**Git workshop**

**Bash Commands**

**Notes**

* Always use bash completion to save yourself from typing more than you need to.**Example:** type less REA then hit the tab key to complete it to less README.txt
* Use the up arrow to run commands again to save more typing.

cd {dirname} **→** change directory to {dirname}

cd .. **→** change directory to the parent directory (go up one directory)

ls **→** list the files in the current directory

rm {filename} **→** remove (delete) a file called {filename}

mv {filename} **→** move or rename a file called {filename}

mkdir {dirname} **→** make directory {dirname}

rmdir {dirname} **→** remove directory {dirname}

rm -r {dirname} **→** remove directory {dirname} and all of its contents (recursively)

less {filename} **→** look at the contents of {filename} (press arrows to go up and down, press "q" to quit)

clear or **control+L** **→** clear the terminal.

Git config --global color.ui true

Git config --global merge.conflictstyle diff3

git config --global core.editor notepad

git config --global alias.l “log --graph --decorate --oneline --all --date-order”

**Editing a File in Unix (Linux or Mac OSX)**

nano {filename} **→** use arrows to move, Ctrl-O, enter, Ctrl-X to save and exit

**Editing a File in Windows**

notepad {filename} **→** save and quit to get back to the terminal

**Setup**

mkdir workshop**→** this is where our project will be stored

cd workshop**→** enter the directory

git init **→** initialize the current directory as a new git repository

ls -a **→** see the new .git directory

**Basics**

git status **→** show the current status of the git repository

git log **→** show the project's commits, most recent until least recent

git config --global alias.l “log --graph --decorate --oneline --all --date-order” **→** use git l to make very useful graph logs

git l **→** we made an alias for this earlier, show the entire tree of the repository in order of time (most recent to least recent). Shows the hashes for each commit at the beginning of each line. "HEAD" is the current commit which your repository is pointing to, meaning that your files will look like HEADs files, except for the changes you have which are either staged or not staged (git status shows these changes).

**Committing**

git add {filename} **→** stage {filename}'s changes, or add {filename} as a new file.

git add -A **→** stages All

git commit -a -m 'message' **→** add a commit of all changes in the current directory with a message

git reset {filename} **→** unstage {filename}

git reset **→** unstage all staged files

git checkout . **→** discard unstaged changes

**Branches**

git branch {branch} **→** create a branch called {branch}

git checkout {branch} **→** checkout a branch. Your files will now look like they did in the commit which {branch} points to. Make sure your working directory is clean before switching branches (nothing staged or unstaged).

git checkout -b {branch} **→** create {branch} and check it out at the same time.

**Reset, Stash**

git reset --hard {commitSHA} **→** reset current branch to the given {commitSHA} and **delete** all changes

git reset --soft {commitSHA} **→** reset current branch to the {commitSHA} and keep all changes (will appear as uncommitted files)

git stash save {stashName} **→** stash all changes that are tracked: staged and unstaged files (**new files added will not be stashed if they are not tracked: git add -A**)

git stash list **→** list all stashes

git stash apply stash@{0} **→** apply stash located at position 0

git stash drop stash@{0} **→** delete stash located at position 0

**Rebasing - workflow**

git checkout {feature} **→** move to the branch that you want to rebase

git rebase {master} **→** rebase current branch to the given branch name

git rebase add –A **→** if there are conflicts, fix the change and then stage all changes

git rebase –continue **→** after adding all files continue the rebase.

git rebase –abort **→** cancels the rebase

git rebase -i {master} **→** rebase interactive and choose which commits will be applied, or skipped, squashed..

**Merging**

**Notes**

* If only one branch has changes on it, it is a "fast-forward" commit.
* If there are changes to both in the same line of a file, then there can be a "merge conflict"
* During a merge conflict, diff3 shows the local file, the merging file, and the common ancestor (very useful). It looks like this:
* <<<<<<<  
  changes made on my branch  
  |||||||  
  the common ancestor version  
  =======  
  changes made on the branch which is being merged in  
  >>>>>>>
* To resolve a merge conflict, get rid of the <<<, ||||, ====, and >>>> lines while resolving the code how it should be.

Then use git commit

git merge {branch} **→** combine two branches (the current branch, and {branch}). The current branch will be automatically updated to point to the new merge commit.

**Remote Repositories**

git clone {url} **→** clone a remote repository into its own directory under the current directory

git pull or git pull origin {branch} **→** fetch all changes from the server and automatically update fast-forwardable changes from the default remote repository into the current branch or into {branch}.

git fetch origin {branch} **→** bring in changes, but do not fast-forward anything. You can later use git pull to fast-forward them.

git push origin **→** push all of your branches to the remote repository

git push origin {branch} **→** push {branch} to the remote repository

git push origin --delete {branch} **→** delete a branch from the remote repository. WARNING: make sure it's merged into another branch first, or all of its commits will disappear.

**Tags**

**Notes**

* Mostly used for version numbers.
* Don't have branches, tags, or files which share the same name. It makes things complicated.
* Like branches, but more permanently associated with a specific commit.
* Can be used almost anywhere {branch} or {hash} can.

git tag **→** show tags

git tag -a {tag} **→** add a tag at the current commit.

git tag -d {tag} **→** delete {tag}.

git push --tags origin **→** push your tags to the server.

**Recommended Workflow**

**Notes**

* master branch is what everyone else looks at and uses.
* Create your own branch, and work exclusively on it. You can create as many branches as you need.
* If you end up working on the wrong branch, don't panic. See git help stash or cherry-pick.

Try not to commit when your code does not compile. If you must, make it known in the commit message. **Do not** push non-compiling commits to master.

**To be done often**

git pull origin

git rebase master **→** Only run this if your branch is not being used by anyone else, or if all your changes on that branch are local. If this has any merge conflicts at all, run git rebase --abort and use git merge master instead.

**To be done when you have something for other people**

The things in the previous section (pull, rebase/merge)

Test your code to make sure it does **exactly** what you want.

git checkout master

git merge { branch } **→** Where { branch } is your working branch.

git push origin master **→** If you've made changes to the master branch.

git push origin { branch }