Kenneth Benzie (Benie)

Runtime Engineer
Codeplay Software Ltd.

Source Control

Git

Everything is local

Source Control Systems

Distributed

- Git
- Mercurial
- ...

Client - Server

- Subversion
- Perforce
- CVS
- ...

Git Terminology

Repository - Init - Commit - Status

Add/Staging - Stash - Log - Branch

Checkout - Head - Reset

Merge Conflicts - Mergetool - Diff

Patch - Revert - Clean - Remote - Clone

Pull - Push - Fetch - Blame

Submodule - Rebase

Repository

- Is a directory containing all of your code
- But also the entire history of the project
- Can be local - where you work
- Or remote - where you share and backup

Init

`git init`

- Creates a new empty repository
- Adds ‘`.git`’ directory containing project configuration files

Commit

`git commit`

- Saves staged changes to your files
- Starts the default editor, such as vim
- Default editor can be changed
- A message must be provided, describing the changes made in the commit

Commit

`git commit -m “message”`

- Option ‘`-m`’ allows the commit message to be added on the command line
- Doesn’t launch the default editor
- Limited to 80 characters or less

Commit early and often

No regrets

Staging

- Relates to the state of changed files
- A staged file is about to be committed

Staging

`git add file`

- Relates to the state of changed files
- A staged file is about to be committed

- Files are staged by an add command

Staging

```
git add file
```

- Relates to the state of changed files
- A staged file is about to be committed

- Files are staged by an add command
- Multiple files can be staged
- Staged files reside in the cache

Staging

```
git add -p file
```

- The Option ‘-p’ invokes interactive mode
- Changes to files can be added in sections known as hunks

Staging

```
git add -p file
```

- The Option ‘-p’ invokes interactive mode
- Changes to files can be added in sections known as hunks

- Hunks can be added
- Skipped
- Or split into smaller hunks and then added

Reset

`git reset file`

- Files can also be removed from staging
- Resetting a file does not abandon changes
- Only stops the file being tracked

Status

`git status`

- The status of staged files is very useful
- Green files are staged
- Red files are not staged or are untracked

Stash

`git stash`

- Temporary store for uncommitted changes
- Useful when merging new changes
- Can hold multiple stashes
- Acts like a stack

Stash

`git stash pop`

- Applies the most recent stash to the working directory
- Removes the applied stash from the stack
- Performs automatic merge

Stash

`git stash list`

- Lists the stashes stored on the stack
- Displays the following, per stash
    - Stash id
    - Commit id
    - Commit message

Stash

```
git stash show -p stash@{0}
```

- Lists the stashes stored on the stack
- Displays the following, per stash
  - Stash id
  - Commit id
  - Commit message

- Display the stashed changes as a patch with the '-p' option

Stash

`git stash apply`

- Alternative to popping
- Uses the stash id
- Applies the stashed changes
- Does not remove the stash from the stack
- Useful when a merge conflict is expected

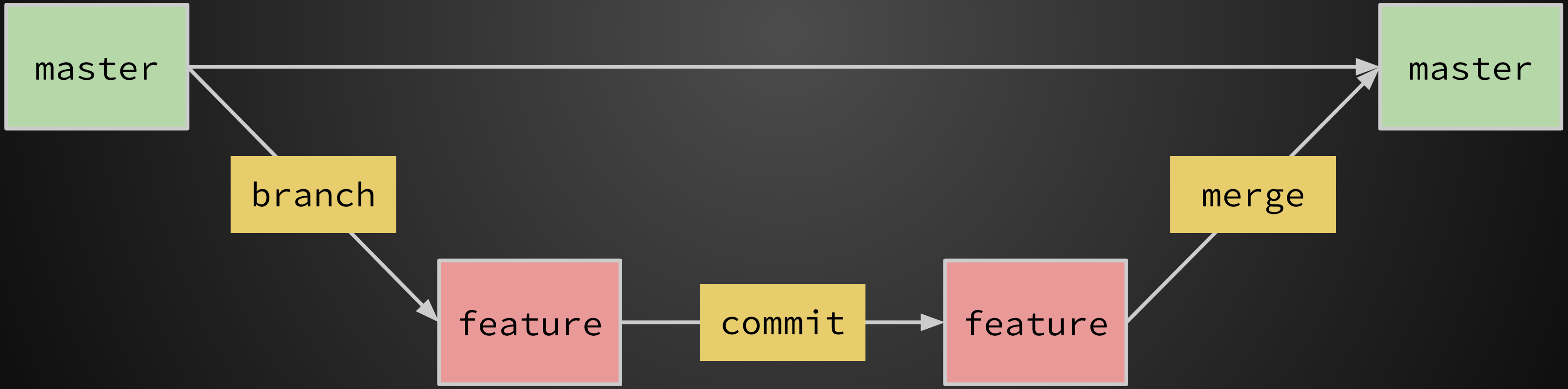
Log

`git log`

- Displays the history of the repository
- Shows the following, per commit
    - Commit id
    - Author
    - Date
    - Commit message

Branches

Branching enables separation of concerns.



Branches

`git branch`

- Calling branch with no arguments lists all available branches

Branches

```
git branch feature
```

- Calling branch with no arguments lists all available branches
- Providing ‘feature’ creates a new named branch

Branches

```
git branch -d feature
```

- Calling branch with no arguments lists all available branches
- Providing ‘feature’ creates a new named branch
- Deleting a branch is simple, but be careful not to lose unmerged work

Checkout

`git checkout`

- Changes the state of the repository to match a specified revision

Checkout

```
git checkout branch
```

- Changes the state of the repository to match a specified revision
- Switches between branches or commits

Checkout

`git checkout commit`

- Changes the state of the repository to match a specified revision

- Switches between branches or commits
- The HEAD of the repository is the currently checked out branch

Reset

- Undo changes made to the working directory

Reset

```
git reset --soft HEAD^
```

- Undo changes made to the working directory
- A soft reset reverses a commit, in this case the commit before HEAD denoted by the ‘`^`’ character

Reset

`git reset --soft HEAD^`

- Undo changes made to the working directory
- A soft reset reverses a commit, in this case the commit before HEAD denoted by the ‘`^`’ character
- Soft resets are safe, you will not lose changes to your files

Reset

```
git reset --hard HEAD
```

- Undo changes made to the working directory
- Hard resets are **NOT SAFE**, but can be useful

Reset

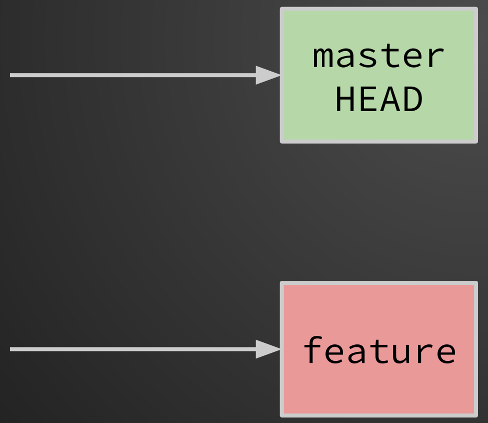
`git reset --hard HEAD`

- Undo changes made to the working directory
- Hard resets are **NOT SAFE**, but can be useful

- All changes in the working directory will be nuked
- The state will be exactly the same as the HEAD commit

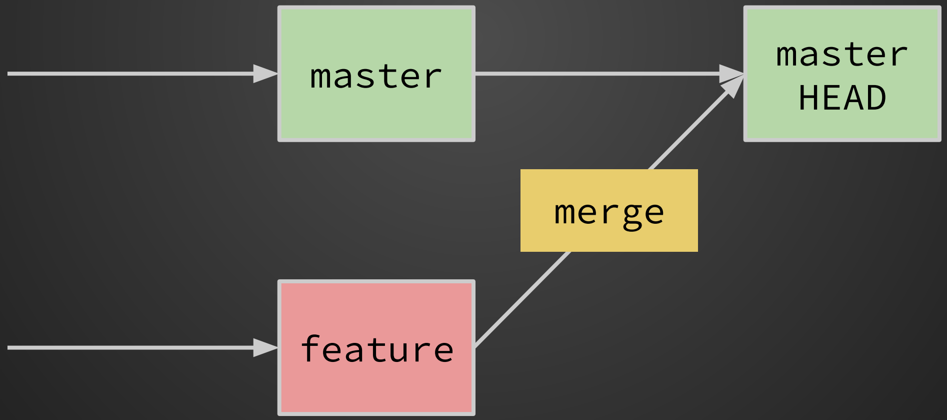
Merging

- Combining two branches into a single branch



Merging

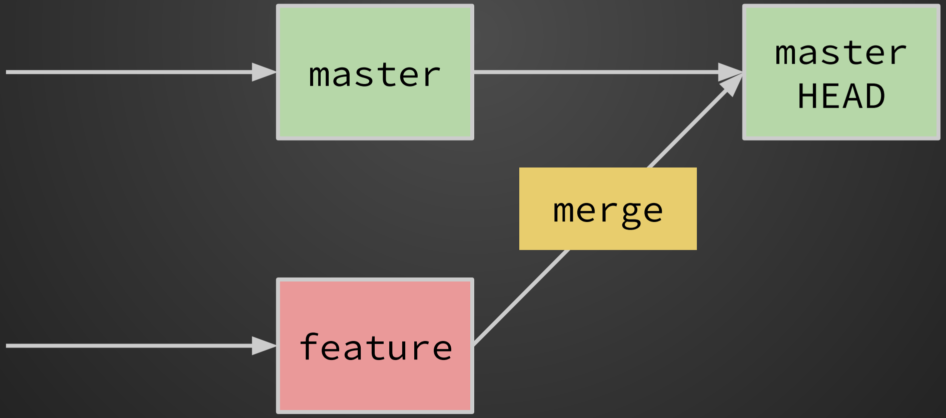
- Combining two branches into a single branch



Merging

`git merge feature`

- Merge the feature branch into master



Conflicts

- Files are conflicting when changes have been made to the same line of the same file in one or more branches
- This is common when working in a team
- Conflicts need manual intervention

Merge tool

`git mergetool file`

- The merge tool command invokes an external merging tool
- Tool may require configuration
- An example of a visual merge tool is Meld, which runs on Linux, Mac, and Windows
    - http://meldmerge.org/

Diff

`git diff`

- Differential of all the files in the source tree

Diff

```
git diff --cached
```

- Differential of all the files in the source tree
- The diff of staged files can be viewed with the cached argument

Diff

```
git diff file
```

- Differential of all the files in the source tree
- The diff of staged files can be viewed with the cached argument
- Can be called on individual files

Patches

```
git diff > a.patch
```

- Patches are created from a diff and saved to a file
- A patch is another way of storing changes to a repository

Patches

```
git apply a.patch
```

- Patches are created from a diff and saved to a file
- A patch is another way of storing changes to a repository

- Patches can then be applied
- Useful when you don’t have write access to a repository

Revert

```
git revert commit
```

- Sometimes changes don’t work
- Revert these changes using the commit id of the offending changes

Revert

`git revert commit`

- Sometimes changes don't work
- Revert these changes using the commit id of the offending changes

- This creates a new commit with the original changes removed
- Reverts can be reverted

Clean

`git clean path`

- Removes files which are not staged from the supplied path

Clean

`git clean -x path`

- Removes files which are not tracked from the repository
- Supplying the ‘-x’ option also removes ignored files

Blame

`git blame file`

- Displays which author last changed each line in a file

Blame

`git blame -Ln,m file`

- Displays which author last changed each line in a file
- Providing ‘`-Ln,m`’ limits the output and shows only the lines where ‘`n`’ and ‘`m`’ represent an increasing range of line numbers

Remote Repositories

- Until now everything we have covered can be applied to locally

Remote Repositories

- Until now everything we have covered can be applied to locally
- This is vulnerable to data loss

Remote Repositories

- Until now everything we have covered can be applied to repositories locally
- This is vulnerable to data loss
- Hard to share your project

Remote Repositories

- A git repository can point to multiple remote repositories
- These are referred to as ‘remotes’

Remote Repositories

- A git repository can point to multiple remote repositories
- These are referred to as ‘remotes’

- Most common remote is called ‘origin’
- The origin is automatically generated when a remote repository is cloned

Clone

```
git clone git@server.net
```

- Takes a copy of an existing repository
- Usually a remote repository
- Automatically checks out the most recent commit

Clone

```
git clone git@server.net dir
```

- Takes a copy of an existing repository
- Usually a remote repository
- Automatically checks out the most recent commit
- Destination directory can be specified

Add a remote URL

- To add a remote repository to an existing local repository use the following command

```
git remote add --track origin master git@server.net
```

Add a remote URL

- To add a remote repository to an existing local repository use the following command
- By changing ‘origin’ to ‘other’ we add another remote to the repository

```
git remote add --track other master git@server.net
```

Pull

```
git pull origin branch
```

- Grabs the new changes from the remote branch
- Attempts to merge them into your local branch

Pull

```
git pull origin branch
```

- Grabs the new changes from the remote branch
- Attempts to merge them into your local branch

- It is usual to merge remote and local branches of the same name

Push

```
git push origin branch
```

- Uploads any local changes to the remote repository
- Creates a new remote branch if one doesn’t already exist

Push

```
git push origin :branch
```

- Uploads any local changes to the remote repository
- Creates a new remote branch if one doesn’t already exist
- To delete a remote branch add a ‘:’ to the start of the branch name

Fetch

`git fetch`

- Download the latest commits from the remote repository
- Differs from pull because an implicit merge is not performed

Submodules `git submodule add other-repo`

- It is possible to add another repository to your own using submodules
- This is most useful when you want to eliminate your project external dependencies

Submodules

```
git clone --recursive repo
```

- It is possible to add another repository to your own using submodules
- This is most useful when you want to eliminate your project external dependencies

- When cloning a repository with submodules it is useful do a recursively clone

Rebase

- To port local commits to an updated remote commit tree
- Rebasing is NOT SAFE unless the commit trees are related
- It is possible to break a repository if a rebase is done incorrectly

Questions

Links

- Here you will find the practical section
  - https://github.com/kbenzie/git-workshop

- These are useful references
  - http://git-scm.com/book/
  - http://nvie.com/posts/a-successful-git-branching-model/