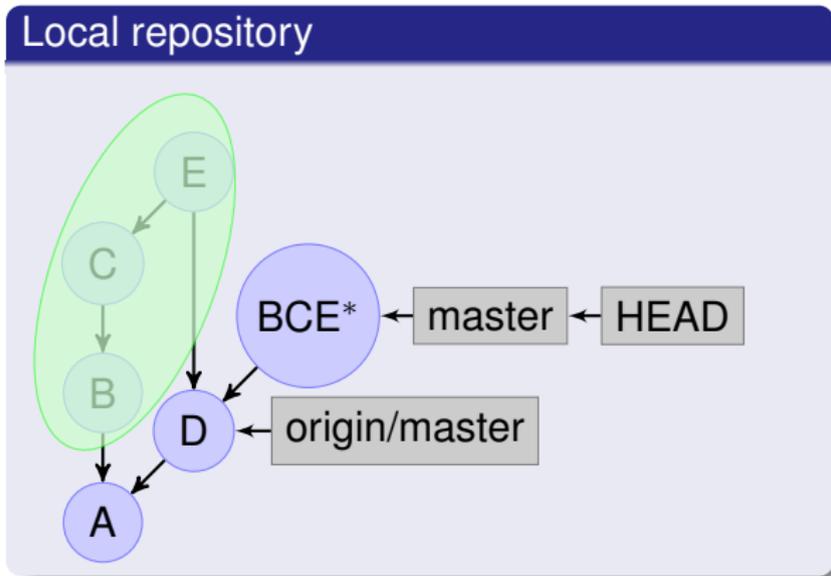
`$ eg squash [--against origin/master]`



You can use `eg rebase -i [--since origin/master]` to selectively combine, reorder, drop, split, or edit commits.

Squashing Commits together

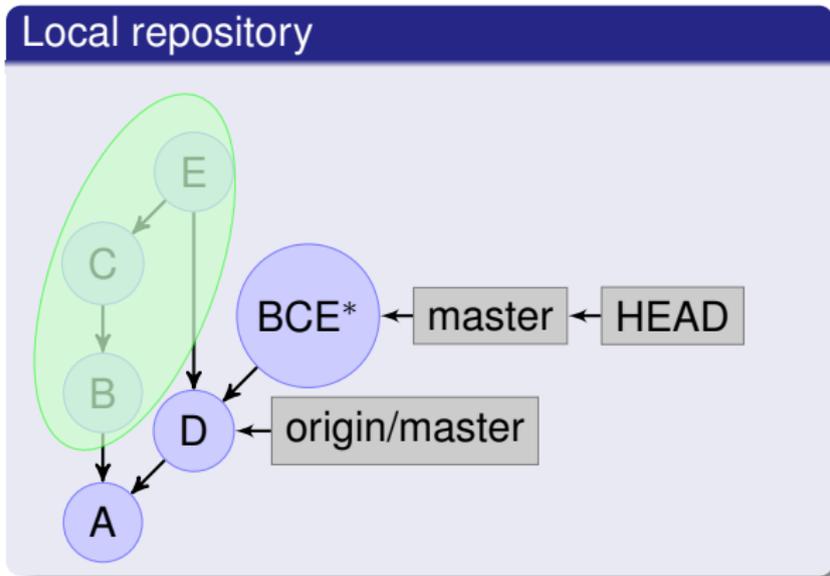
`$ eg squash [--against origin/master]`



You can use `eg rebase -i [--since origin/master]` to selectively combine, reorder, drop, split, or edit commits.

Squashing Commits together

`$ eg squash [--against origin/master]`



You can use `eg rebase -i [--since origin/master]` to selectively combine, reorder, drop, split, or edit commits.

Collaboration – Pulling & Pushing Commits

Pulling changes down (“updating”):

```
$ eg pull [repository] [branch]
```

```
Updating 22d0ed8..e12a6c8
```

```
Fast forward
```

```
packages/zoltan/src/CMakeLists.txt      |      1 +
```

```
packages/zoltan/src/order/hsfcOrder.c   |    250 ++++++
```

```
packages/zoltan/src/order/hsfcOrder.h   |      25 ++++
```

```
packages/zoltan/src/order/order.c       |      21 ++-
```

```
4 files changed, 292 insertions(+), 5 deletions(-)
```

Pushing your commits to publish them:

```
$ eg push [repository] [branch]
```

```
Counting objects: 5, done.
```

```
Delta compression using 8 threads.
```

```
Compressing objects: 100% (3/3), done.
```

```
Writing objects: 100% (3/3), 418 bytes, done.
```

```
Total 3 (delta 2), reused 0 (delta 0)
```

```
To software.sandia.gov:/space/git/temp/Trilinos
```

```
350cc6e..0583c4e  master -> master
```

- 1 Preliminaries
- 2 When Things Go Well: The Basics
- 3 When Things Go Wrong: Common Error Messages
 - “New unknown files”
 - “You have both staged and unstaged changes present”
 - “Entry <filename> not uptodate. Cannot merge.”
 - “Non-fast forward update”
 - Extra Merge Commit(s)
 - You “pull[ed] without telling me which branch to merge with”
 - Merge Conflicts



If you see:

```
$ eg commit
```

```
Aborting: You have new unknown files present and it is not clear
whether they should be committed. Run 'eg help commit' for details.
New unknown files:
  random.C
```

then eg is warning you that you may have forgotten to **add** a new file (in this case `random.C`) that you created.

If you:

- Want the file(s) included: use **eg add**
- Want to commit without the file(s): Pass the **-b** (or **--bypass-unknown-check**) flag to **eg commit**. Using this flag also prevents eg from warning you about these new files in subsequent commits.

“both staged and unstaged changes”

If you see:

```
$ eg commit
```

```
Aborting: It is not clear which changes should be committed; you have both staged (explicitly marked as ready for commit) changes and unstaged changes present. Run 'eg help commit' for details.
```

then **eg** is warning you that you have marked some files as ready for commit (with **add** or **stage**) but not others.

If you:

- Want to commit all files you changed: pass the **-a** flag to **eg commit**.
- Want to commit just the staged changes: pass the **--staged** flag to **eg commit**.

you may want to run **eg status** to see which changes are staged.

“Entry <filename>...Cannot merge.”

If you see (a minor variation of):

```
$ eg pull  
Updating 22a7936..8d7e7a1  
error: Entry 'packages/tifpack/src/Tifpack_IlukGraph.hpp' not uptodate.  
Cannot merge.
```

then eg is warning you that you have uncommitted changes that conflict with changes someone else has made, and proceeding would not be reversible.

Either:

- Commit your changes first (**eg commit**), then repeat the pull
- Stash your changes away: run **eg stash [save <stash description>]**, then repeat the pull, then reapply your changes (**eg stash pop [<stash description>]**).

If you see:

```
$ eg push
```

```
To software.sandia.gov:/space/git/temp/Trilinos
```

```
! [rejected]          master -> master (non-fast forward)
```

```
error: failed to push some refs to 'software:/space/git/temp/Trilinos'
```

then git is using weird language to tell you that your project is not up-to-date. You'll need to first pull down the commits you don't have yet.

If you see:

```
$ eg push
```

```
...
```

```
The commit:
```

```
[commit message information here]
```

Looks like it was produced by typing 'eg pull' without the --rebase option when you had local changes. Running 'eg pull --rebase' now will fix the problem. Then please try, 'eg push' again. Please see:

```
https://software.sandia.gov/developer/git/hip/ExtraMergeCommits.html
```

```
---
```

```
error: hooks/pre-receive exited with error code 1
```

```
To software.sandia.gov:/space/git/temp/Trilinos
```

```
! [remote rejected] master -> master (pre-receive hook declined)
```

```
error: failed to push some refs to 'software:/space/git/temp/Trilinos'
```

then you have a commit we consider 'useless' that you should remove before pushing.

You can run **eg squash** to get rid of these commits (and combine all your local unpushed commits into one), or run the command suggested by the error message. You can follow the suggested link to learn more about this issue.

“pull[ed] without telling me which branch”

If you see:

```
$ eg pull
```

```
You asked me to pull without telling me which branch you  
want to merge with, and 'branch.master.merge' in  
your configuration file does not tell me either.
```

```
...
```

(the full error message takes a full screen) then git does not know which branch you are trying to track, probably because you created a new branch without specifying a remote branch as a starting point.

Simply tell git which branch you want the active branch to track:

```
$ eg track origin/master
```

If during a pull (or merge or rebase) you see:

```
Auto-merging packages/amesos/src/Amesos.cpp
CONFLICT (content): Merge conflict in packages/amesos/src/Amesos.cpp
```

then you changed the same file someone else did and have conflicts.

You can either

- Go back (only safe **if** you committed or stashed before pulling): **eg reset --working-copy ORIG_HEAD.**
- or
- Edit files to remove conflict markers
- Tell Git that you have resolved the conflicts: **eg stage <filename>**
- Continue/complete the operation (for merges: **eg commit**; for rebases: **eg rebase --continue**)

NOTE: You can run **eg status** to see which files are unmerged (i.e. have conflicts), see which type of operation you are in the middle of, and get a pointer to help for resolving conflicts.

```
$ eg status
```

```
(On branch master)
```

```
(Your branch and 'origin/master' have diverged,)
```

```
(and have 1 and 1 different commit(s) each, respectively.)
```

```
( YOU ARE IN THE MIDDLE OF A MERGE; RUN 'eg help topic middle-of-merge' FOR MORE INFO. )
```

```
Changes ready to be committed ("staged"):
```

```
new file: packages/tifpack/src/Tifpack_CreateOverlapGraph.hpp
```

```
modified: packages/tifpack/src/Tifpack_OverlapGraph.hpp
```

```
modified: packages/tifpack/test/unit_tests/CMakeLists.txt
```

```
new file: packages/tifpack/test/unit_tests/Tifpack_UnitTestCreateOverlapGraph.cpp
```

```
new file: packages/tifpack/test/unit_tests/Tifpack_UnitTestHelpers.hpp
```

```
new file: packages/tifpack/test/unit_tests/Tifpack_UnitTestOverlapGraph.cpp
```

```
modified: packages/tifpack/test/unit_tests/Tifpack_UnitTestTemplate.cpp
```

```
Changed but not updated ("unstaged"):
```

```
unmerged: packages/tifpack/src/Tifpack_IlukGraph.hpp
```

```
( YOU ARE IN THE MIDDLE OF A MERGE; RUN 'eg help topic middle-of-merge' FOR MORE INFO. )
```

CVS Usage

Git Usage

\$ **cvs checkout -d**

:ext:USER@MACHINE:/PATH REPOSITORY → \$ **eg clone [USER@]MACHINE:/PATH**

\$ **cvs commit**

→ \$ **eg commit**
\$ **eg push**

\$ **cvs [-q] update [-dP]**

→ \$ **eg commit # To allow backout**
\$ **eg pull**

\$ **cvs update**
\$ **cvs commit**

→ \$ **eg commit # To allow backout**
\$ **eg pull**
\$ **eg squash # To cleanup**
\$ **eg push**

\$ **rm FILE**
\$ **cvs update FILE**

→ \$ **eg revert FILE**

\$ **cvs -H COMMAND**

→ \$ **eg help COMMAND**

\$ **cvs status**

→ \$ **eg status**

\$ **cvs -nq update**

→ \$ **eg status**

\$ **cvs diff -u**

→ \$ **eg diff**

\$ **cvs add FILE**

→ \$ **eg add FILE-OR-DIRECTORY**

\$ **cvs rm [-f] FILE**

→ \$ **eg rm FILE-OR-DIRECTORY**

You're kidding, right?

→ \$ **eg mv FILE-OR-DIR NEWPATH**

\$ **cvs log FILE**

→ \$ **eg log [FILE-OR-DIR]**
