clearvision

Module 4

Working with Git

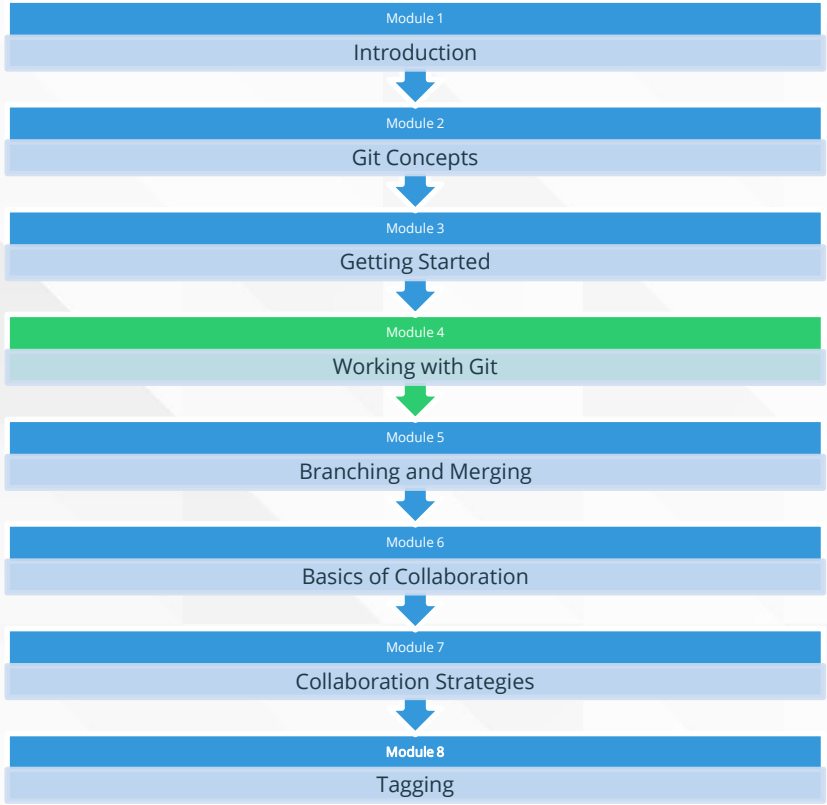
Atlassian Confluence

Module Contents

clearvision

Module Contents

- Viewing Changes
- Fixing Errors
- Amend
- Stash
- Reset
- Checkout
- Revert



git

clearvision

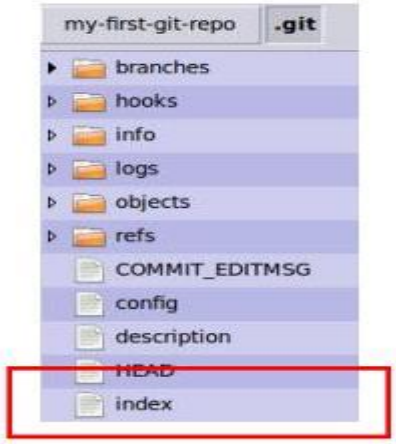
Viewing Changes

Atlassian Confluence

Git - Status

clearvision

- To view the status of your staging area at any time use the *git status* command
- This command will query your git index file
- The results from the index are then compared to both the working directory and the head revision in your repository



git

Ignoring Files

clearvision

Often, you may find files in your working directory which should not be tracked, such as build files, or backups

This can be achieved by creating an ignore file in a specific location

These can be in one of three locations

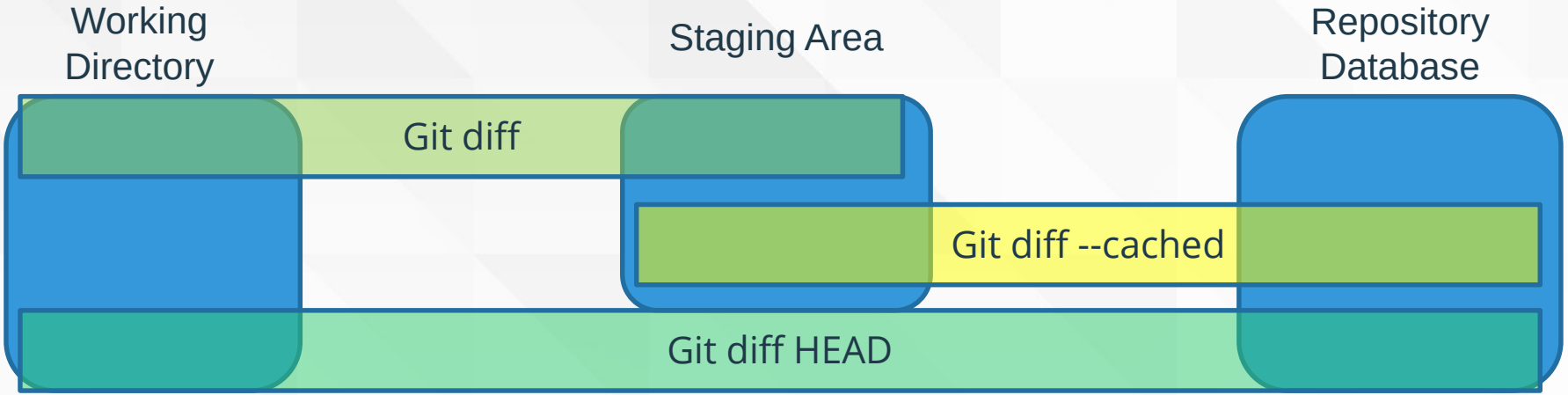
.gitignore	exclude	.gitconfig
The working directory	Placed within the “info” directory of the repository database.	Your user home directory

Once ignored the *git status* command will not report on these files.

git

Diff

- If you want to see more detail about the changes you have made than is given by the *git status* command, you can use the *git diff* command.
- The *git diff* command shows exactly which lines have been changed, added or removed in your modified files.



 |

Git Stash

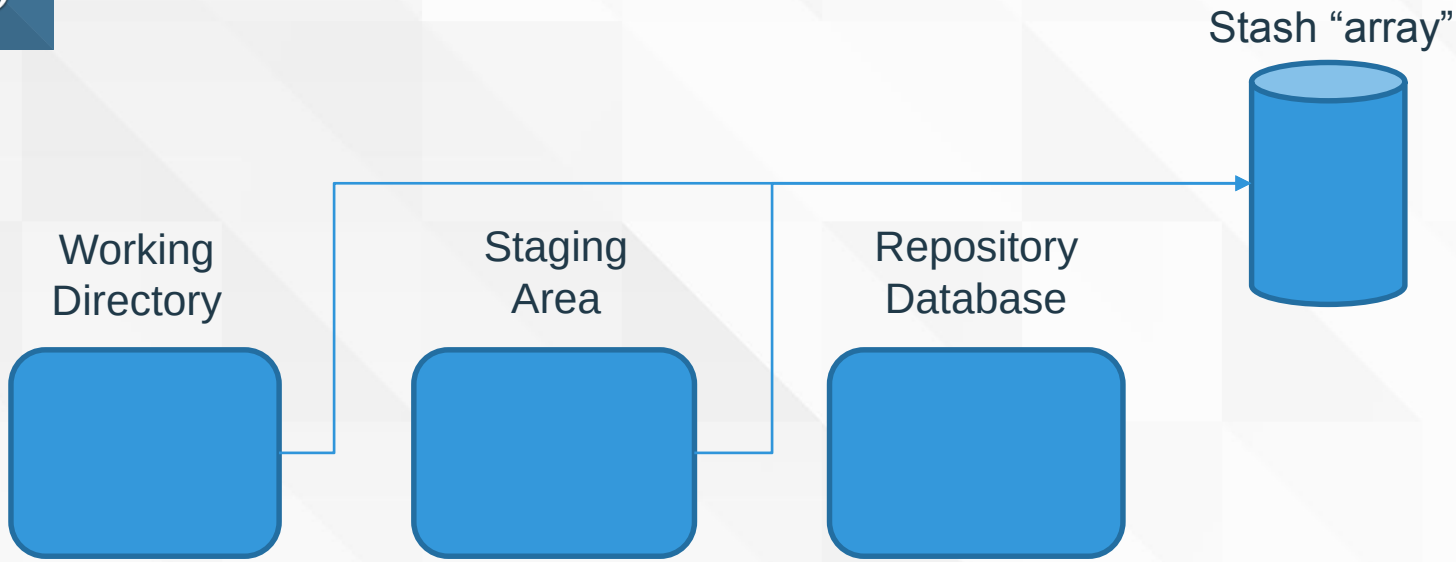
clearvision

- It is not uncommon to be in the middle of an activity when you need to switch to something different
- What do you do with this unfinished code?
  - If you swap branches, uncommitted code will come with you!
  - If you commit it, you are writing immutable history
  - If you clean up, you lose your changes
- The answer :
  - *git stash*

git

Git Stash

clearvision



- *Git stash* reads your working directory and (optionally) the index and stores those changes in a separate location
- It then cleans up your working directory

git

Git Stash

clearvision

git stash save	Save the current changes in the next available stash position
git stash list	List all available stashes
git stash apply stash@{0}	Re-apply the stash at position 0
git stash drop stash@{0}	Delete the stash at position 0
git stash branch <branchname> stash@{0}	Create a new branch based on the stash at 0
--index	Include the index when saving or applying a stash

git

clearvision

Fixing Errors

Atlassian Confluence

Fixing Mistakes

clearvision

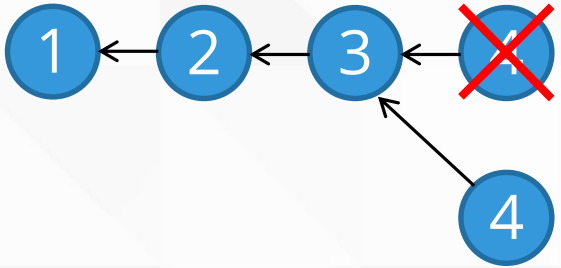
- Git is very helpful when it comes to controlling what you commit with status, stash etc but :
- What if you staged something by mistake?
- Or imagine a scenario where you have mistakenly committed your changes and need to back it out.
- The Git commands reset , revert & checkout can all be used to undo changes depending on the specific scenario.

git

Commit --Amend

clearvision

- If you notice a mistake just after a commit – you can easily correct this.
- Add the --amend option to the git commit command.
- This will overwrite the last commit you made with a current snapshot.
- Use this to :
  - Correct mistakes missed from the last commit
  - Change the commit message without changing it's changeset.



git

Reset

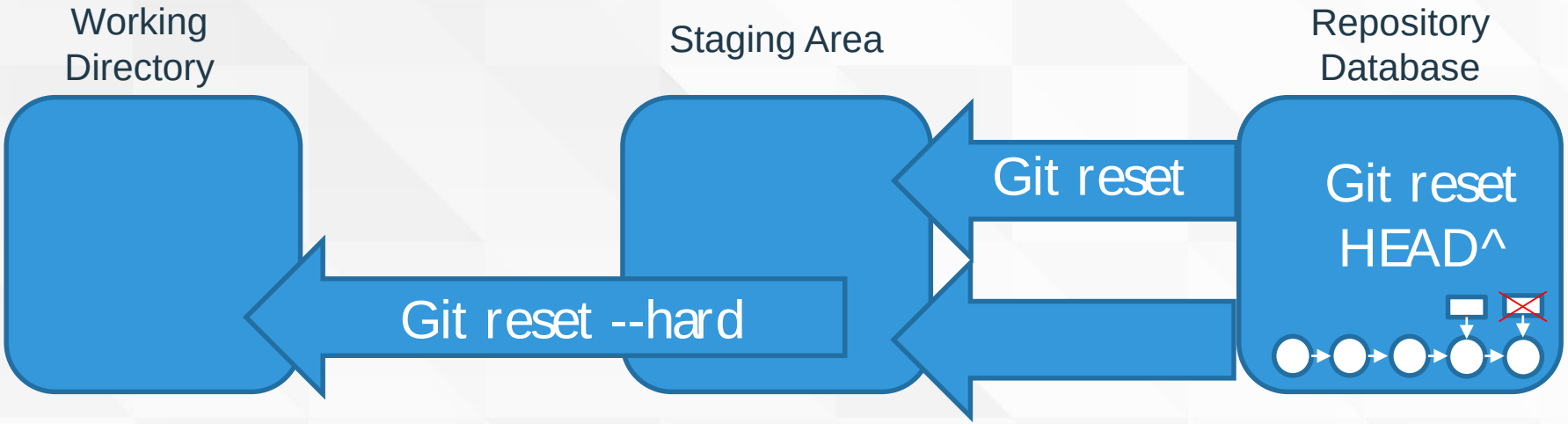
clearvision

- The reset command is a powerful tool for rewinding changes
- You can :
  - “Unstage” a change (*git reset <file>)*
  - Clear out local modifications (*git reset --hard)*
  - Delete the history or a branch (*git reset HEAD~n)*
  - “Squash” multiple commits together (*git reset --soft HEAD~n)*

git

Reset

clearvision



git

Checkout

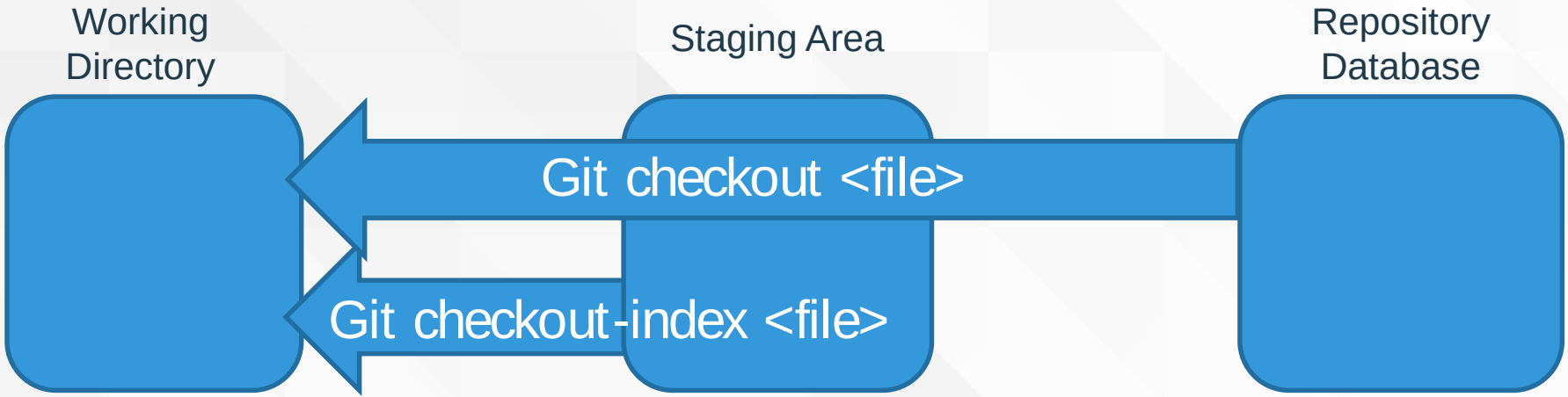
clearvision

- The Checkout command can be used to recover files or fix errors in the working directory.
- By checking out either a branch or file you can change the contents of your working directory to match the HEAD (or any other reference / commit).
- If you have made local changes (which would be lost) you will have to use the --force (-f) option to overwrite those changes.

git

Removing local changes

clearvision



git

Rewriting History

clearvision

- Because git is distributed, the history you have created exists locally.
- This leaves you free to rewrite it as often as you want to e.g.
  - commit --amend
  - reset HEAD~n
  - rebase (more on that later)
  - merge --squash
- However what happens if others share that history?

git

Rewriting History

clearvision

user 1

user 2

user 1 squashes history

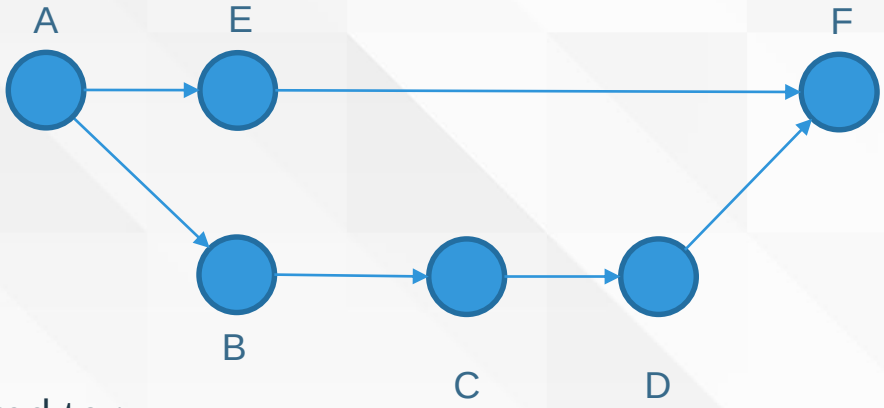
This will be rejected by default when you try to share your changes with other users

git

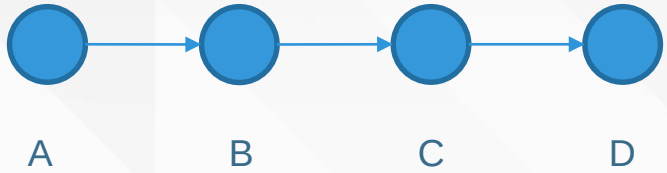
Rewriting History

clearvision

Even though nothing has actually changed, the end result of the merge will look something like this :



Compared to :



git

Revert

clearvision

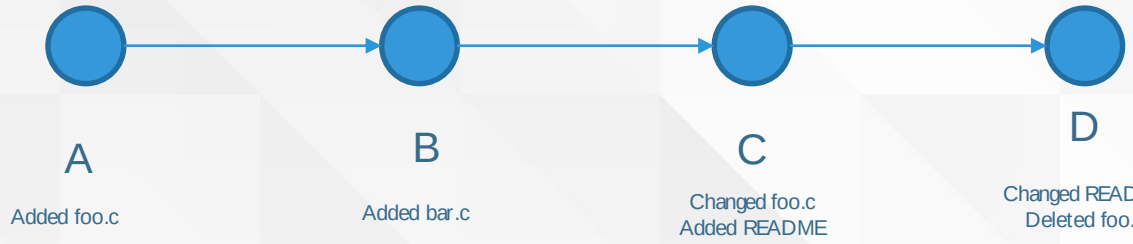
- The revert command introduces a new commit to the object database that reverses the effects (delta) of a specified commit. i.e. to 'undo' a commit that is buried in the history.
- Ensure you have a clean working directory prior to invoking git revert.
- Because history is not affected, this is perfectly safe.

- There is a full list of error recovery commands at the end of this section

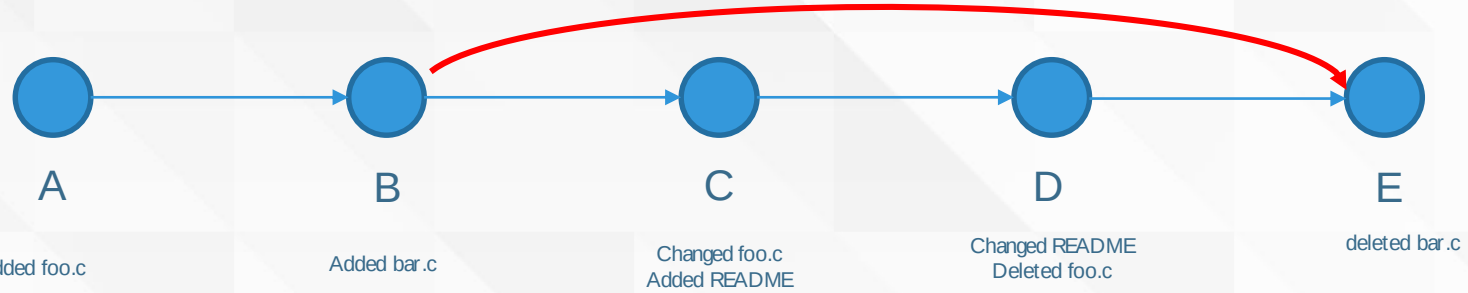
git

Revert

clearvision



If we revert B...



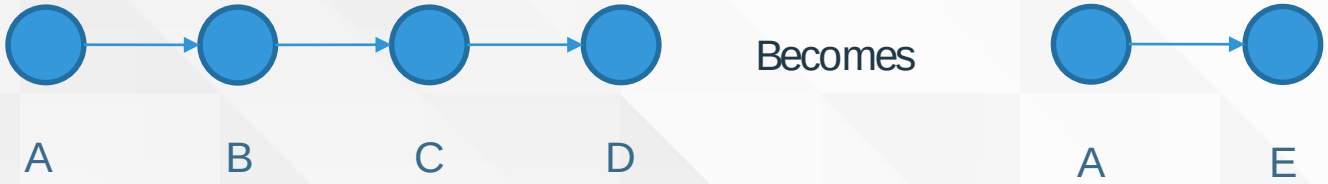
The opposite delta is applied to E

git

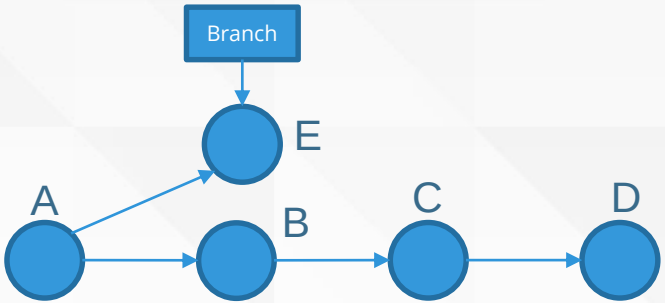
Unreachable Objects

clearvision

- There is another impact of rewriting history, what happens if



- B,C and D are no longer referenced by the branch, they (and all ***unique*** blob and tree objects beneath them) become *unreachable objects*



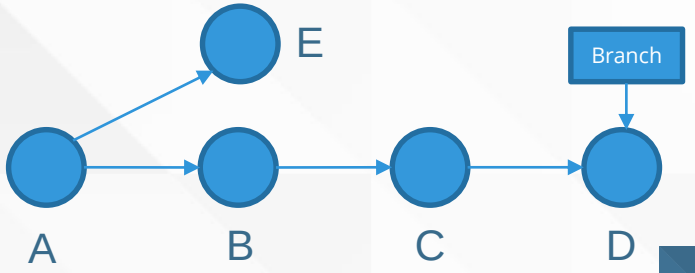
- A & E are referenced, the others are not

git

Unreachable Objects

clearvision

- Unreachable objects will be cleaned up automatically by garbage collection after 21 days.
- However until then they can be recovered.
- A record is maintained whenever a reference is moved
- For example, if we squash…
  - When we reset from D to A
  - When we commit to create E
- This record is called the *reflog*



git

The Reflog

clearvision

- The command *git reflog* will list recent movements of any reference including the HEAD (up to 90 days)

```
$ git reflog
f9a3170 HEAD@{0}: commit: squashed stuff together
bbd0af9 HEAD@{1}: reset: moving to HEAD~4
0a55e2a HEAD@{2}: commit: added README
49bd05f HEAD@{3}: commit: edited foo.c
530b31f HEAD@{4}: commit: edited bar.c
9e8ba1d HEAD@{5}: commit: added bar.c
bbd0af9 HEAD@{6}: commit (initial): foo.c
```

- To recover a commit, simply use
  - *git checkout –b <newbranch> <commit>*
- E.g.
  - *git checkout –b recovery_branch 0a55e2a*

git

clearvision

Lab Exercise

Atlassian Confluence

clearvision

Module 04 – Working with Git

ADDITIONAL RESOURCES

Undoing Changes - Recap

clearvision

Whilst checkout appears similar to reset, its operation is fundamentally different and only overlaps when referencing the HEAD commit object. Reference any other object and you will see very different results.

- Reset : points the tip of the current branch to a different commit
- Checkout : never changes the commit at the tip of any branch

Some scenarios and commands to use:

- Unstage a file, but keep changes - git reset <file>
- Unstage a file and throw away changes since last commit - git checkout HEAD <file>
- Undo a change in working copy to match the index - git checkout -- <file>
- To reset working directory and index to the last commit:
  - git checkout -f <current branch>
  - git reset --hard
- To rollback history to a previous commit to redo - git reset --soft <ref>
- To rollback history to a previous commit permanently - git reset --hard <ref>
- To undo a specific commit but record in history - git revert <ref>
- To alter the last commit with your current index – git commit --amend

git