Git

Beyond the Basics

Foivos Zakkak

foivos.zakkak@manchester.ac.uk

Preamble

Feel free to interrupt if you don't understand something

Please ask. There are no stupid questions!

The slides are available at
`https://foivos.zakkak.net/uploads/git-beyond-the-basics.pdf`

Apologies in advance if I inadvertently offend you in any way

foivos.zakkak@manchester.ac.uk

Assumptions

You know how to:

1. create a new repository — `git init`
2. clone — `git clone <path.to.git.repository>`
3. pull — `git pull`
4. commit — `git add <file>; git commit`
5. push — `git push`
6. use branches — `git branch mybranch; git checkout mybranch`

foivos.zakkak@manchester.ac.uk

What Is This Tutorial About?

Time traveling	visiting older states, undoing mistakes
"Blaming" others	detecting commits that introduced bugs
Gardening	working with multiple branches
Cleaning after yourself	rebasing, squash, fixup, editing logs
Interacting with people	working with others
Being social	working with multiple remotes

Best practices in general!

foivos.zakkak@manchester.ac.uk

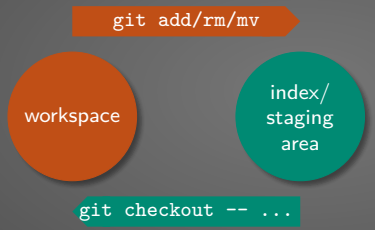
Git Terminology & Background

Visualization inspired by Patrick Zahnd's "*git data transport commands*". (`http://www.patrickzahnd.ch`)

foivos.zakkak@manchester.ac.uk

Git Terminology & Background

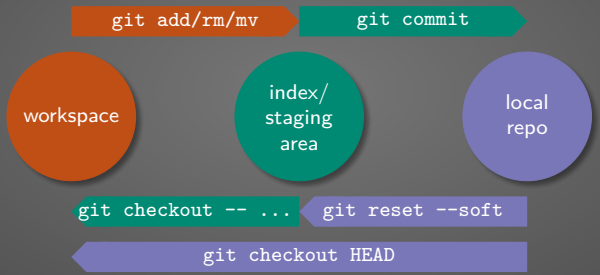
Visualization inspired by Patrick Zahnd's "*git data transport commands*". (`http://www.patrickzahnd.ch`)



foivos.zakkak@manchester.ac.uk

Git Terminology & Background

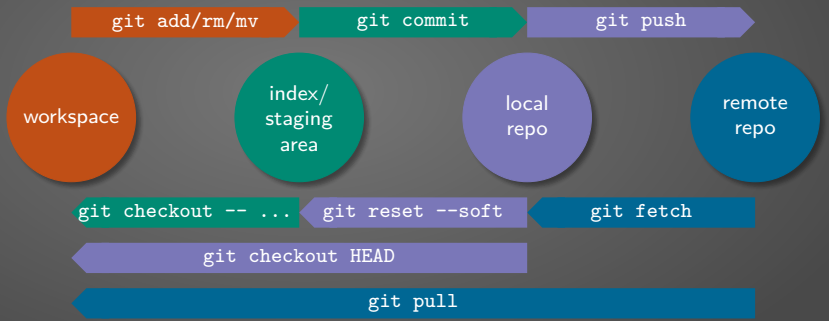
Visualization inspired by Patrick Zahnd's "*git data transport commands*". (`http://www.patrickzahnd.ch`)



foivos.zakkak@manchester.ac.uk

Git Terminology & Background

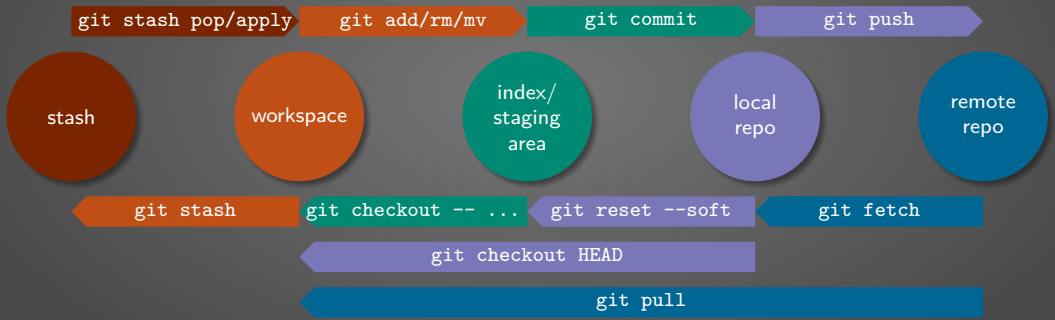
Visualization inspired by Patrick Zahnd's "*git data transport commands*". (`http://www.patrickzahnd.ch`)



foivos.zakkak@manchester.ac.uk

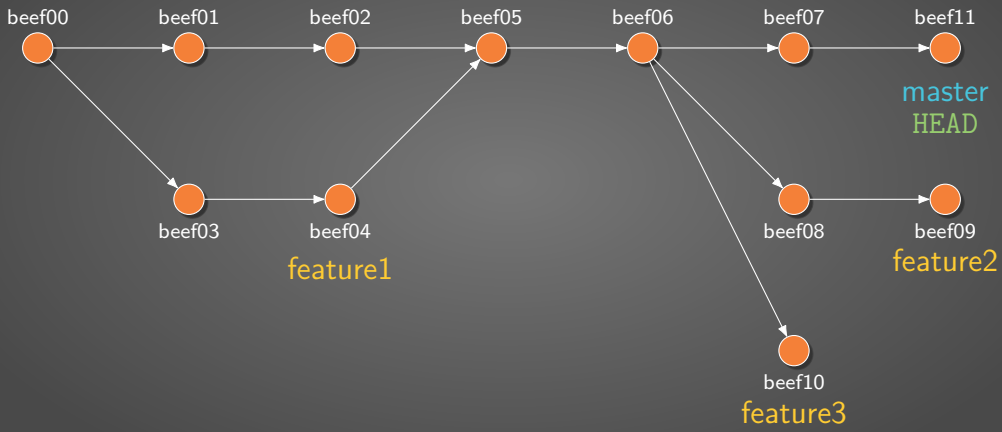
Git Terminology & Background

Visualization inspired by Patrick Zahnd's "*git data transport commands*". (`http://www.patrickzahnd.ch`)



foivos.zakkak@manchester.ac.uk

Git log Visualization



foivos.zakkak@manchester.ac.uk

Checkout: Jump to a certain point in time (commit)

```
git checkout <commit>
```

- Brings the state of `<commit>` in the workspace
- Keeps any changes if there are no conflicts, fails otherwise
- Resets index
- Detaches `HEAD`

foivos.zakkak@manchester.ac.uk

Checkout: Undo changes in specific file(s)

```
git checkout -- <path(s)>
```

Get (overwrite) <path(s)> from index

If path has been staged it will revert to the staged state

foivos.zakkak@manchester.ac.uk

Checkout: Obtain specific version of specific file(s)

```
git checkout <branch/commit> -- <path(s)>
```

Get (overwrite) <path(s)> from <branch/commit>

foivos.zakkak@manchester.ac.uk

Reset: Undo staging

`git reset`

Resets the staging area (not altering the files in the workspace)

foivos.zakkak@manchester.ac.uk

Reset: Undo staging

`git reset`

Resets the staging area (not altering the files in the workspace)

`git reset -- <path(s)>`

Only affect `<path(s)>`

foivos.zakkak@manchester.ac.uk

Reset: Move HEAD to another branch/commit

```
git reset <branch/commit>
```

Point HEAD to <branch/commit>

--mixed (the default) change the HEAD and reset staging area

foivos.zakkak@manchester.ac.uk

Reset: Move HEAD to another branch/commit

```
git reset <branch/commit>
```

Point HEAD to <branch/commit>

`--mixed` (the default) change the HEAD and reset staging area

`--soft` only change the HEAD
(Undo last commits without losing changes)

foivos.zakkak@manchester.ac.uk

Reset: Move HEAD to another branch/commit

```
git reset <branch/commit>
```

Point HEAD to <branch/commit>

`--mixed` (the default) change the HEAD and reset staging area

`--soft` only change the HEAD
(Undo last commits without losing changes)

`--hard` change the HEAD, reset staging area and workspace
(Throw away last commits)

foivos.zakkak@manchester.ac.uk

Reset: Move HEAD to another branch/commit

```
git reset <branch/commit>
```

Point HEAD to <branch/commit>

`--mixed`	(the default) change the HEAD and reset staging area
`--soft`	only change the HEAD (Undo last commits without losing changes)
`--hard`	change the HEAD, reset staging area and workspace (Throw away last commits)
`--merge`	See `man git reset`

foivos.zakkak@manchester.ac.uk

Reset: Move HEAD to another branch/commit

```
git reset <branch/commit>
```

Point HEAD to <branch/commit>

`--mixed`	(the default) change the HEAD and reset staging area
`--soft`	only change the HEAD (Undo last commits without losing changes)
`--hard`	change the HEAD, reset staging area and workspace (Throw away last commits)
`--merge`	See `man git reset`
`--keep`	See `man git reset`

foivos.zakkak@manchester.ac.uk

Undo changes old changes

```
git revert <commit>
```

Creates a new commit that reverts the changes made by <commit>

foivos.zakkak@manchester.ac.uk

Undo changes old changes

```
git revert <commit>
```

Creates a new commit that reverts the changes made by <commit>

```
git revert <commit1>..<commit2>
```

Creates a new commit that reverts the changes made by all commits in the range (inclusive) <commit1> to <commit2>

foivos.zakkak@manchester.ac.uk

Undo changes old changes

`git revert <commit>`

Creates a new commit that reverts the changes made by <commit>

`git revert <commit1>..<commit2>`

Creates a new commit that reverts the changes made by all commits in the range (inclusive) <commit1> to <commit2>

`git revert -n <commit1>..<commit2>`

Reverts the changes made by all commits in the range (inclusive) <commit1> to <commit2> but does not commit them

foivos.zakkak@manchester.ac.uk

Debugging or Bisecting

```
git bisect start
```

Enter the bisecting mode

foivos.zakkak@manchester.ac.uk

Debugging or Bisecting

`git bisect start`

Enter the bisecting mode

`git bisect good/bad`

Mark current commit as good or bad

foivos.zakkak@manchester.ac.uk

Debugging or Bisecting

`git bisect start`

Enter the bisecting mode

`git bisect good/bad`

Mark current commit as good or bad

`git bisect good/bad <commit>`

Mark `<commit>` as good or bad

foivos.zakkak@manchester.ac.uk

Debugging or Bisecting

`git bisect start`

Enter the bisecting mode

`git bisect good/bad`

Mark current commit as good or bad

`git bisect good/bad <commit>`

Mark `<commit>` as good or bad

`git bisect log`

Show tested commits and their status. This log can be saved and used to *replay* part of the process. `git bisect replay <logfile>`

foivos.zakkak@manchester.ac.uk

Branching

```
git branch -a
```

List all branches, local and remote-tracking

foivos.zakkak@manchester.ac.uk

Branching

```
git branch -a
```

List all branches, local and remote-tracking

```
git checkout -b <newbranch>
```

Fastest way to create and checkout a new branch

foivos.zakkak@manchester.ac.uk

Branching

```
git branch -a
```

List all branches, local and remote-tracking

```
git checkout -b <newbranch>
```

Fastest way to create and checkout a new branch

```
git branch -d <branchname>
```

Delete <branchname> locally

foivos.zakkak@manchester.ac.uk

Branching

`git branch -a`

List all branches, local and remote-tracking

`git checkout -b <newbranch>`

Fastest way to create and checkout a new branch

`git branch -d <branchname>`

Delete <branchname> locally

`git branch -m <newname>` or `git branch -m <oldname> <newname>`

Rename current branch or given branch

foivos.zakkak@manchester.ac.uk

Branching

`git branch -a`

List all branches, local and remote-tracking

`git checkout -b <newbranch>`

Fastest way to create and checkout a new branch

`git branch -d <branchname>`

Delete <branchname> locally

`git branch -m <newname>` or `git branch -m <oldname> <newname>`

Rename current branch or given branch

`git branch -u <remote> <remotebranchname>`

Set the remote branch to be used as upstream for current branch

foivos.zakkak@manchester.ac.uk

Merging

```
git merge <branch>
```

Merge <branch> with current branch

--ff (the default) if merging can be resolved as an append doesn't create a *merge commit*

foivos.zakkak@manchester.ac.uk

Merging

`git merge <branch>`

Merge `<branch>` with current branch

`--ff` (the default) if merging can be resolved as an append doesn't create a *merge commit*

`--ff-only` if merging cannot be resolved as an append it fails

foivos.zakkak@manchester.ac.uk

Merging

```
git merge <branch>
```

Merge <branch> with current branch

`--ff` (the default) if merging can be resolved as an append doesn't create a *merge commit*

`--ff-only` if merging cannot be resolved as an append it fails

`--squash` squash all changes and stage them but do not commit

foivos.zakkak@manchester.ac.uk

Rebasing

```
git rebase <branch>
```

Replay current branch's commits on top of `<branch>`

foivos.zakkak@manchester.ac.uk

Rebasing

```
git rebase <branch>
```

Replay current branch's commits on top of `<branch>`

Caution

Re-writes the history of your branch, changing the commit hashes

foivos.zakkak@manchester.ac.uk

Re-writing history

```
git rebase -i <commit>
```

Opens a file with all commits from `HEAD` to `<commit>` (inclusive) and allows us to:

drop	reorder	edit commit
squash	fixup	edit commit message

Caution

Re-writes the history of your branch, changing the commit hashes

foivos.zakkak@manchester.ac.uk

Fixup and Squash commits

```
git commit --fixup/--squash <commit>
```

Mark commit as a fixup or squash of `<commit>`. Commit message will automatically be set.

foivos.zakkak@manchester.ac.uk

Fixup and Squash commits

```
git commit --fixup/--squash <commit>
```

Mark commit as a fixup or squash of `<commit>`. Commit message will automatically be set.

```
git rebase -i --autosquash
```

Automatically squash commits starting with `fixup!` or `squash!`

foivos.zakkak@manchester.ac.uk

Rules for successfull collaboration

Use descriptive yet short commit subjects

foivos.zakkak@manchester.ac.uk

Rules for successfull collaboration

Use descriptive yet short commit subjects

Never rewrite history of shared branches

foivos.zakkak@manchester.ac.uk

Rules for successfull collaboration

Use descriptive yet short commit subjects

Never rewrite history of shared branches

Merge often

foivos.zakkak@manchester.ac.uk

Rules for successfull collaboration

Use descriptive yet short commit subjects

Never rewrite history of shared branches

Merge often

Do not commit changes that brake the previous state

foivos.zakkak@manchester.ac.uk

Rules for successfull collaboration

Use descriptive yet short commit subjects

Never rewrite history of shared branches

Merge often

Do not commit changes that brake the previous state

Keep commits self contained and as small as possible

foivos.zakkak@manchester.ac.uk

Managing multiple remotes

```
git remote add <remotename> <url>
```

Adds a new remote

foivos.zakkak@manchester.ac.uk

Managing multiple remotes

```
git remote add <remotename> <url>
```

Adds a new remote

```
git push <remotename>
```

Pushes to `<remotename>` instead of `origin`

foivos.zakkak@manchester.ac.uk

Managing multiple remotes

```
git remote add <remotename> <url>
```

Adds a new remote

```
git push <remotename>
```

Pushes to <remotename> instead of origin

```
git pull <remotename>
```

Pulls from <remotename> instead of origin

foivos.zakkak@manchester.ac.uk

Managing multiple remotes

```
git remote add <remotename> <url>
```
Adds a new remote

```
git push <remotename>
```
Pushes to <remotename> instead of origin

```
git pull <remotename>
```
Pulls from <remotename> instead of origin

```
git fetch <remotename>
```
Fetches from <remotename> instead of origin

foivos.zakkak@manchester.ac.uk

Managing multiple remotes

```
git checkout <remotename>/<branch>
```

Checks out `<branch>` from `<remotename>` instead of the local repository. It still doesn't fetch it though!

foivos.zakkak@manchester.ac.uk

Managing multiple remotes

```
git checkout <remotename>/<branch>
```

Checks out <branch> from <remotename> instead of the local repository. It still doesn't fetch it though!

```
git fetch --all
```

Fetches all remotes

foivos.zakkak@manchester.ac.uk

Managing multiple remotes

```
git checkout <remotename>/<branch>
```

Checks out <branch> from <remotename> instead of the local repository. It still doesn't fetch it though!

```
git fetch --all
```

Fetches all remotes

```
git fetch --all --prune
```

Removes any no longer existing remote branches

foivos.zakkak@manchester.ac.uk

Cheers!

Git

Beyond the Basics

Foivos Zakkak

foivos.zakkak@manchester.ac.uk