



Git Tips and Tricks

Elijah Newren, Pat Notz

Sandia National Laboratories

March 17, 2009

- 1 History Layout
 - Global vs. Local
 - Sharing Commits
 - Pull and autosesetuprebase
 - Revision Names
- 2 Getting Information from git
- 3 Cleaning Up
- 4 Multiple Branches, Multiple Repositories
- 5 Tools and Miscellaneous Tips

Branches and Remote-tracking Branches

Initial State:

sierra-trac repository

A ← master

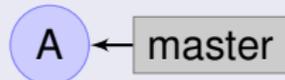
Local clone

A

Branches and Remote-tracking Branches

After cloning:

sierra-trac repository



Local clone



Branches and Remote-tracking Branches

After cloning:

sierra-trac repository



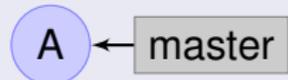
Local clone



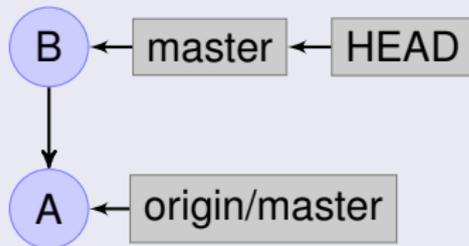
Branches and Remote-tracking Branches

After committing:

sierra-trac repository



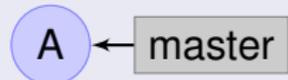
Local clone



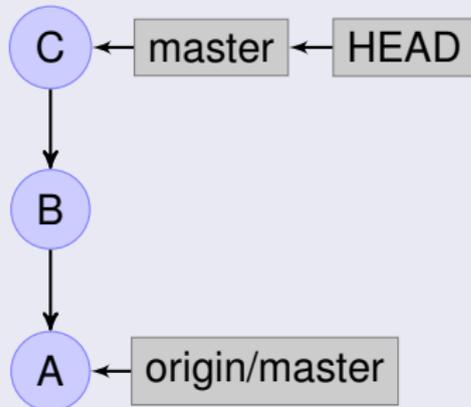
Branches and Remote-tracking Branches

After committing again:

sierra-trac repository



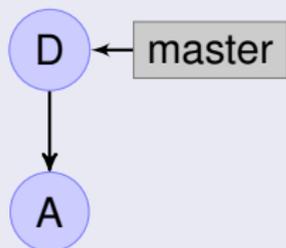
Local clone



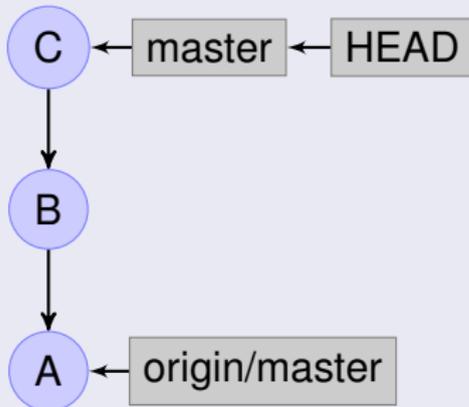
Branches and Remote-tracking Branches

After someone pushes to sierra-trac:

sierra-trac repository

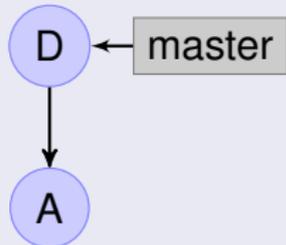


Local clone

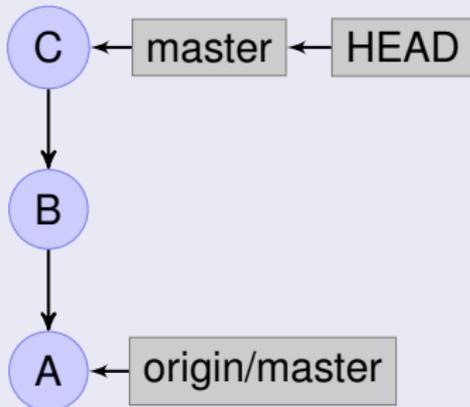


Initial state:

sierra-trac repository

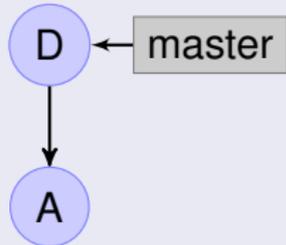


Local clone

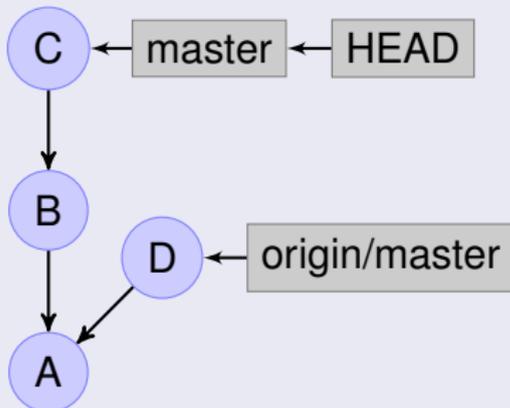


\$ eg fetch

sierra-trac repository

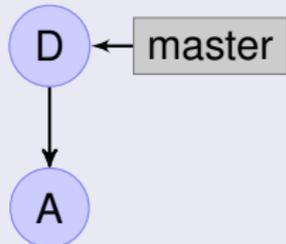


Local clone

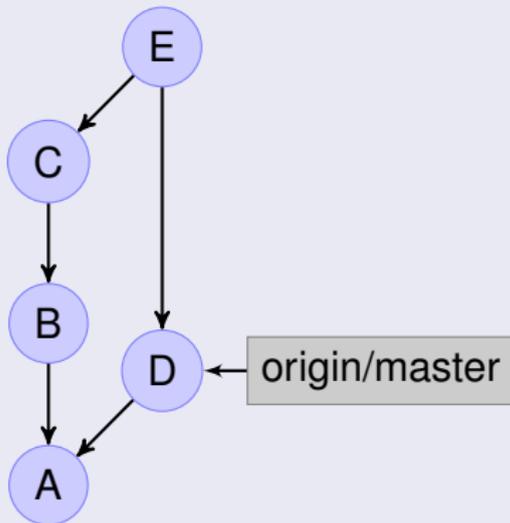


\$ eg merge origin/master

sierra-trac repository

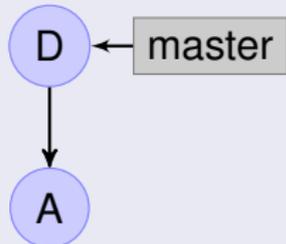


Local clone

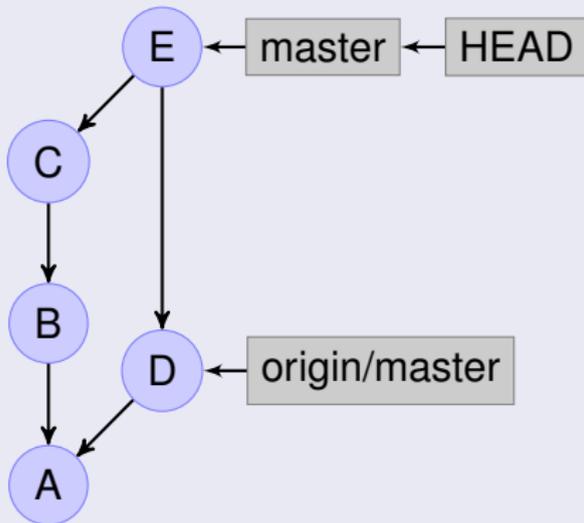


\$ eg merge origin/master

sierra-trac repository

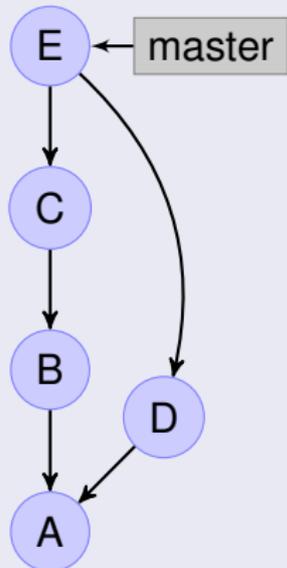


Local clone

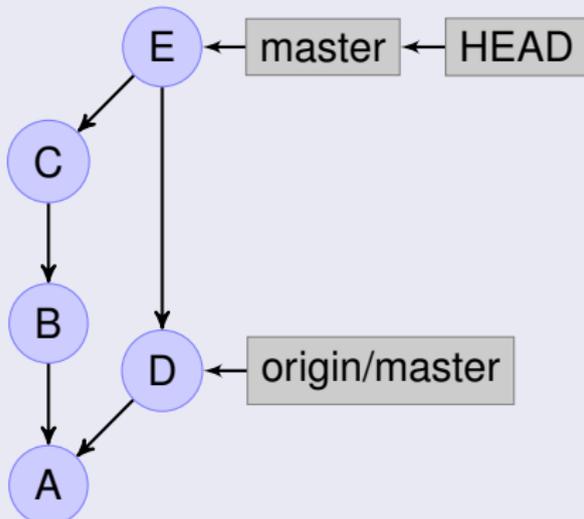


\$ eg push

sierra-trac repository

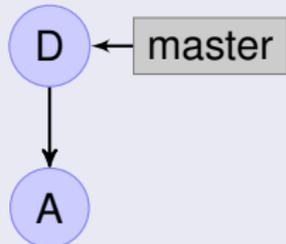


Local clone

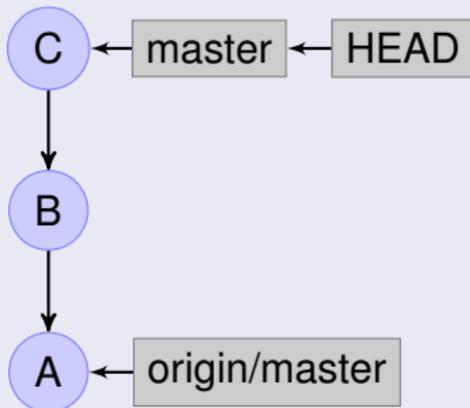


Initial state:

sierra-trac repository

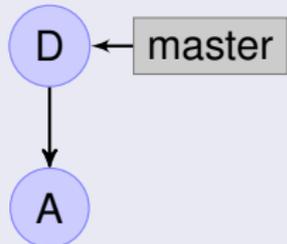


Local clone

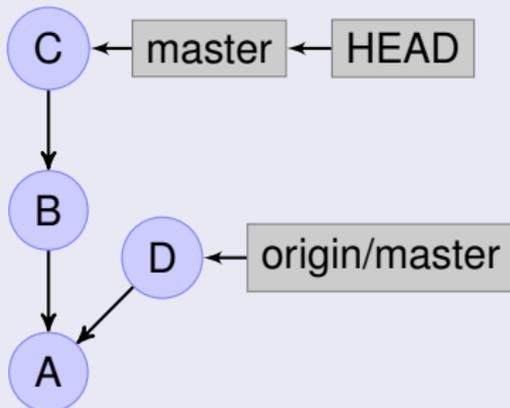


\$ eg fetch

sierra-trac repository

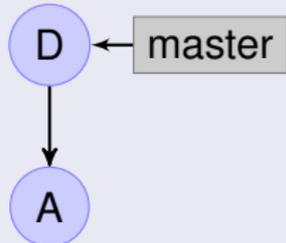


Local clone

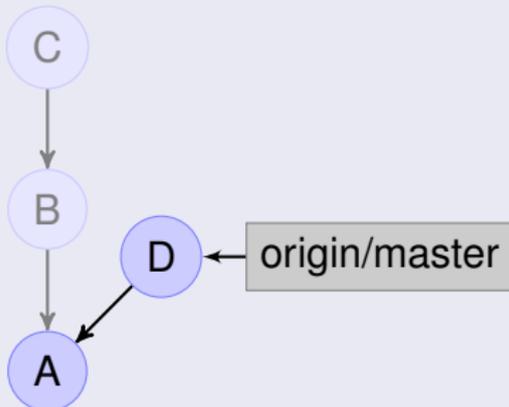


\$ eg rebase --against origin/master

sierra-trac repository

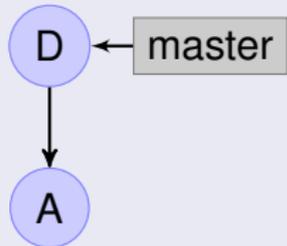


Local clone

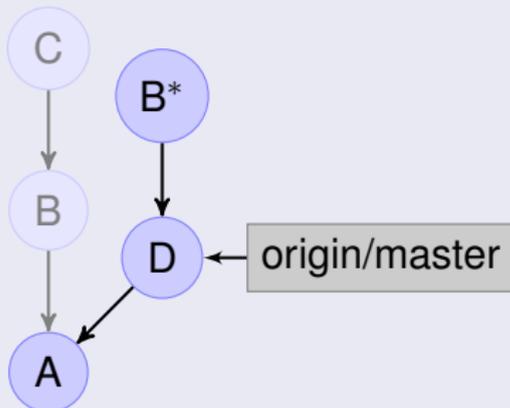


\$ eg rebase --against origin/master

sierra-trac repository

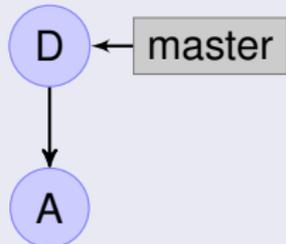


Local clone

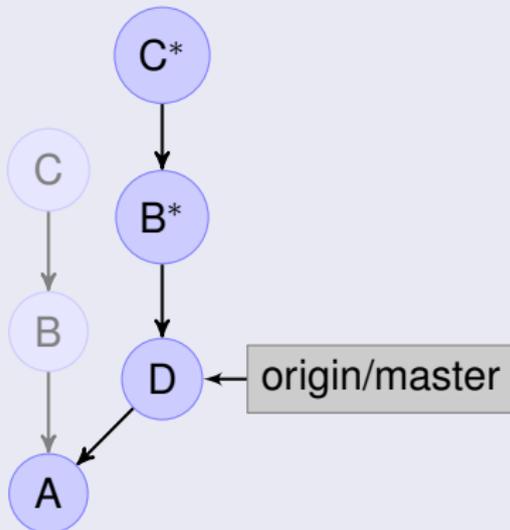


\$ eg rebase --against origin/master

sierra-trac repository

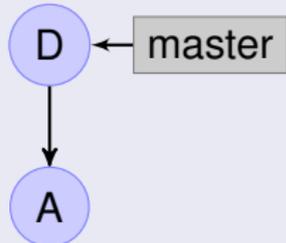


Local clone

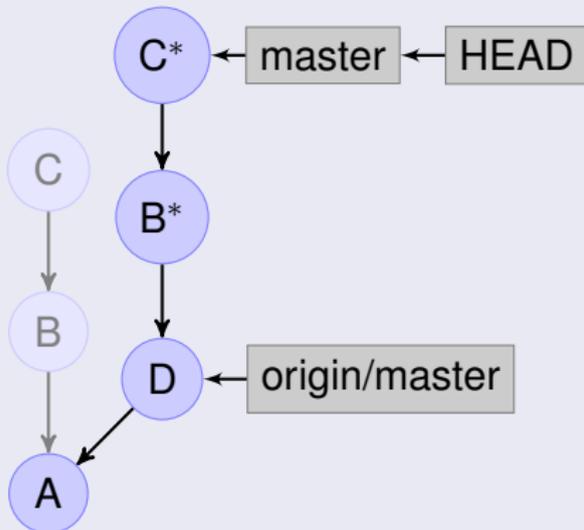


\$ eg rebase --against origin/master

sierra-trac repository

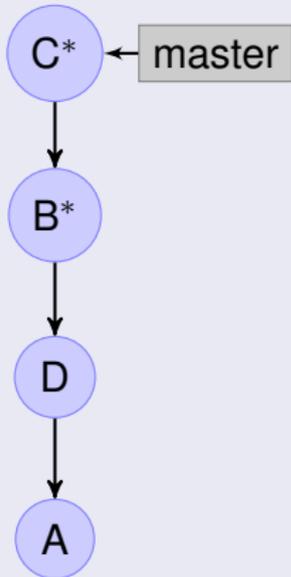


Local clone

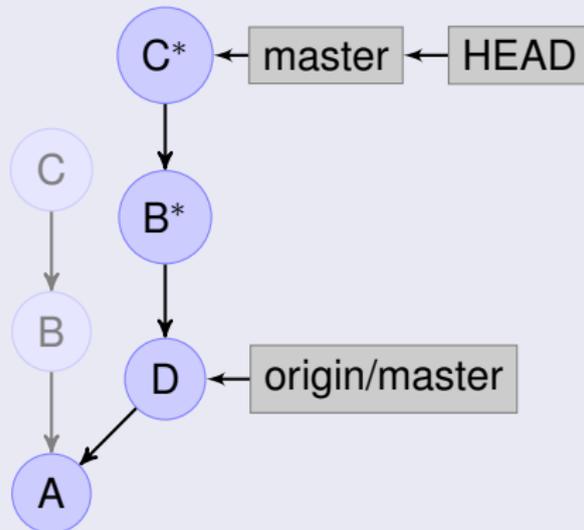


\$ eg push

sierra-trac repository



Local clone





pull and branch.autosetuprebase config setting

By default,

$$\text{eg pull} = \left\{ \begin{array}{l} \text{eg fetch} \\ + \\ \text{eg merge REMOTE-TRACKING-BRANCH} \end{array} \right.$$

If `branch.<current branch>.rebase` is true,

$$\text{eg pull} = \left\{ \begin{array}{l} \text{eg fetch} \\ + \\ \text{eg rebase --against REMOTE-TRACKING-BRANCH} \end{array} \right.$$

`branch.autosetuprebase` headaches:

When a new branch (*<new branch>*) is created, sets

`branch.<new branch>.rebase = true`

and `branch.autosetuprebase` to

pull and branch.autosetuprebase config setting

By default,

$$\text{eg pull} = \left\{ \begin{array}{l} \text{eg fetch} \\ + \\ \text{eg merge REMOTE-TRACKING-BRANCH} \end{array} \right.$$

If `branch.<current branch>.rebase` is true,

$$\text{eg pull} = \left\{ \begin{array}{l} \text{eg fetch} \\ + \\ \text{eg rebase --against REMOTE-TRACKING-BRANCH} \end{array} \right.$$

`branch.autosetuprebase` headaches:

When a new branch (*<new branch>*) is created, sets
branch.<new branch>.rebase = true



pull and branch.autosetuprebase config setting

By default,

$$\text{eg pull} = \left\{ \begin{array}{l} \text{eg fetch} \\ + \\ \text{eg merge REMOTE-TRACKING-BRANCH} \end{array} \right.$$

If `branch.<current branch>.rebase` is true,

$$\text{eg pull} = \left\{ \begin{array}{l} \text{eg fetch} \\ + \\ \text{eg rebase --against REMOTE-TRACKING-BRANCH} \end{array} \right.$$

`branch.autosetuprebase` headaches:

When a new branch (*<new branch>*) is created, sets

branch.<new branch>.rebase = true

...at least it is *supposed* to.



pull and branch.autosetuprebase config setting

By default,

$$\text{eg pull} = \left\{ \begin{array}{l} \text{eg fetch} \\ + \\ \text{eg merge REMOTE-TRACKING-BRANCH} \end{array} \right.$$

If `branch.<current branch>.rebase` is true,

$$\text{eg pull} = \left\{ \begin{array}{l} \text{eg fetch} \\ + \\ \text{eg rebase --against REMOTE-TRACKING-BRANCH} \end{array} \right.$$

`branch.autosetuprebase` headaches:

When a new branch (*<new branch>*) is created, sets

`branch.<new branch>.rebase = true`

..at least it is *supposed* to.



pull and branch.autosetuprebase config setting

By default,

$$\text{eg pull} = \left\{ \begin{array}{l} \text{eg fetch} \\ + \\ \text{eg merge REMOTE-TRACKING-BRANCH} \end{array} \right.$$

If `branch.<current branch>.rebase` is true,

$$\text{eg pull} = \left\{ \begin{array}{l} \text{eg fetch} \\ + \\ \text{eg rebase --against REMOTE-TRACKING-BRANCH} \end{array} \right.$$

`branch.autosetuprebase` headaches:

When a new branch (*<new branch>*) is created, sets

branch.<new branch>.rebase = true

..at least it is *supposed* to.

Suffixes:

- n = nth parent
- $\sim n$ = nth ancestor following 1st parents ($\sim n = ^1^1 \dots ^1$)

Local Repository



Local Commit Names

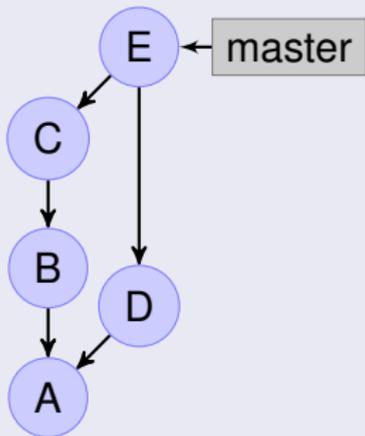
- E = master
- C = $\text{master}^1 = \text{master}\sim 1$
- D = master^2
- B = $\text{master}^1^1 = \text{master}\sim 2$
- A = $\text{master}\sim 3$ or A = $\text{master}^2\sim 1$

Or, use globally unique identifiers shown in `eg log` or `gitk` (e.g. E=a99816d0b6c626befb0234485b4964887faca4d5), or their abbreviation (e.g. E=a99816d0b)

Suffixes:

- n = nth parent
- $\sim n$ = nth ancestor following 1st parents ($\sim n = ^1^1 \dots ^1$)

Local Repository



Local Commit Names

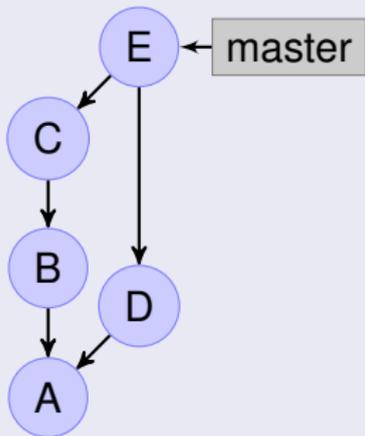
- E = master
- C = $\text{master}^1 = \text{master}\sim 1$
- D = master^2
- B = $\text{master}^1^1 = \text{master}\sim 2$
- A = $\text{master}\sim 3$ or A = $\text{master}^2\sim 1$

Or, use globally unique identifiers shown in `log` or `gitk` (e.g. E=a99816d0b6c626befb0234485b4964887faca4d5), or their abbreviation (e.g. E=a99816d0b)

Suffixes:

- n = nth parent
- $\sim n$ = nth ancestor following 1st parents ($\sim n = ^1^1 \dots ^1$)

Local Repository



Local Commit Names

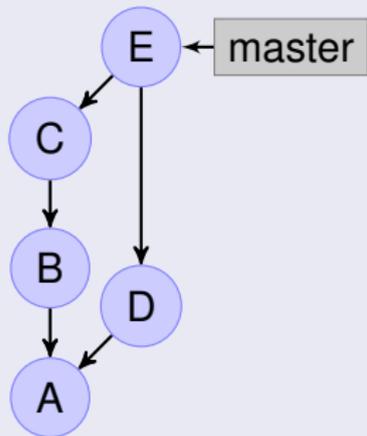
- E = master
- C = $\text{master}^1 = \text{master}\sim 1$
- D = master^2
- B = $\text{master}^1^1 = \text{master}\sim 2$
- A = $\text{master}\sim 3$ or A = $\text{master}^2\sim 1$

Or, use globally unique identifiers shown in `log` or `gitk` (e.g. E=a99816d0b6c626befb0234485b4964887faca4d5), or their abbreviation (e.g. E=a99816d0b)

Suffixes:

- n = nth parent
- $\sim n$ = nth ancestor following 1st parents ($\sim n = ^1^1 \dots ^1$)

Local Repository



Local Commit Names

- E = master
- C = $\text{master}^1 = \text{master}\sim 1$
- D = master^2
- B = $\text{master}^1^1 = \text{master}\sim 2$
- A = $\text{master}\sim 3$ or A = $\text{master}^2\sim 1$

Or, use globally unique identifiers shown in `log` or `gitk` (e.g. E=a99816d0b6c626befb0234485b4964887faca4d5), or their abbreviation (e.g. E=a99816d0b)

1 History Layout

2 Getting Information from git

- Viewing Changes with `gitk`
- Viewing Changes with `eg diff`
- Viewing and Digging for changes with `eg log`
- Searching for text in files with `eg grep`

3 Cleaning Up

4 Multiple Branches, Multiple Repositories

5 Tools and Miscellaneous Tips

Viewing Changes with gitk

\$ gitk &

The screenshot shows the gitk application window titled "gitk: code". The interface includes a menu bar (File, Edit, View), a toolbar, and a main content area. The commit being viewed is SHA1 ID: c645948e000717c0de7c576223a05043525fe047. The commit message is: "Bug fix for contact command block ordering within the Region command block." The commit was authored by @sandia.gov on 2009-03-13 17:48:18. The commit log shows a list of changes with their respective authors and dates. The main content area displays the commit details, including the author, committer, parent, child, and branches. The commit message is displayed in a text area. The commit log shows a list of changes with their respective authors and dates. The main content area displays the commit details, including the author, committer, parent, child, and branches. The commit message is displayed in a text area. The commit log shows a list of changes with their respective authors and dates.

The screenshot shows the "Gitk view definition - criteria for selecting revisions" dialog box. The dialog has a title bar and a close button. It contains several sections for configuring the view criteria. The "View Name" is set to "View 2". There is a checkbox for "Remember this view". The "References (space separated list):" section has a text input field. Below it are checkboxes for "All refs", "All (local) branches", "All tags", and "All remote-tracking branches". The "Commit Info (regular expressions):" section has input fields for "Author:" and "Committer:". There is a "Commit Message:" label and a text input field. A checkbox "Matches all Commit Info criteria" is present. The "Changes to Files:" section has radio buttons for "Fixed String" and "Regular Expression". There is a "Search string:" label and a text input field. The "Commit Dates ('2 weeks ago', '2009-03-17 15:27:38', 'March 17, 2009 15:27:38'):" section has input fields for "Since:" and "Until:". The "Limit and/or skip a number of revisions (positive integer):" section has input fields for "Number to show:" and "Number to skip:". The "Miscellaneous options:" section has checkboxes for "Strictly sort by date", "Mark branch sides", and "Limit to first parent". There is an "Additional arguments to git log:" label and a text input field. The "Enter files and directories to include, one per line:" section has a large text area. The "Command to generate more commits to include:" section has a text input field. At the bottom are "OK", "Apply (F5)", and "Cancel" buttons.

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

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

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

```
# See the changes between HEAD and the working copy  
$ eg diff
```

```
# See the changes between master~1 and the working copy  
$ eg diff master~1
```

```
# See the changes between origin/master and master  
$ eg diff origin/master master
```

```
# See the changes to framework since master~2  
$ eg diff master~2 -- framework
```

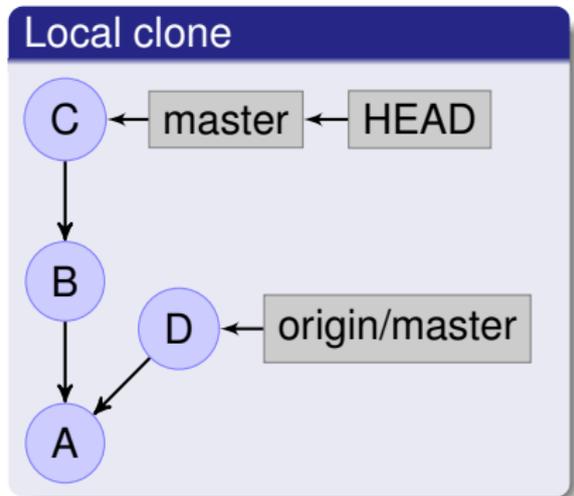


Following Changes with Diff (2/2) — high level stats

Diff has various forms of high level statistics:

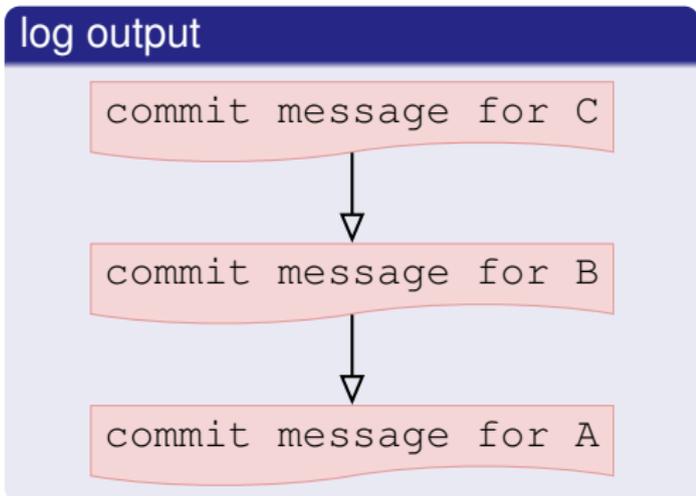
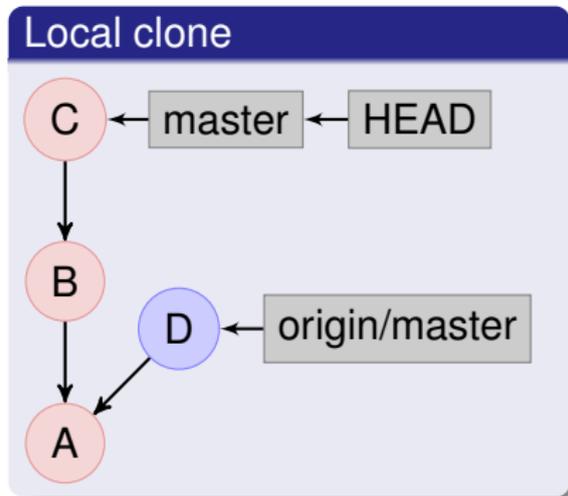
- eg `diff --name-only` *# Just list the names of the changed files*
- eg `diff --name-status` *# List files that changed and the type of change*
- eg `diff --stat` *# List files that changed and lines added and removed*
- eg `diff --dirstat` *# List directories by percentage of line changes*
- eg `diff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

Initial state after `eg fetch`



Viewing commits with `eg log`

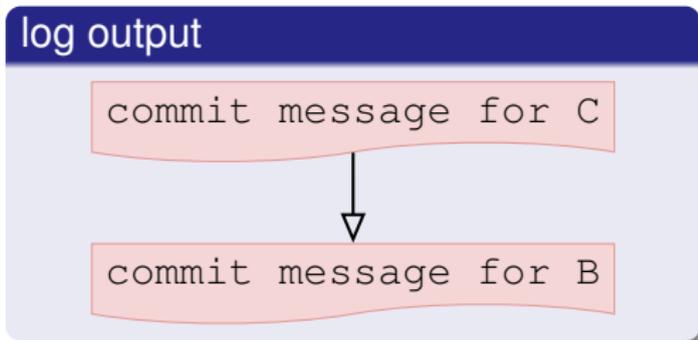
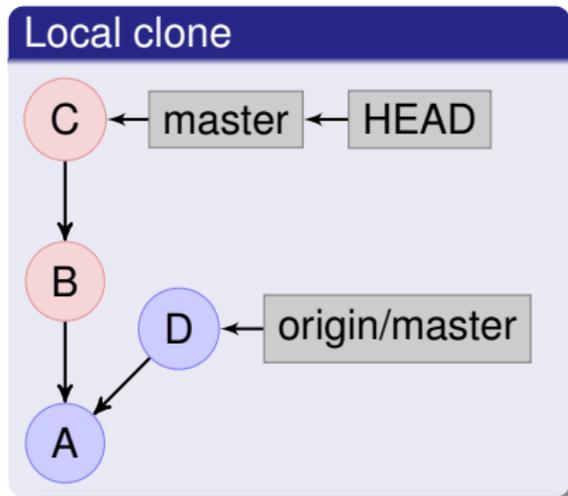
`$ eg log`



Shows all commits in **master**

Viewing commits with `eg log`

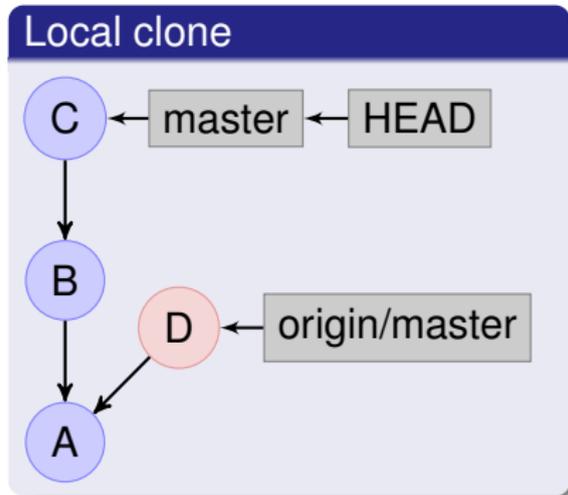
```
$ eg log origin/master..master
```



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

Viewing commits with `eg log`

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



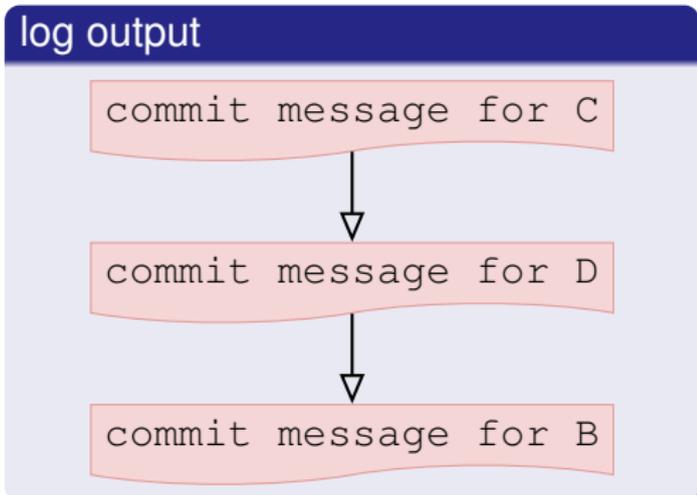
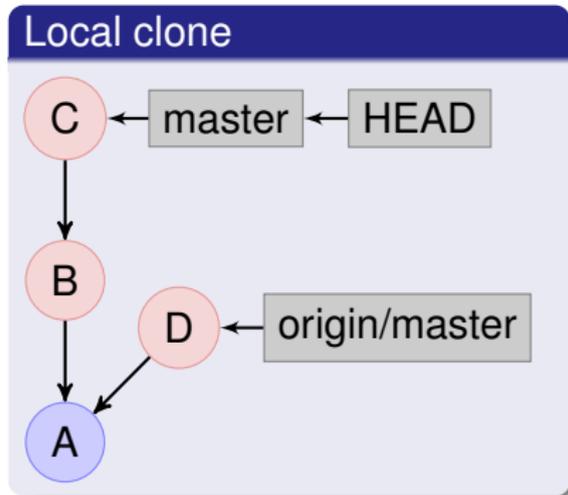
log output

```
commit message for D
```

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

Viewing commits with `eg log`

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



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

(log output is ordered by date)

- Commit Messages + Changes

`eg log` shows commit information only by default, but can also show the changes made in a commit (in diff/patch format):

```
$ eg log -p ...
```

- High level change statistics

You can also use the same high level statistic flags as with `eg diff`, instead of (or in addition to) passing the `-p` flag to `eg log`: `--name-only`, `--name-status`, `--stat`, `--dirstat`, and `--shortstat`.

- Data Mining

All “View” options from `gitk` (and some extras) can be accessed from `eg log`; for example:

- `--author=author regex`
- `--grep=commit message regex`
- `--since=date string`
- `--until=date string`
- `-Stext to find in a file change`

- Commit Messages + Changes

`eg log` shows commit information only by default, but can also show the changes made in a commit (in diff/patch format):

```
$ eg log -p ...
```

- High level change statistics

You can also use the same high level statistic flags as with `eg diff`, instead of (or in addition to) passing the `-p` flag to `eg log`: `--name-only`, `--name-status`, `--stat`, `--dirstat`, and `--shortstat`.

- Data Mining

All “View” options from `gitk` (and some extras) can be accessed from `eg log`; for example:

- `--author=author regex`
- `--grep=commit message regex`
- `--since=date string`
- `--until=date string`
- `-Stext to find in a file change`

- Commit Messages + Changes

`eg log` shows commit information only by default, but can also show the changes made in a commit (in diff/patch format):

```
$ eg log -p ...
```

- High level change statistics

You can also use the same high level statistic flags as with `eg diff`, instead of (or in addition to) passing the `-p` flag to `eg log`: `--name-only`, `--name-status`, `--stat`, `--dirstat`, and `--shortstat`.

- Data Mining

All “View” options from `gitk` (and some extras) can be accessed from `eg log`; for example:

- `--author=author regex`
- `--grep=commit message regex`
- `--since=date string`
- `--until=date string`
- `-Stext to find in a file change`



Getting less info out of the log

eg log can provide a lot of information. Sometimes, you want less — just the one-line summaries, grouped by author. In such cases, use shortlog. An example:

```
$ eg shortlog Sierra_4_10_branch..master
```



Searching for text in files with `eg` `grep`

You can look for string or regex matches in currently checked out files

\$ `eg grep PATTERN`

This is nicer than standard `grep` in that it skips executables, test results, and other untracked files.

You can also search in files of a different revision

\$ `eg grep PATTERN REVISION`

and/or search in a specific list of files and directories:

\$ `eg grep PATTERN REVISION -- DIRECTORY`



Searching for text in files with `eg` `grep`

You can look for string or regex matches in currently checked out files

```
$ eg grep PATTERN
```

This is nicer than standard `grep` in that it skips executables, test results, and other untracked files.

You can also search in files of a different revision

```
$ eg grep PATTERN REVISION
```

and/or search in a specific list of files and directories:

```
$ eg grep PATTERN REVISION -- DIRECTORY
```



Searching for text in files with `eg` `grep`

You can look for string or regex matches in currently checked out files

\$ `eg grep PATTERN`

This is nicer than standard `grep` in that it skips executables, test results, and other untracked files.

You can also search in files of a different revision

\$ `eg grep PATTERN REVISION`

and/or search in a specific list of files and directories:

\$ `eg grep PATTERN REVISION -- DIRECTORY`



Searching for text in files with `eg` `grep`

You can look for string or regex matches in currently checked out files

\$ `eg grep PATTERN`

This is nicer than standard `grep` in that it skips executables, test results, and other untracked files.

You can also search in files of a different revision

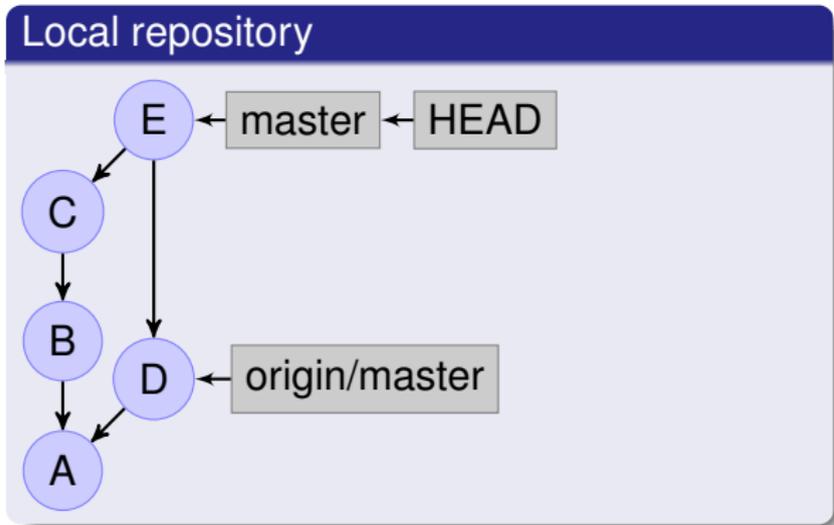
\$ `eg grep PATTERN REVISION`

and/or search in a specific list of files and directories:

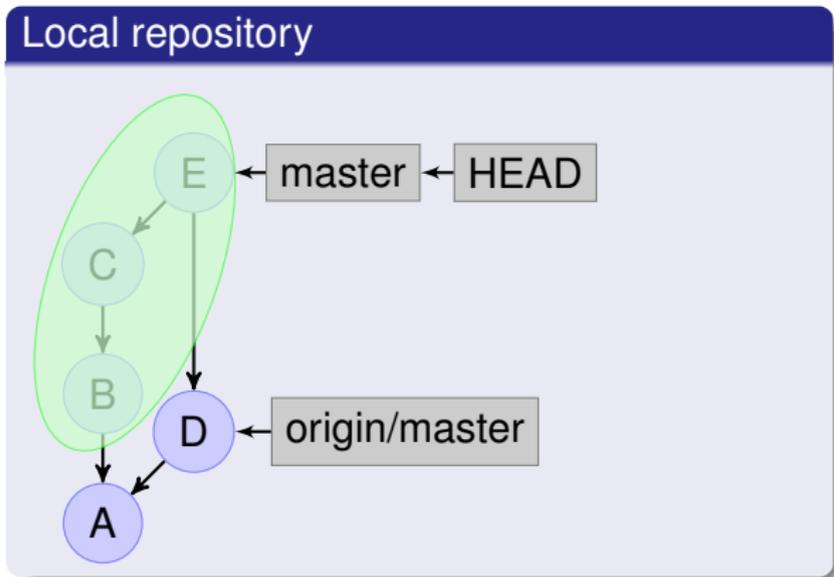
\$ `eg grep PATTERN REVISION -- DIRECTORY`

- 1 History Layout
- 2 Getting Information from git
- 3 Cleaning Up**
 - Rebasing (replaying) sequences of commits
 - Squashing commits together
 - Amending commits
 - Undoing and redoing commits
- 4 Multiple Branches, Multiple Repositories
- 5 Tools and Miscellaneous Tips

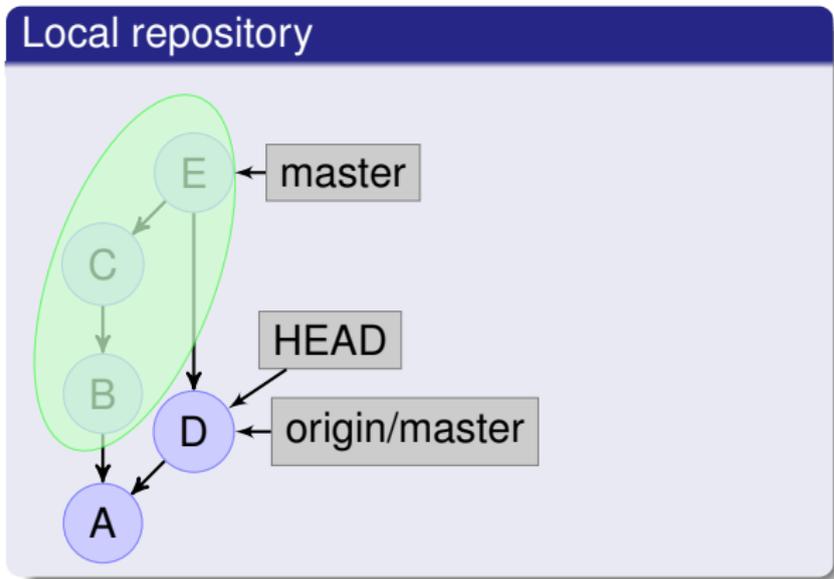
Starting point:



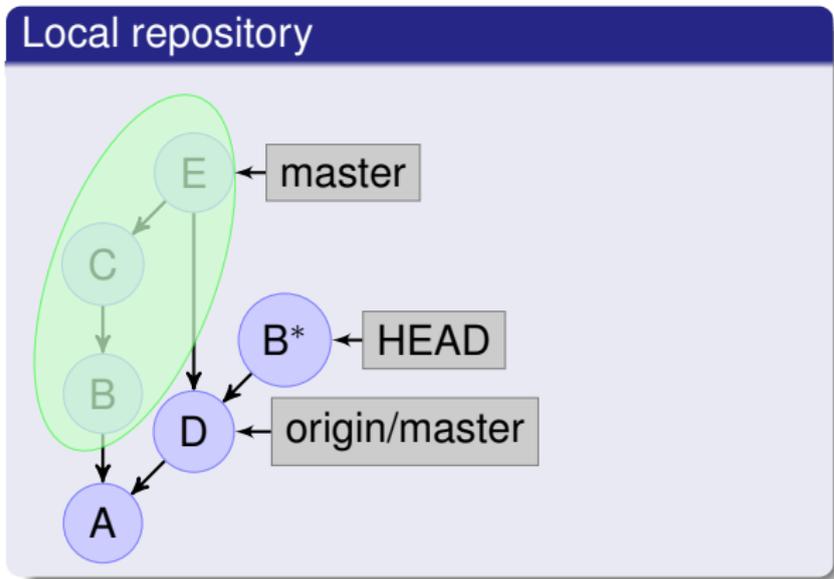
\$ eg rebase --against origin/master



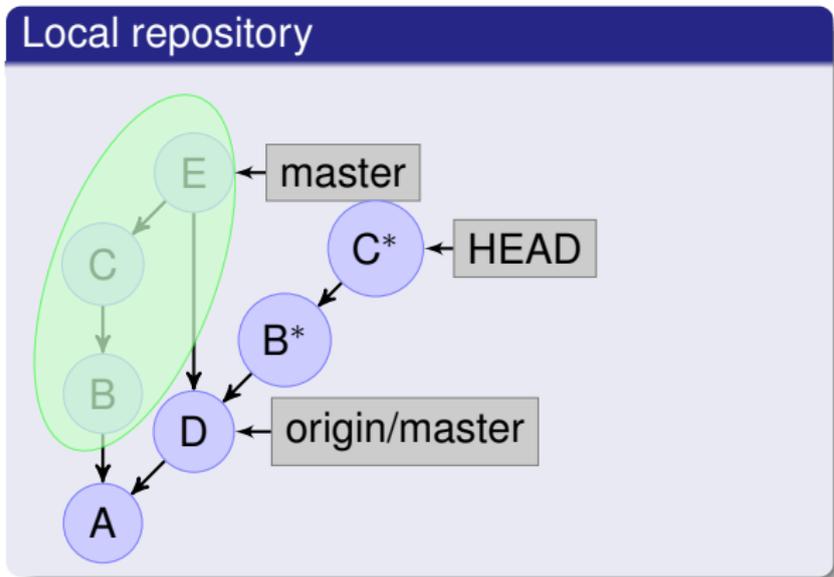
\$ eg rebase --against origin/master



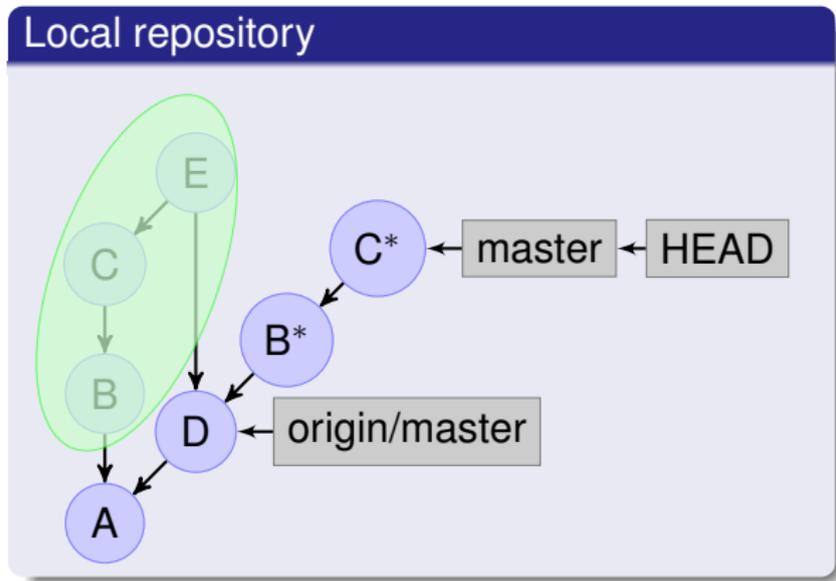
\$ eg rebase --against origin/master



\$ eg rebase --against origin/master

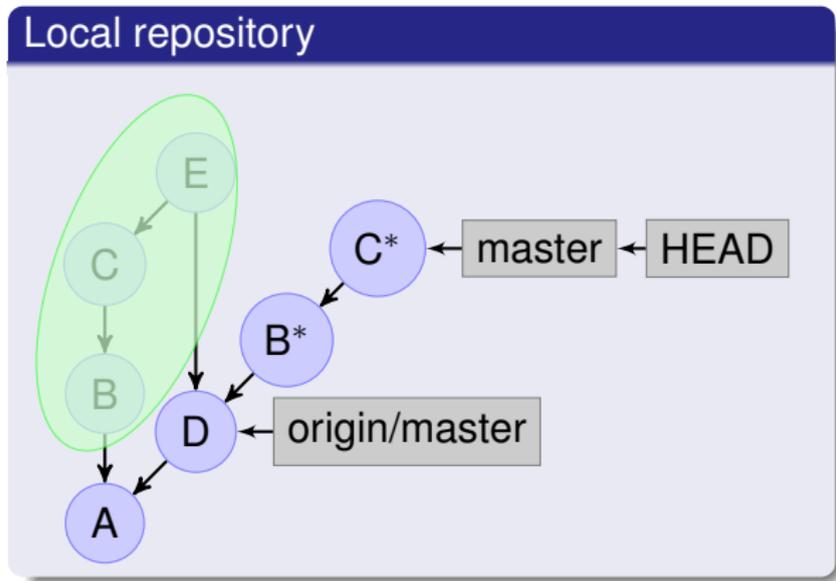


\$ eg rebase --against origin/master



Merge commits are dropped during rebases, so E is not applied.

\$ eg rebase --against origin/master

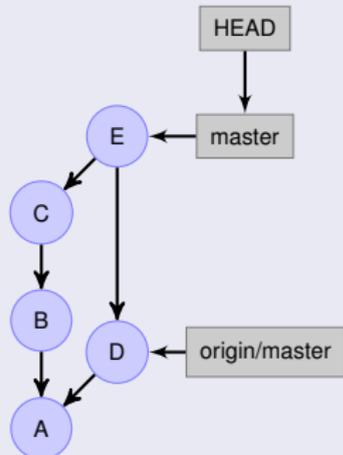


Merge commits are dropped during rebases, so E is not applied.

Interactive Rebasing (1/3) – Basic Rebasing

Starting point:

Local repository



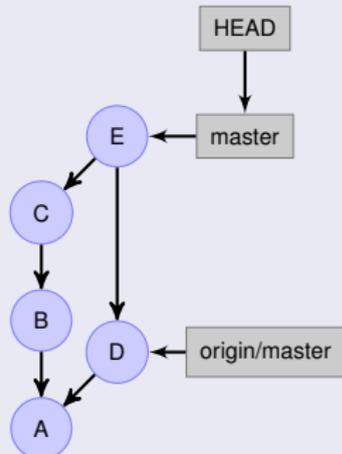
Terminal

\$

Interactive Rebasing (1/3) – Basic Rebasing

\$ eg rebase --interactive --against origin/master

Local repository



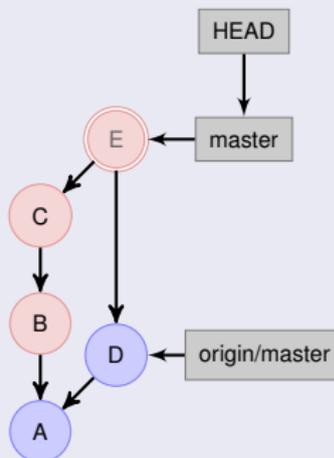
Terminal

```
$ eg rebase -i --against origin/master
```

Interactive Rebasing (1/3) – Basic Rebasing

\$ **eg rebase --interactive --against origin/master**

Local repository



Editor

```
pick b6bafc0 Commit message B
pick 44a6ae8 Commit message C

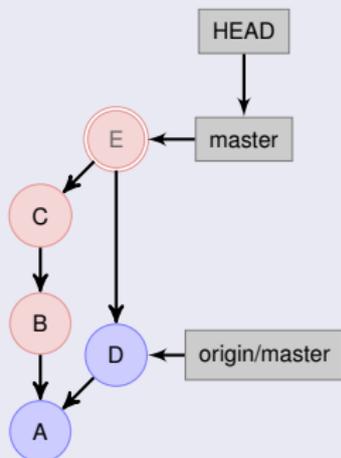
# Rebase e67fc45..fc16033 onto e67fc45
#
# Commands:
# p, pick = use commit
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#
```

Unnecessary merge commit E automatically dropped

Interactive Rebasing (1/3) – Basic Rebasing

\$ **eg rebase --interactive --against origin/master**

Local repository



Editor

```
pick b6bafc0 Commit message B
pick 44a6ae8 Commit message C

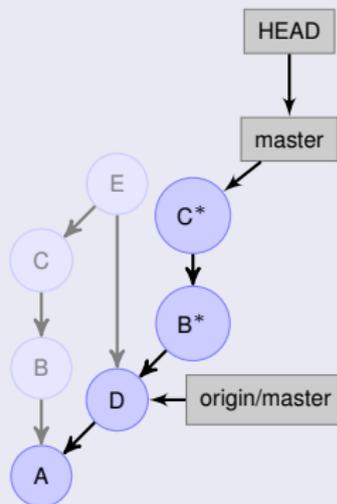
# Rebase e67fc45..fc16033 onto e67fc45
#
# Commands:
# p, pick = use commit
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#
```

Save and exit editor

Interactive Rebasing (1/3) – Basic Rebasing

\$ **eg rebase --interactive --against origin/master**

Local repository



Terminal

```
Rebasing (1/2)
```

```
Rebasing (2/2)
```

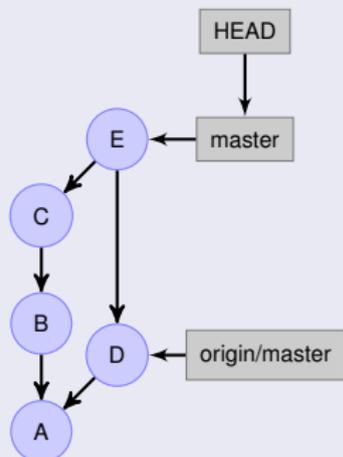
```
Successfully rebased and updated refs/heads/master.
```

```
$
```

Interactive Rebasing (2/3) – Reordering Commits

Starting point:

Local repository



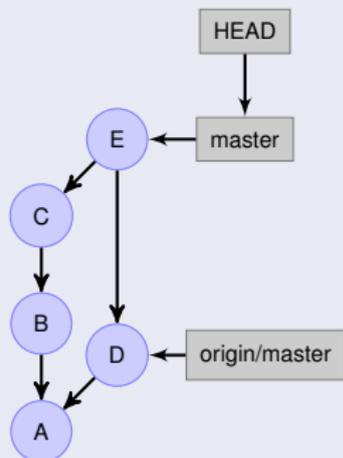
Terminal

\$

Interactive Rebasing (2/3) – Reordering Commits

\$ eg rebase --interactive --against origin/master

Local repository



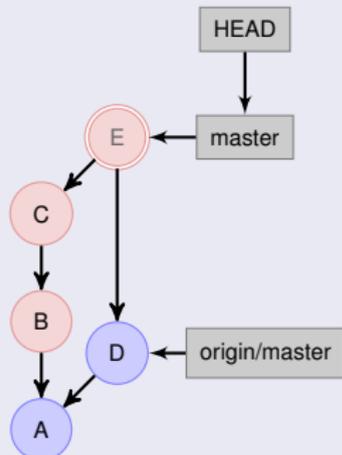
Terminal

```
$ eg rebase -i --against origin/master
```

Interactive Rebasing (2/3) – Reordering Commits

\$ **eg rebase --interactive --against origin/master**

Local repository



Editor

```
pick b6bafc0 Commit message B
pick 44a6ae8 Commit message C

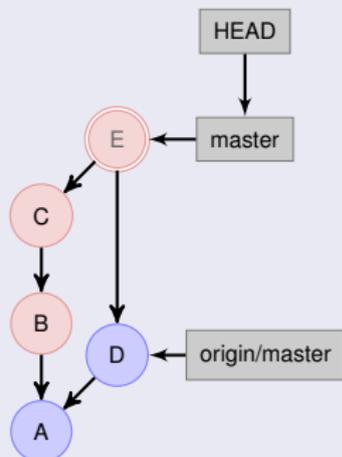
# Rebase e67fc45..fc16033 onto e67fc45
#
# Commands:
# p, pick = use commit
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#
```

Unnecessary merge commit E automatically dropped

Interactive Rebasing (2/3) – Reordering Commits

\$ **eg rebase --interactive --against origin/master**

Local repository



Editor

```
pick 44a6ae8 Commit message C
pick b6bafc0 Commit message B

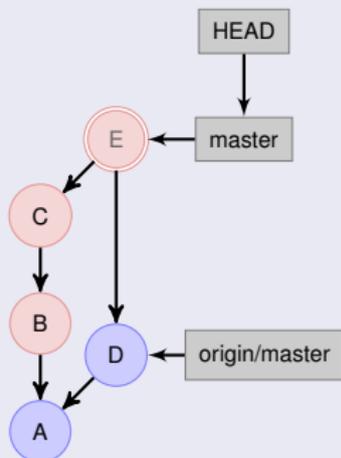
# Rebase e67fc45..fc16033 onto e67fc45
#
# Commands:
# p, pick = use commit
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#
```

Change the order of the commits

Interactive Rebasing (2/3) – Reordering Commits

\$ eg rebase --interactive --against origin/master

Local repository



Editor

```
pick 44a6ae8 Commit message C
pick b6bafc0 Commit message B

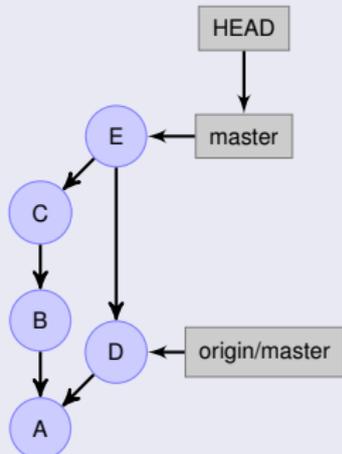
# Rebase e67fc45..fc16033 onto e67fc45
#
# Commands:
# p, pick = use commit
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#
```

Save and exit editor

Interactive Rebasing (3/3) – Squashing Commits

Starting point:

Local repository



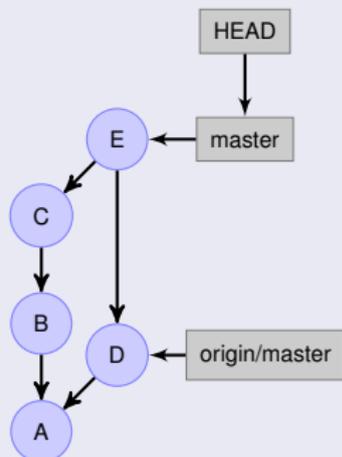
Terminal

\$

Interactive Rebasing (3/3) – Squashing Commits

\$ eg rebase --interactive --against origin/master

Local repository



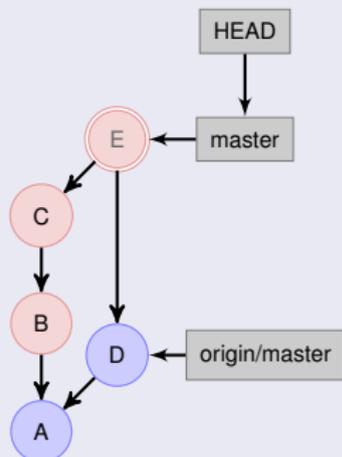
Terminal

```
$ eg rebase -i --against origin/master
```

Interactive Rebasing (3/3) – Squashing Commits

\$ **eg rebase --interactive --against origin/master**

Local repository



Editor

```
pick b6bafc0 Commit message B
pick 44a6ae8 Commit message C

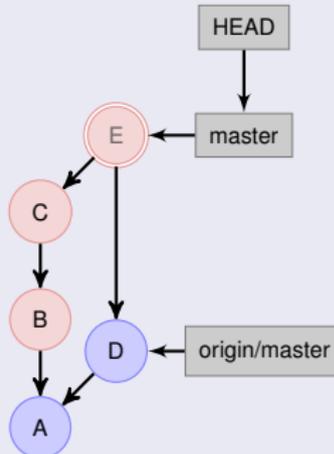
# Rebase e67fc45..fc16033 onto e67fc45
#
# Commands:
# p, pick = use commit
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#
```

Unnecessary merge commit E automatically dropped

Interactive Rebasing (3/3) – Squashing Commits

\$ **eg rebase --interactive --against origin/master**

Local repository



Editor

```
pick b6bafc0 Commit message B
squash 44a6ae8 Commit message C

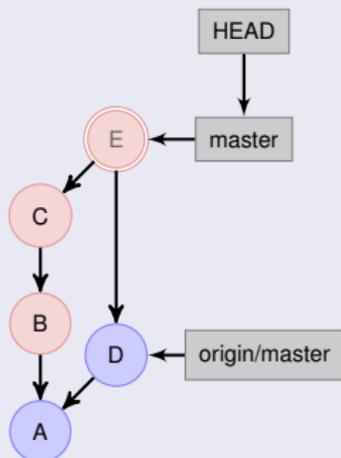
# Rebase e67fc45..fc16033 onto e67fc45
#
# Commands:
# p, pick = use commit
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#
```

*Change **pick** to **squash** for commit C*

Interactive Rebasing (3/3) – Squashing Commits

\$ **eg rebase --interactive --against origin/master**

Local repository



Editor

```
pick b6bafc0 Commit message B
squash 44a6ae8 Commit message C

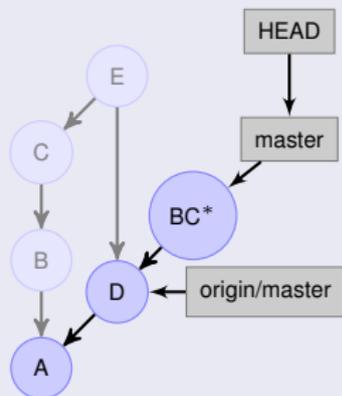
# Rebase e67fc45..fc16033 onto e67fc45
#
# Commands:
# p, pick = use commit
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#
```

Save and exit editor

Interactive Rebasing (3/3) – Squashing Commits

\$ **git rebase --interactive --against origin/master**

Local repository



Editor

```
# This is a combination of two commits.
# The first commit's message is:
```

Commit message B

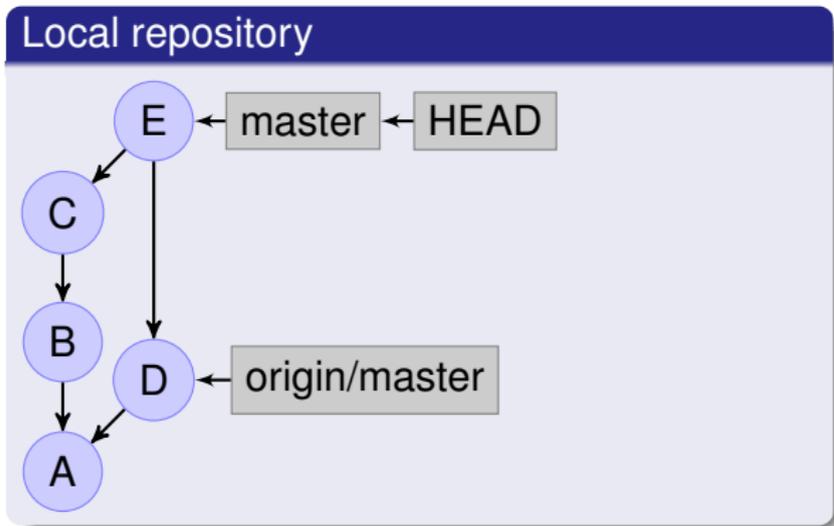
```
# This is the 2nd commit message:
```

Commit message C

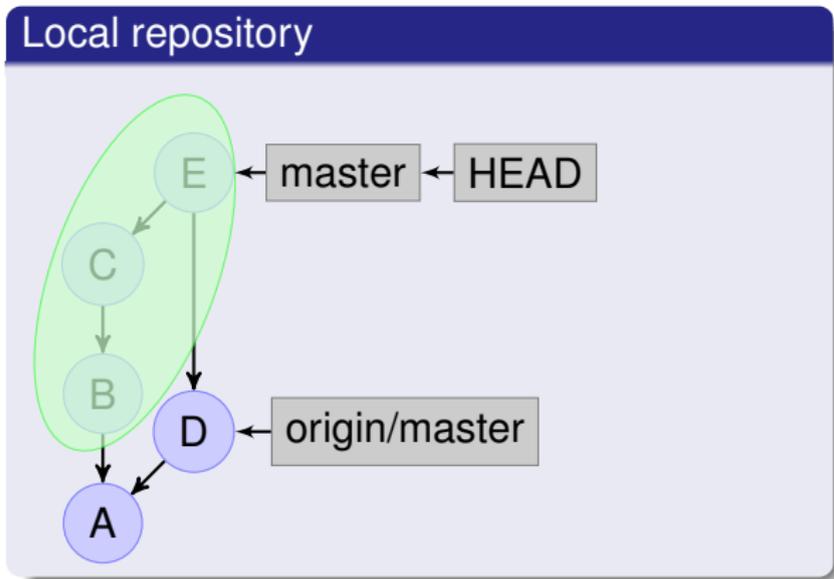
```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# Not currently on any branch.
# Changes to be committed:
# (use "git reset HEAD <file>..." to unstage)
#
# modified: A
#
```

Edit commit message for the ombined commit
Save and exit editor

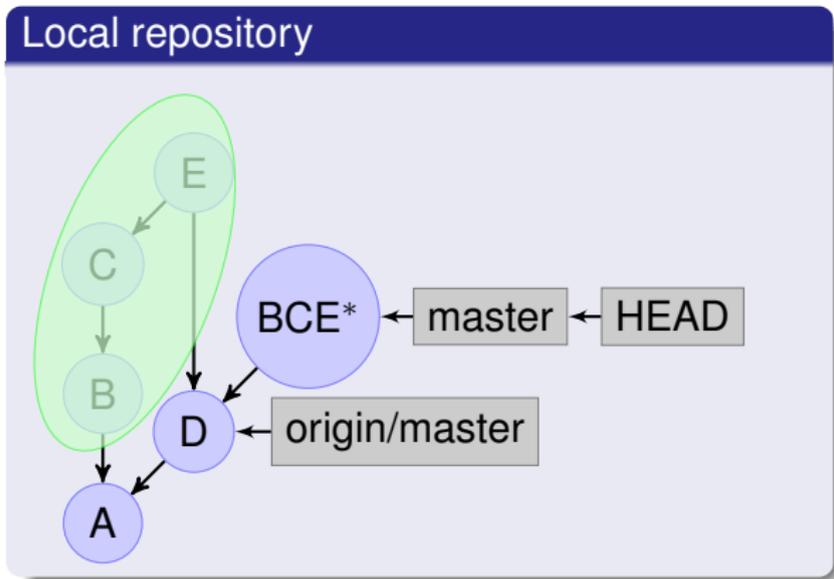
Starting point:



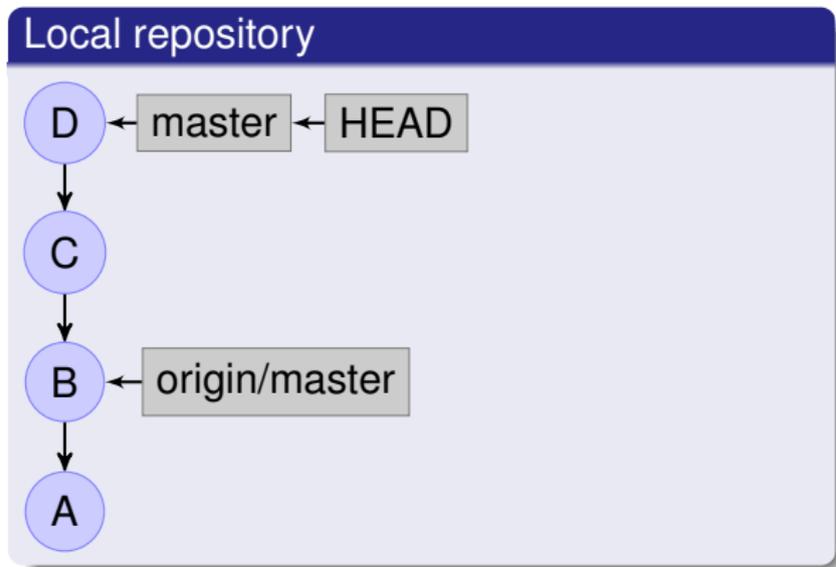
\$ eg squash [--against origin/master]



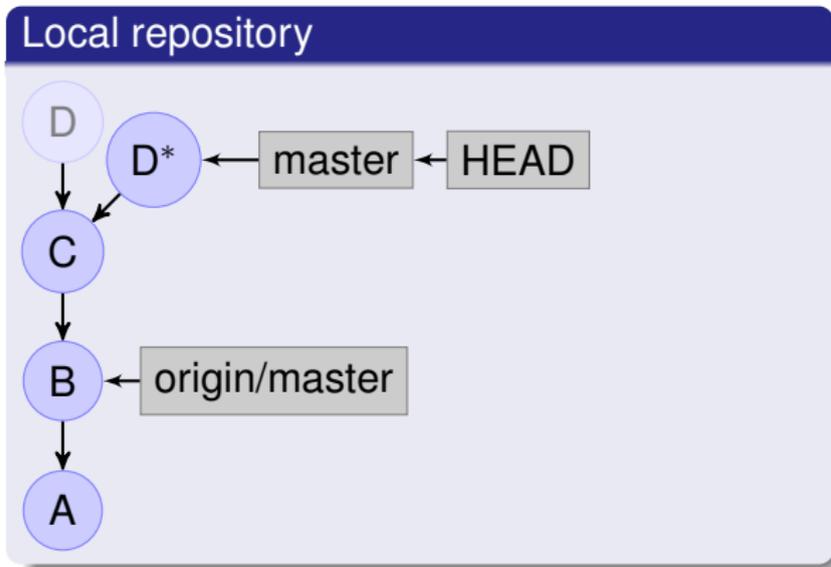
\$ eg squash [--against origin/master]



Starting point:



`$ eg commit --amend`



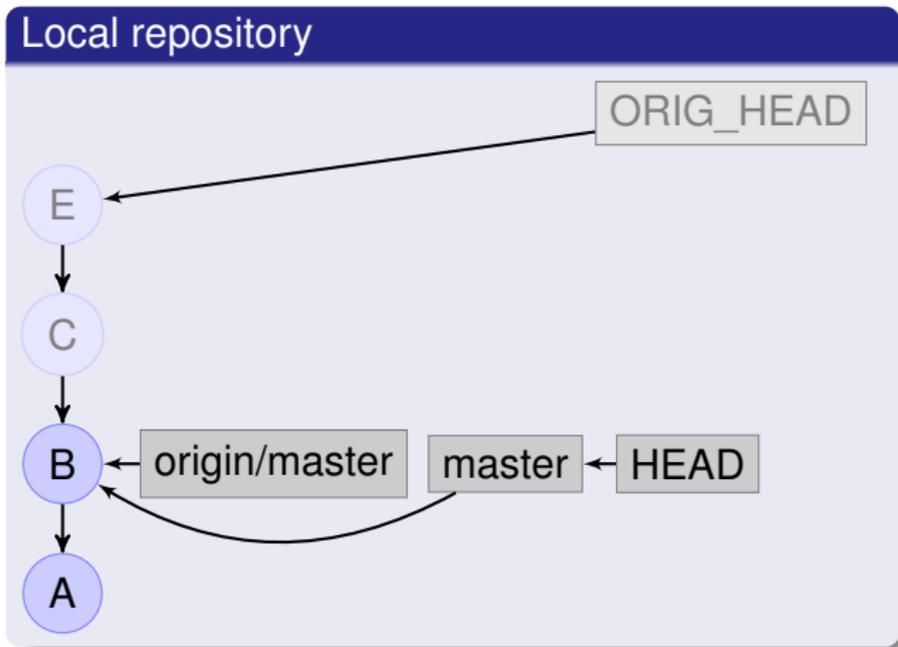
Undoing/Redoing Commits

Starting point:



Undoing/Redoing Commits

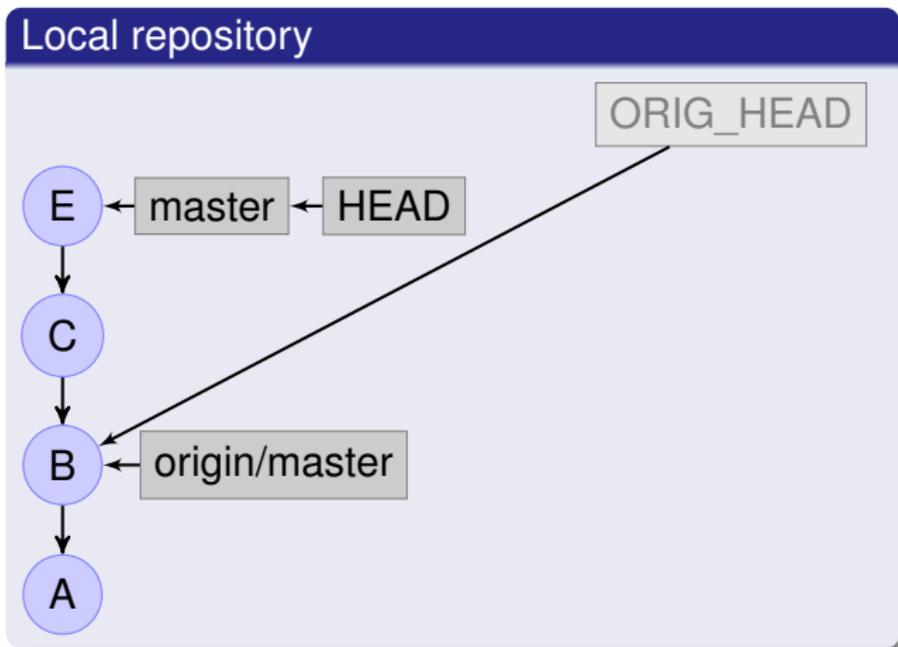
\$ eg reset --working-copy origin/master



`--working-copy`: Make sure the working copy exactly matches the specified commit.

Undoing/Redoing Commits

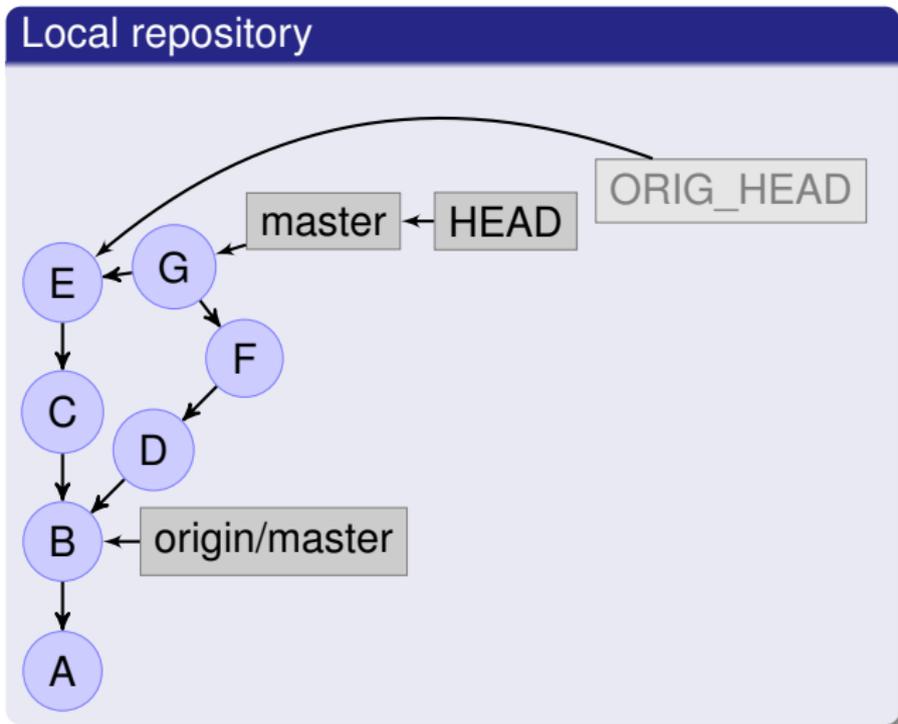
\$ eg reset --working-copy ORIG_HEAD



ORIG_HEAD: set by merge, rebase, and reset.

Undoing/Redoing Commits

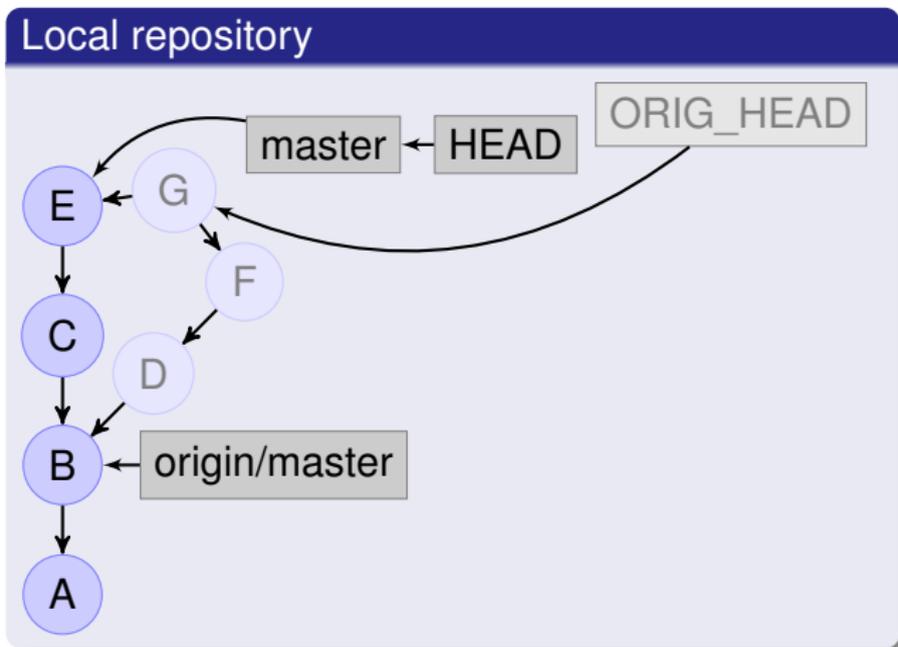
\$ eg pull



ORIG_HEAD: set by merge, rebase, and reset.

Undoing/Redoing Commits

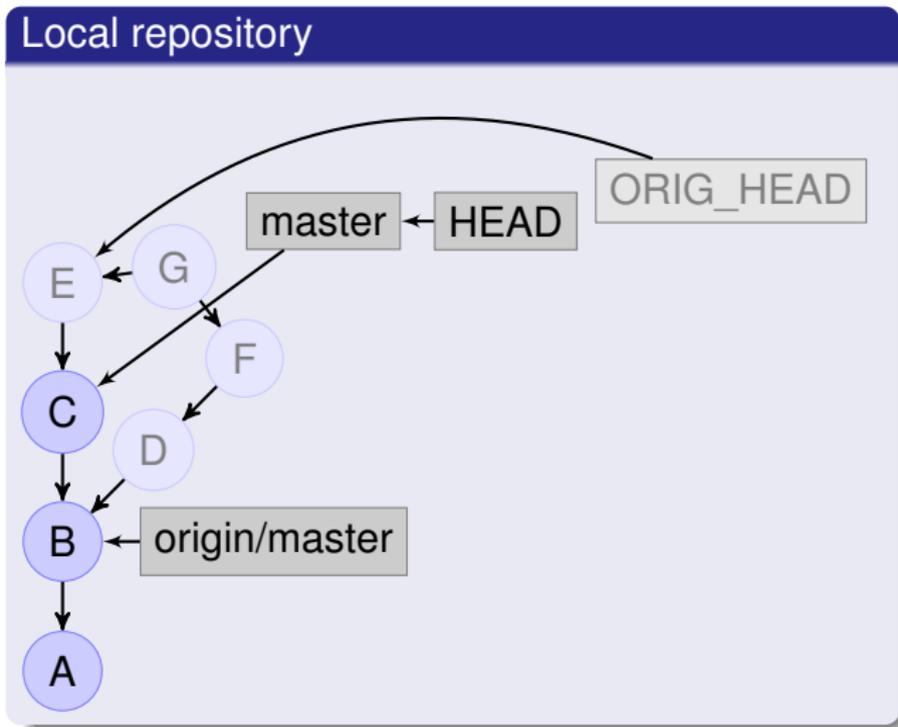
\$ eg reset --working-copy ORIG_HEAD



ORIG_HEAD: set by merge, rebase, and reset.

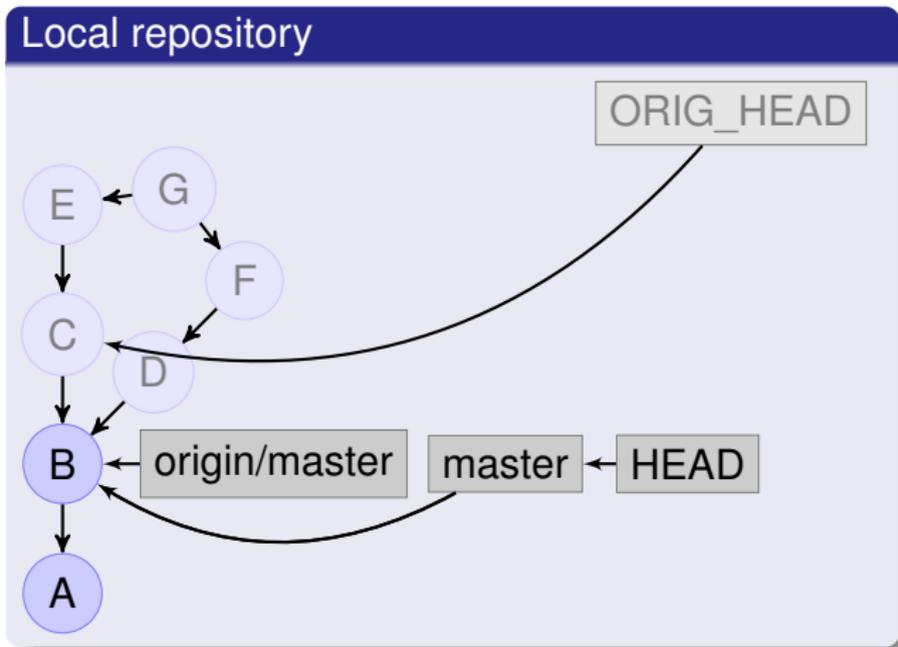
Undoing/Redoing Commits

\$ eg reset --working-copy master~1



Undoing/Redoing Commits

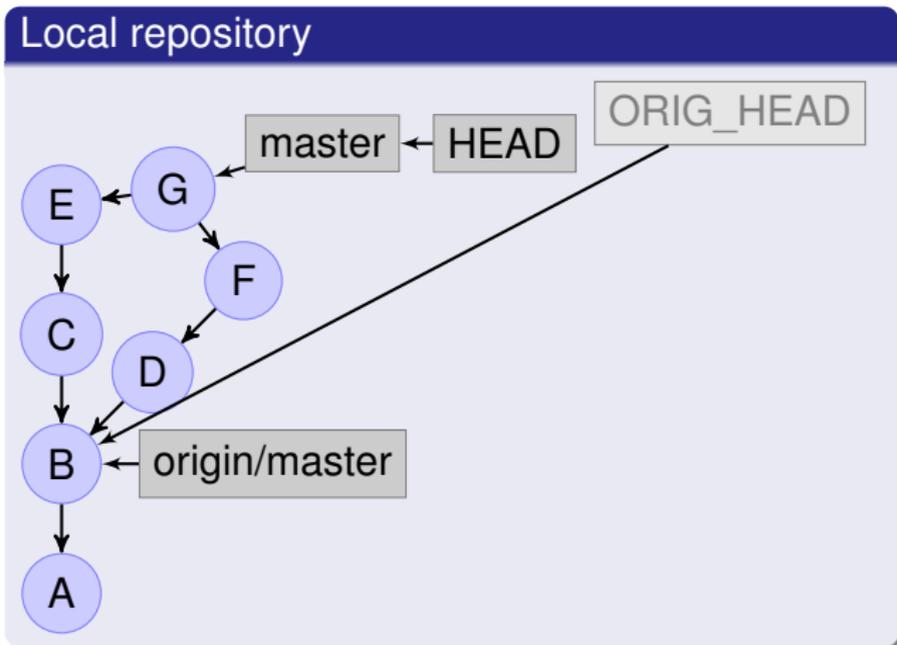
```
$ eg reset --working-copy master@{4}
```



To see where master previously pointed, run
`eg reflog show master`

Undoing/Redoing Commits

```
$ eg reset --working-copy master@{"30 seconds ago"}
```

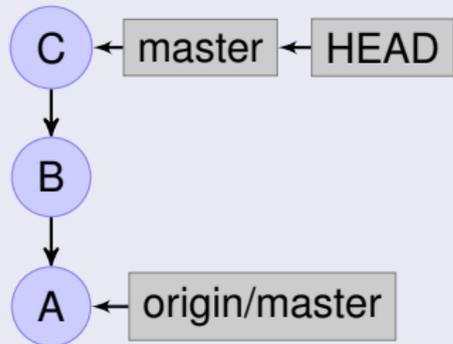


To see where master previously pointed, run
`eg reflog show master`

- 1 History Layout
- 2 Getting Information from git
- 3 Cleaning Up
- 4 Multiple Branches, Multiple Repositories**
 - Multiple branches within a project
 - Direct collaboration
- 5 Tools and Miscellaneous Tips

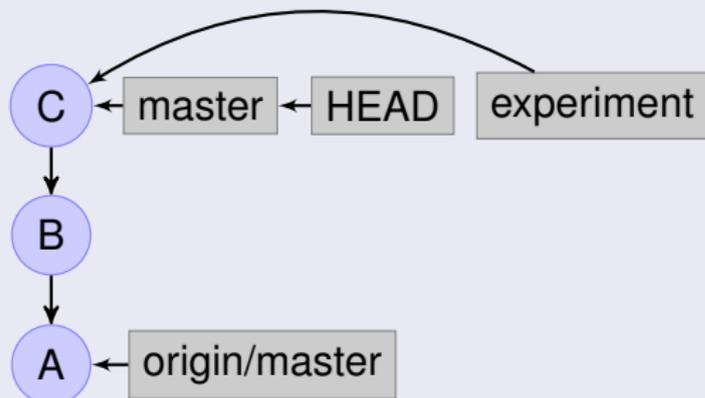
Starting point:

Local repository



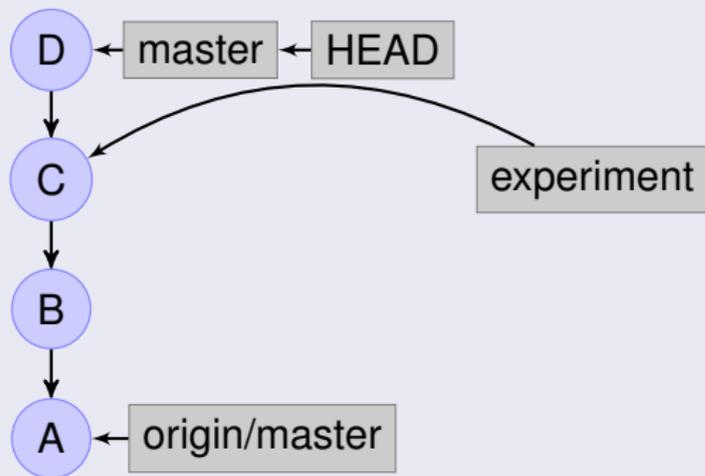
\$ eg branch experiment

Local repository



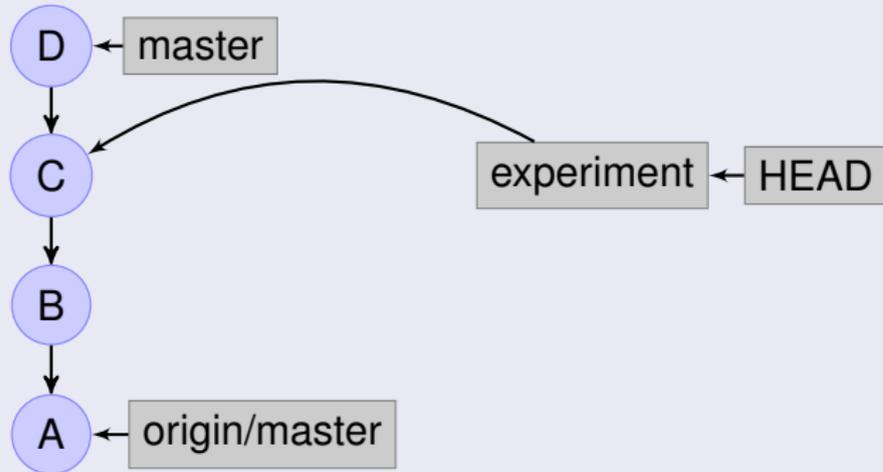
After making a commit

Local repository



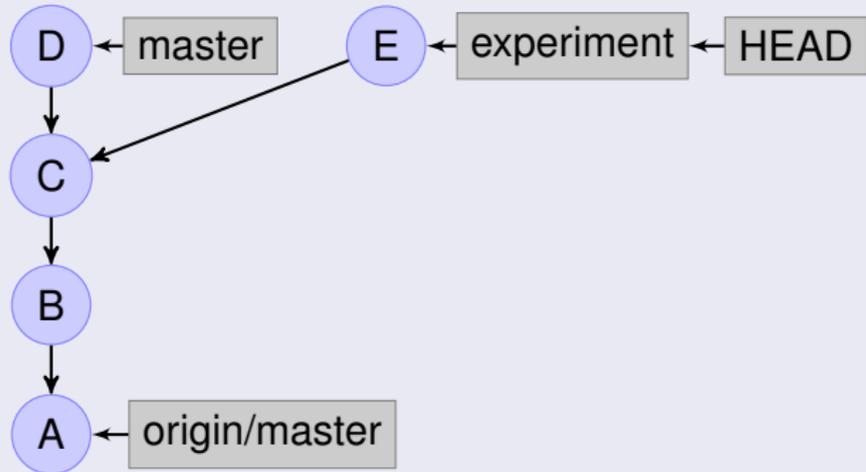
\$ eg switch experiment

Local repository



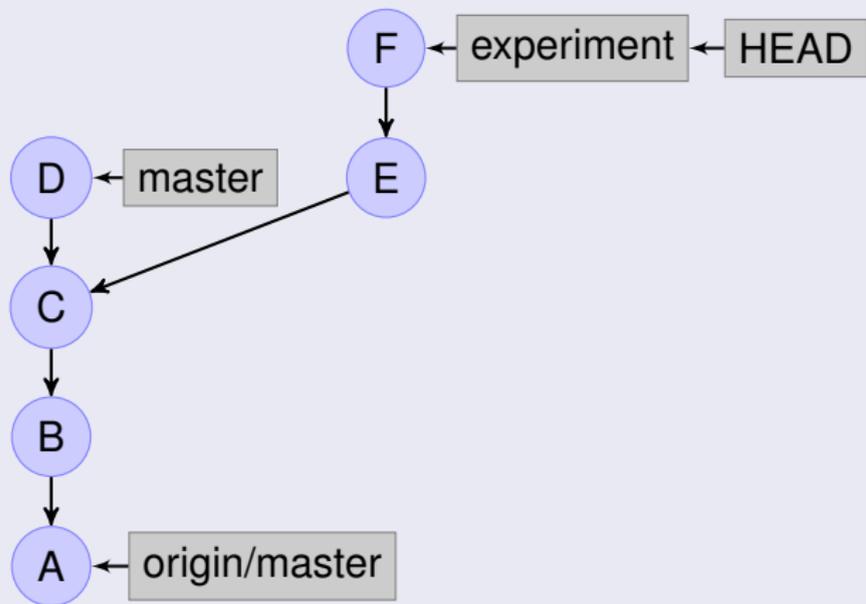
After making a commit

Local repository



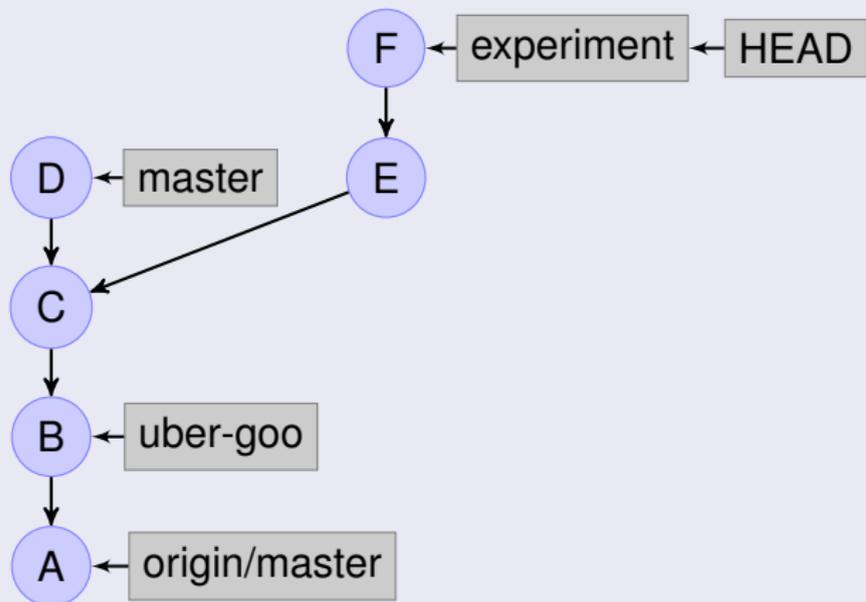
...and another commit

Local repository



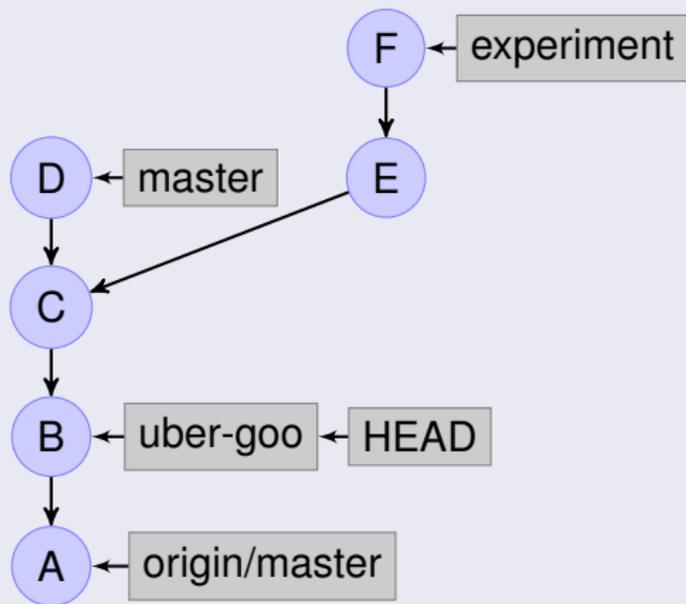
\$ eg branch uber-goo master~2

Local repository



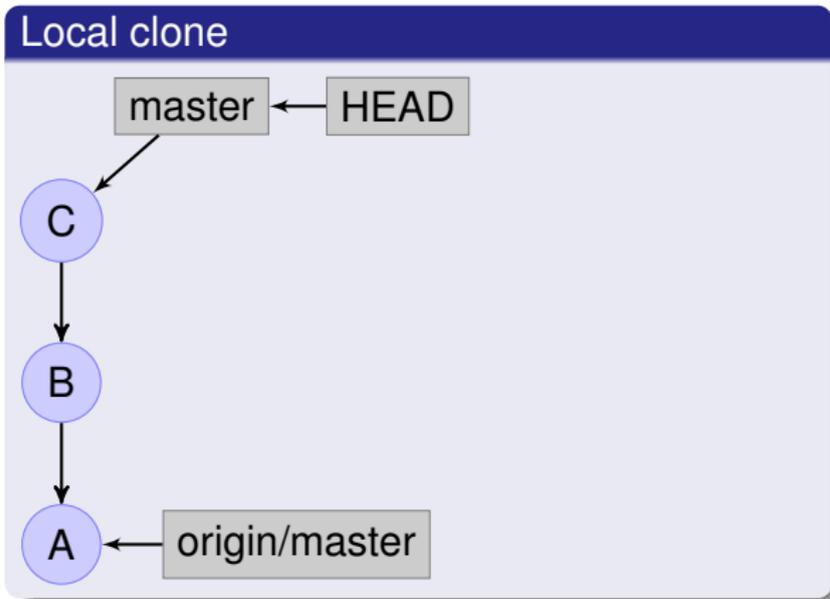
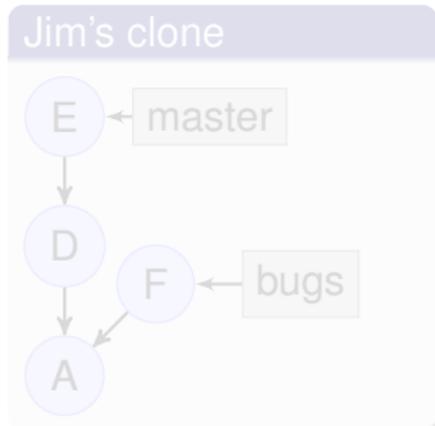
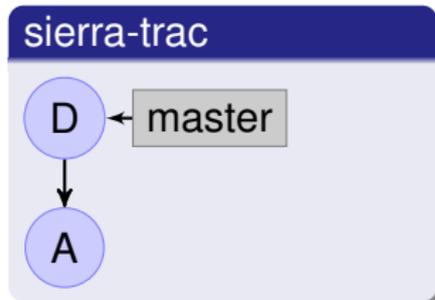
\$ eg switch uber-goo

Local repository



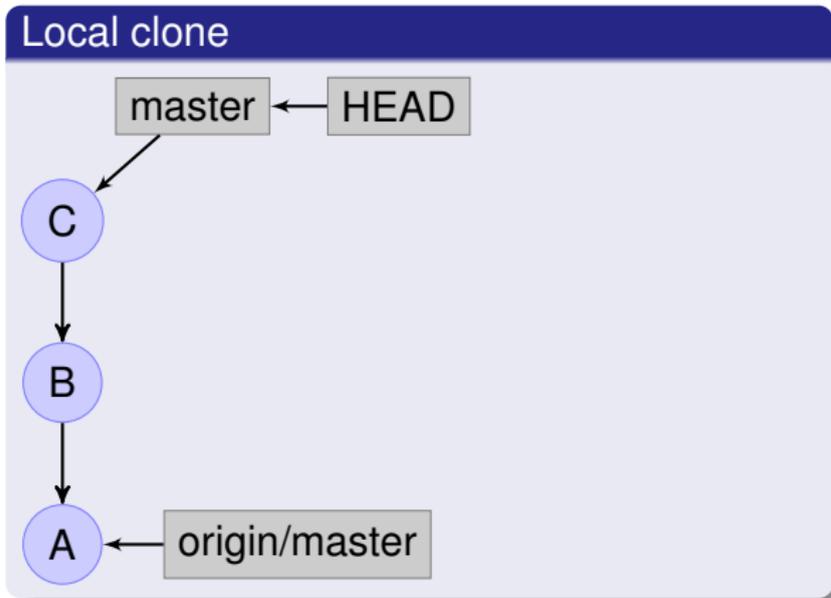
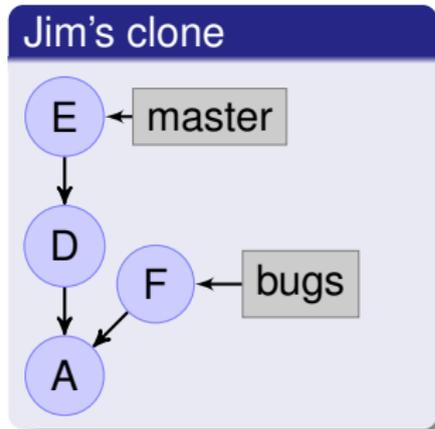
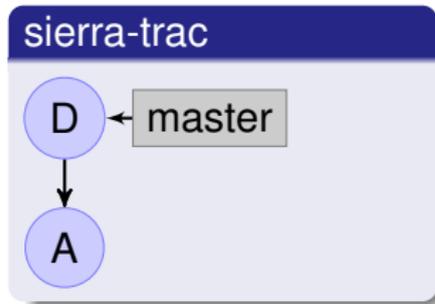
Direct Collaboration (Remotes)

Initial State:



Direct Collaboration (Remotes)

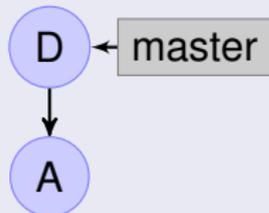
Initial State:



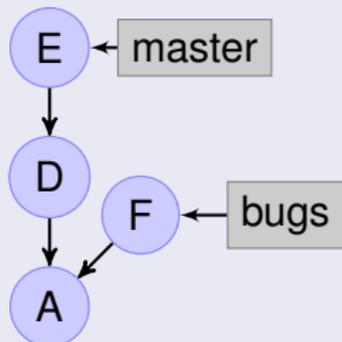
Direct Collaboration (Remotes)

\$ **eg remote add jim machine:/PATH...**

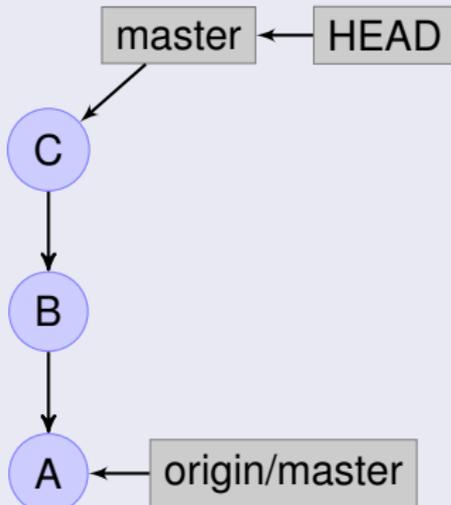
sierra-trac



Jim's clone



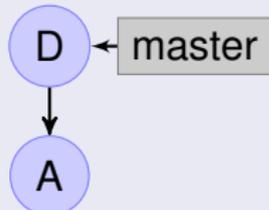
Local clone



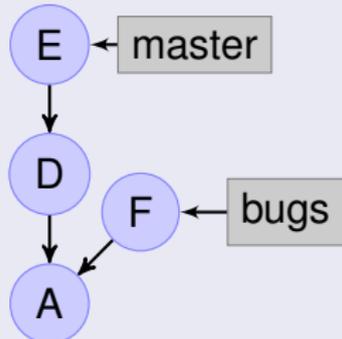
Direct Collaboration (Remotes)

\$ eg fetch jim

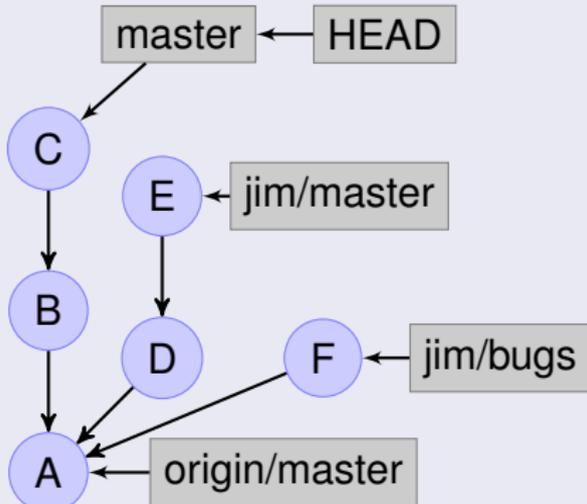
sierra-trac



Jim's clone



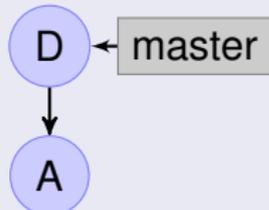
Local clone



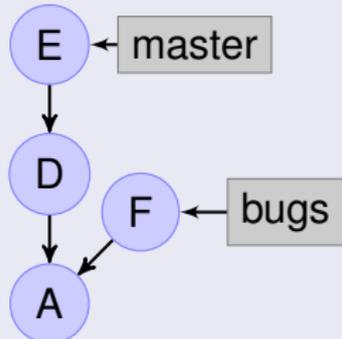
Direct Collaboration (Remotes)

\$ eg remote rm origin

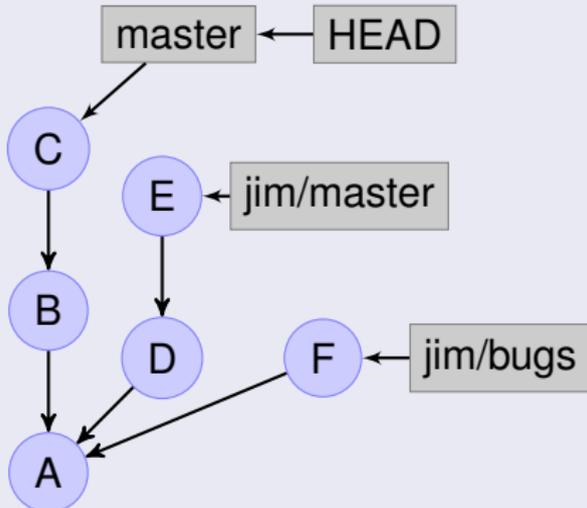
sierra-trac



Jim's clone



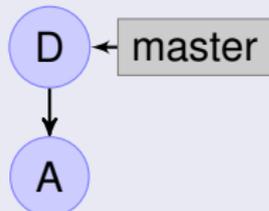
Local clone



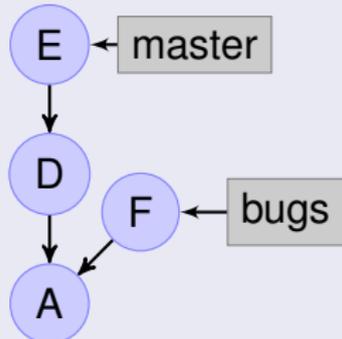
Direct Collaboration (Remotes)

\$ eg remote rename jim origin

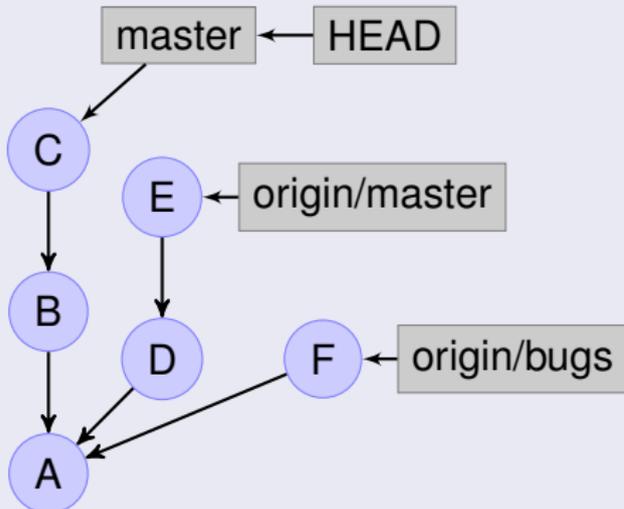
sierra-trac



Jim's clone



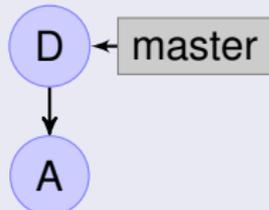
Local clone



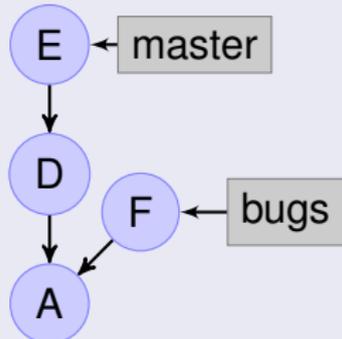
Direct Collaboration (Remotes)

Use `eg remote show` or `eg info` to see remotes.

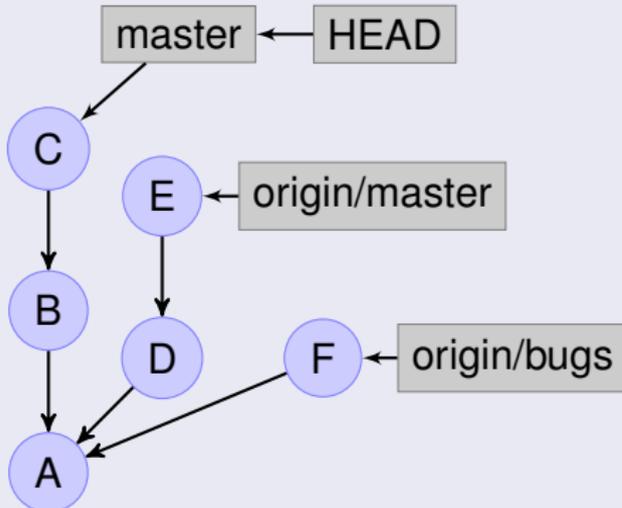
sierra-trac



Jim's clone



Local clone



- 1 History Layout
- 2 Getting Information from git
- 3 Cleaning Up
- 4 Multiple Branches, Multiple Repositories
- 5 Tools and Miscellaneous Tips**
 - Ignoring files

Git provides three mechanisms for ignoring files

- `.gitignore` files – can be checked-in and shared via repository
- `$GIT_DIR/info/excludes` file – local to a project/clone (personal)
- `core.excludesfile` configuration parameter – can be per-project or global (personal)

Common usage

Setup a `~/ .gitignore` file for all your Git projects

```
$ echo checkin-comments.txt > ~/ .gitignore
$ echo /tests/ >> ~/ .gitignore
$ echo *.bak >> ~/ .gitignore
$ git config --global core.excludesfile ~/ .gitignore
```

Git provides three mechanisms for ignoring files

- `.gitignore` files – can be checked-in and shared via repository
- `$GIT_DIR/info/excludes` file – local to a project/clone (personal)
- `core.excludesfile` configuration parameter – can be per-project or global (personal)

Common usage

Setup a `~/.gitignore` file for all your Git projects

```
$ echo checkin-comments.txt > ~/.gitignore
$ echo /tests/ >> ~/.gitignore
$ echo \*.bak >> ~/.gitignore
$ git config --global core.excludesfile ~/.gitignore
```

Git provides three mechanisms for ignoring files

- `.gitignore` files – can be checked-in and shared via repository
- `$GIT_DIR/info/excludes` file – local to a project/clone (personal)
- `core.excludesfile` configuration parameter – can be per-project or global (personal)

Common usage

Setup a `~/ .gitignore` file for all your Git projects

```
$ echo checkin-comments.txt > ~/ .gitignore
$ echo /tests/ >> ~/ .gitignore
$ echo \*.bak >> ~/ .gitignore
$ git config --global core.excludesfile ~/ .gitignore
```

Motivation

You want save off your uncommitted changes to clear out your working copy but you're not ready to do a commit.

Sometimes you want to

- safely do a pull...
 - switch to another branch to work on something else...
 - experiment with different changes...
- ... without making a commit.

Motivation

You want save off your uncommitted changes to clear out your working copy but you're not ready to do a commit.

Sometimes you want to

- safely do a pull. . .
- switch to another branch to work on something else. . .
- experiment with different changes. . .

. . . without making a commit.

Motivation

You want save off your uncommitted changes to clear out your working copy but you're not ready to do a commit.

Sometimes you want to

- safely do a pull. . .
- switch to another branch to work on something else. . .
- experiment with different changes. . .

. . . without making a commit.

Motivation

You want save off your uncommitted changes to clear out your working copy but you're not ready to do a commit.

Sometimes you want to

- safely do a pull. . .
- switch to another branch to work on something else. . .
- experiment with different changes. . .

... without making a commit.

Motivation

You want save off your uncommitted changes to clear out your working copy but you're not ready to do a commit.

Sometimes you want to

- safely do a pull. . .
 - switch to another branch to work on something else. . .
 - experiment with different changes. . .
- . . . without making a commit.

```
$ eg status
```

```
(On branch master)
```

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

```
    modified:   texmf-paths.sh
```

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

```
    modified:   git-sierra-advanced.tex
```

```
$ eg stash
```

```
Saved working directory and index state "WIP on master: 5395e8a Lead with .  
HEAD is now at 5395e8a Lead with ./ in the texmf path search.
```

```
(To restore them type "git stash apply")
```

```
$ eg stash list
```

```
5395e8a Lead with ./ in the texmf path search.
```

```
$ eg status
```

```
(On branch master)
```

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

```
    modified:   texmf-paths.sh
```

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

```
    modified:   git-sierra-advanced.tex
```

```
$ eg stash
```

```
Saved working directory and index state "WIP on master: 5395e8a Lead with .  
HEAD is now at 5395e8a Lead with ./ in the texmf path search.
```

```
(To restore them type "git stash apply")
```

```
$ eg stash list
```

```
5395e8a Lead with ./ in the texmf path search.
```

```
$ eg status
```

```
(On branch master)
```

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

```
    modified:   texmf-paths.sh
```

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

```
    modified:   git-sierra-advanced.tex
```

```
$ eg stash
```

```
Saved working directory and index state "WIP on master: 5395e8a Lead with .  
HEAD is now at 5395e8a Lead with ./ in the texmf path search.
```

```
(To restore them type "git stash apply")
```

```
$ eg stash list
```

```
5395e8a Lead with ./ in the texmf path search.
```

```
$ eg stash pop
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout - <file>..." to discard changes in working directory)
#
# modified:   git-sierra-advanced.tex
# modified:   texmf-paths.sh
#
no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@0 (3033e15e15e0b4037cbaafbdfdf89ec578aa90f)
```

Questions?

For the most part you can replace `eg` with `git` in all the commands listed in this presentation. There are some exceptions:

- `eg rebase --against ...`
⇒ `git rebase ...`
- `eg diff (no arguments)`
⇒ `git diff HEAD`
- `eg squash --against ...`
⇒ `git reset --soft ... && git commit`
- `eg reset --working-copy ...`
⇒ `git reset --hard ...`
- `eg switch ...`
⇒ `git checkout ...`

If after doing this command translation, you again replace `git` with `eg`, the commands will continue to work in all cases. (`eg` accepts a superset of the input that `git` accepts)