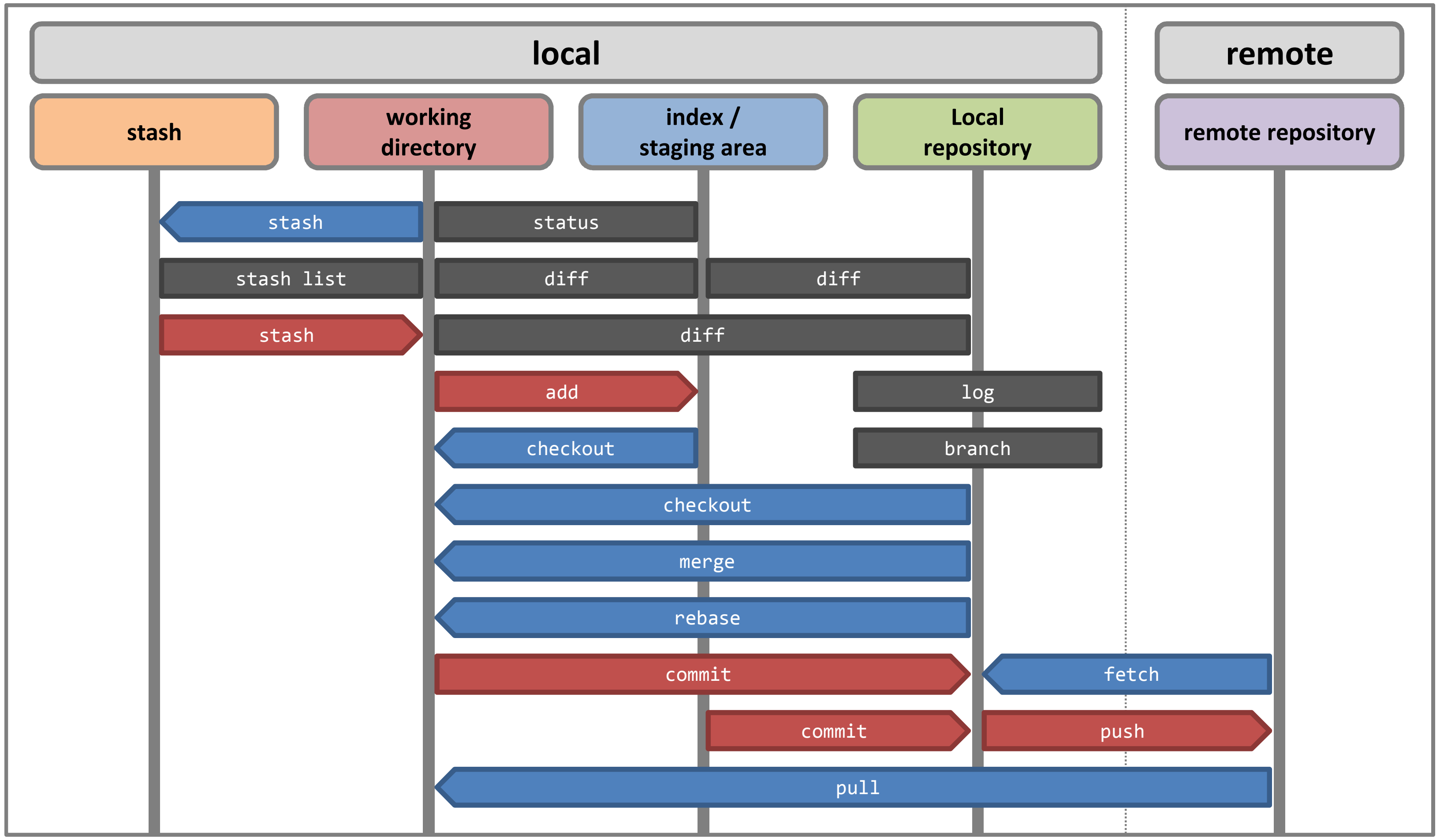
git Cheatsheet

version 1.0



Space for Notes

Configuration

List options

```
  git config -l [PLACE]
```

Show option

```
 git config [PLACE] <KEY>
```

Change options

```
 git config [PLACE] <KEY> <VALUE>
```

--system is stored in /etc/gitconfig
--global is stored in ~/.gitconfig
--local is stored in repository under .git/config

user (used in commits)

```
 user.name <NAME>
 user.email <EMAIL>
```

color

```
 color.ui auto
```

windows

```
 core.autocrlf true
```

alias

```
 alias.[ALIAS] [ORIGINAL]
```

more:

```
 git help config
```

Ignoring Files

Ignore Files

Some .gitignore pattern/naming rules:

- Blank lines are ignored
- Lines beginning with ‘#’ are ignored
- Standard glob patterns work
- Can specify root directory files like ‘/TODO’
- End patterns with a slash to specify a dir (bin/)
- Negate a pattern by beginning it with an ‘!’

Option 1: Edit .git/info/exclude. This won’t be cloned.

Option 2: Add a file .gitignore to the root of your project. This file will be tracked if not ignored.

Object References

master	default development branch
origin	default upstream branch
HEAD	current branch
HEAD^	parent of HEAD
HEAD~4	great, great grandparent
ID1..ID2	from ID1 to ID2

git Cheatsheet

Create

From existing data
```
cd ~/myproject
git init
git add .
```
except .git-directory

From existing repository
```
git clone ~/existing/repo ~/new/repo
git clone you@example.com/project.git
```
default protocol is **ssh**

Browse

Files changed in working directory
```
git status
```
Show changes to tracked files
```
git diff
```
Show changes between ID1 and ID2
```
git diff <ID1> <ID2>
```
History of <NUMBER> changes
```
git log [-<NUMBER>]
```
Show graphical log
```
git log --graph --pretty=oneline
        --abbrev-commit
```
Who changed what and when in a file
```
git blame <FILE>
```
Show commits which affected the file
```
git whatchanged <FILE>
```
A commit identified by ID
```
git show <ID>
```
A specific file from a specific ID
```
git diff <ID>:<FILE>
```
Search for patterns
```
git grep <PATTERN> [PATH]
```
List all branches (* = current)
```
git branch -a
```
Show current branch
```
git show-branch
```
List all tags
```
git tag -l
```
Show last few actions
```
git reflog
```

Always remember:
'git help [COMMAND]' or start with 'git help git'

Track Files

Add files to the index
```
git add <FILES>
```
Move or rename a file, directory or symlink
```
git mv <SOURCE> <DESTINATION>
```
Removes files from the working tree and index
```
git rm <FILES>
```
Removes files from the index
```
git rm --cached <FILES>
```

Update

Fetch changes from origin or another remote
```
git fetch [<REMOTE>]
```
this does not merge them

Pull changes from origin or another remote
```
git pull [<REMOTE>]
```
does a fetch followed by a merge

Apply a patch that someone sent you
```
git am -3 patch.mbox
```
In case of conflict:
resolve the conflict and use: git am --resolve

Stashing

Temporarily set aside changes onto a stack
```
git stash save
```
List current stashes
```
git stash list
```
Get stash <NAME>
```
git stash apply stash@(<NAME>)
```
Pops the last stash
```
git stash pop
```
Clear all stashes
```
git stash clear
```
Deletes stash <NAME>
```
git stash drop stash@(<NAME>)
```

Remotes

List all remotes
```
git remote [-v]
```
Register a new remote repository
```
git remote add <NAME> <URL>
```
Remove a remote repository
```
git remote rm <NAME>
```
Track branches for lazy push and pull
```
git branch --track [LOCAL] [REMOTE]
```

Branch

Switch to a branch
```
git checkout <BRANCH>
```
Merge BRANCH into current branch
```
git merge <BRANCH>
```
Create branch based on current branch
```
git branch <BRANCH>
```
Create branch based on another
```
git checkout <NEW> <BASE>
```
Delete a branch
```
git branch -d <BRANCH>
```
Delete remote branch
```
git push <REMOTE> :<BRANCH>
```

Resolve Conflicts

View merge conflicts
```
git diff
git log –merge
gitk --merge
```
View merge conflicts against base file
```
git diff --base <FILE>
```
View merge conflicts against other changes
```
git diff --theirs <FILE>
```
View merge conflicts against your changes
```
git diff --ours <FILE>
```
After resolving conflicts, merge with
```
git add <CONFLICTING_FILE>
git rebase --continue
```
Discard conflicting patch
```
git reset --hard
git rebase --skip
```

Revert

Return to the last committed state
```
git checkout –f
git reset --hard
```
you cannot undo a hard reset

Revert the last commit
```
git revert HEAD
```
Creates a new commit

Revert specific commit
```
git revert <ID>
```
Creates a new commit

Replace previous commit
```
git commit -a --amend
```
after editing the broken files

Checkout the ID version of a file
```
git checkout <ID> <FILE>
```

Commit

In Git, commit only respects changes that have been marked explicity with add.

Commit all local changes
```
git commit
```

Options:
```
–a: skip index and commit all changes
–m '<MSG>': specify commit message
```
if this isn't passed an editor opens

COMMIT MESSAGES

Some of Git's viewing tools need commit messages in the following format:
```
A brief one-line summary (50 chars).
<blank line>
More details about the commit.
```

Publish

Prepare a patch for other developers
```
git format-patch [BRANCH]
```
Push changes to origin or remote
```
git push [REMOTE] [BRANCH]
```
Make a version or milestone
```
git tag <VERSION_NAME>
```

Other Useful Commands

Create release tarball
```
git archive
```
Binary search for defects
```
git bisect
```
Take single commit from elsewhere
```
git cherry-pick
```
Check tree
```
git fsck
```
Compress metadata (performance)
```
git gc
```
Forward-port local changes to remote branch
```
git rebase <BRANCH>
```
Do not rebase commits that you
have pushed to a public repository!