Supplementary Materials and Methods

Plant materials and trait measurements

To measure traits on field-grown plants, the 226 DH lines and the two parents were evaluated for awns, grain yield, grain protein content, heading, plant height, thousand grain weight, spike length, and lodging susceptibility. Plants were grown between May and September 2017 at two locations: Elora, Ontario and Belwood, Ontario. Each field trial was conducted as a randomized complete block design with two blocks at each location. Each entry was planted in 1 m x 1.43 m plots with six rows and 20 cm row spacing. All field trials were conducted in rainfed conditions using standard agronomic and cultural practices. Heading date was recorded when 50% of spikes in a plot had completely emerged out of the flag leaf. For plant height, we measured three randomly selected plants from each plot and measured the distance from the ground to the end of the spike after heading. Spike length was measured as the length of the inflorescence on the same three plants. Awns were scored visually. Grain was harvested by a Wintersteiger 11 combine and measured by hand, and lodging susceptibility (percent of lodged plants in plot) was measured at harvest. Grain was cleaned and weighed, and protein concentration (%) was determined on whole grain from individual plots using a FOSS-DS 2500 Near-Infrared Reflectance Spectrophotometer (Foss, Hillord, Denmark).

To obtain tissue for RNA from the lines, we grew 154 DHs that were randomly selected from the 226 lines, and we sampled the 2nd leaves from the base of 12 day-old growth room plants for RNA extraction. Leaves from plants grown across three replicates in time were pooled into a single sample. Red Fife and Stettler were represented by four samples, and each other DH line was represented by one sample. All lines were planted in six replicates over time between May and August 2016. For each replication, two seed of a single genotype were randomly assigned to a single pot. The plants were grown in 50/50 sungro horticulture professional growing mix and Turface. The growth room's temperature was 21ºC/18ºC day/night, with a 16 h photoperiod. The room had a relative humidity of 60%, and a light intensity of 150 – 170 $u$mol.m^{-2}.s^{-1}. Plants were watered every three days and watered two days before any measurement.

Statistical analyses of trait data

We evaluated genotypic values of field grown plants using the model:

$$y_{ijk} = u + L_j + R_{k(j)} + G_i + e_{ijk}$$

Where: y_{ijk} denotes the value of the trait for genotype i in the *k*th replication within the *j*th location. The term μ is the grand mean, L_j is the random location effect, $R_{k(j)}$ is the random replication (block) effect nested in location, G_i is the fixed genotype effect, and e_{ijk} is the residual.

For the field data broad sense heritability, we used

$$h^2 = \frac{\sigma_g^2}{\sigma_g^2 + \frac{\sigma_{gl}^2}{l} + \frac{\sigma_e^2}{lr}}$$

Where l is the number of locations (n=2) and r is the number of replicates per location (n=2). We used the SAS Varcomp function to estimate variances. Genotype least square means, F statistics and heritability were obtained using PROC MIXED SAS version 9.3 (SAS Institute Inc. Cary, USA). Genotypic estimates were correlated with PROC CORR.

RNA Sequencing, Read Processing, and SNP Calling

Total RNA from pooled replicates for each genotype's sample was extracted from each sample using an in-house Trizol-based method. Poly-A containing mRNA molecules were purified and libraries constructed according to the Illumina TruSeq RNA sample preparation guide v2. RNA-seq was performed on each sample using the HiSeq 2500 platform at the University Health Network Clinical Genomics lab in Toronto, Ontario, generating 20-60 million 100- 125 nt long paired end reads per sample.

Read quality was assessed with FastQC. Low-quality nucleotides were trimmed and short (<75bp) sequences were removed using Trim Galore (Krueger, 2015). Reads were mapped to the Chinese Spring wheat reference genome (iwgsc_refseqv1.0; IWGSC 2018) using STAR version 2.5.2b (Dobin *et al.*, 2013). A two-pass STAR alignment that identifies unannotated splice junctions in the first iteration was performed. STAR mapping allowed two nucleotide mismatches between a read and the reference. Mapped reads were removed if both reads of a pair did not map. Picard was used to mark duplicate reads, and only a single mapped read was kept.

SAMtools v1.3.1 mpileup (Li *et al.*, 2009) was used to call bases from reads mapped to the reference genome for each of the four parental replicates. SAMtools bcftools was used to remove genome positions with a read coverage less than 3, and VCFtools (v0.1.13) (Danecek et al., 2011) removed indels. To call a putative SNP from a single replicate's alignment to Chinese Spring, we required a low probability of detecting a false positive allele (QUAL=20) and a genotype (GT) designation of 1:1. To identify SNP markers, we analyzed putative SNPs across the four replicate alignments of each parent to Chinese Spring. We called SNPs between Red Fife and Chinese Spring and Stettler and Chinese Spring when putative SNPs were called in three of the four parental replicates. These SNPs were used to identify 22,378 SNPs between Red Fife and Stettler. The SRA accession number for RNASeq reads is SUB7283627. These SNP markers were used to query the 154 RNA-seq alignments from the DH population. Detailed protocols and code are provided at the end of this text.

Marker filtering and genetic map construction

Prior to constructing a genetic map with the 22,378 SNPs, we filtered out SNP markers that fell in the following three categories. First, 3,915 markers (17.5%) were significantly distorted from the expected 1:1 Mendelian segregation pattern ($p < 0.05$, chi-square test). Second, 2,679 (12.0%) did not have a genotype call in over 30% of the DH population. Third, 286 (1.3%) had two or more nucleotides detected across 15% or more of the DH population. The R/ASMap package (Taylor & Butler, 2017) was used to identify the genetic map positions of the remaining 15,497 SNP markers.

Multiple QTL mapping (MQM) was performed on the genotypic least square means of each trait using MapQTL version 5 (Van Ooijen, 2006) using default parameters. A permutation test with 1000 permutations estimated the significance of locus-trait associations (alpha<0.05). Confidence intervals were estimated using QTL genetic map positions for which LOD scores are greater than the significance threshhold. R^2 is the proportion of genotypic variance explained by QTL. QTL were designated following the International Rules of Genetic Nomenclature (http://wheat.pw.usda.gov/ggpages/wgc/98/Intro.htm), which consisted of trait acronym, lab designation (ug= University of Guelph), and chromosome. Genetic maps and QTL graphs were drawn using MapChart v2.32 (Voorips et al. 2002).

References

Danecek P, Auton A, Abecasis G, Albers CA, Banks E, DePristo MA, et al. The variant call format and VCFtools. Bioinformatics. 2011;27(15):2156-8. Epub 2011/06/10. doi: 10.1093/bioinformatics/btr330. PubMed PMID: 21653522; PubMed Central PMCID: PMCPMC3137218.

Dobin A, Davis CA, Schlesinger F, Drenkow J, Zaleski C, Jha S, et al. STAR: ultrafast universal RNA-seq aligner. Bioinformatics. 2013;29(1):15-21. Epub 2012/10/30. doi: 10.1093/bioinformatics/bts635. PubMed PMID: 23104886; PubMed Central PMCID: PMCPMC3530905.

Krueger F. Trim Galore!: A wrapper tool around Cutadapt and FastQC to consistently apply quality and adapter trimming to FastQ files. 0.4; 2015.

Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, et al. The Sequence Alignment/Map format and SAMtools. Bioinformatics. 2009;25(16):2078-9. doi: 10.1093/bioinformatics/btp352

Ossowski S, Schneeberger K, Lucas-Lledo JI, Warthmann N, Clark RM, Shaw RG, et al. The rate and molecular spectrum of spontaneous mutations in Arabidopsis thaliana. Science. 2010;327(5961):92-4. Epub 2010/01/02. doi: 10.1126/science.1180677. PubMed PMID: 20044577; PubMed Central PMCID: PMCPMC3878865.

Taylor J and Butler DJ. R Package ASMap: efficient genetic linkage map construction and diagnosis. 2017.

Van Ooijen JJKB, Wageningen. JoinMap® 4, Software for the calculation of genetic linkage maps in experimental populations. 2006;33(10.1371).

Voorrips RE. MapChart: software for the graphical presentation of linkage maps and QTLs. J Hered. 2002;93(1):77-8. Epub 2002/05/16. PubMed PMID: 12011185.

Code and Protocols

SNP detection and SNP calling protocol
##Quality control and trimming
Fastqc and trim_galore were used for quality control and trimming.

```
fastqc RF1_rawRead_*.fq.gz

trim_galore --phred33 --fastqc --gzip --illumina --trim-n --paired -
    o path/to/the/output --
    length 75 RF1_rawRead_1.fq.gz RF1_rawRead_2.fq.gz
```

##Aligning reads and filtering aligned reads
STAR was used to generate genome index files and to map reads using IWGSC RefSeq v1.0 of Chinese Spring. We used the multi-sample 2-pass mapping approach described in Dobin (2016). The default value was used for all parameters except for one to which the number of allowed mismatches was 2 (-- outFilterMismatchNmax 2). The full STAR protocol is available at: https://chagall.med.cornell.edu/RNASEQcourse/STARmanual.pdf

Samtools view was used to extract uniquely mapped paired end reads and then picard to remove duplicate.

```
samtools view -h -q 255 -f 0x2 RF1_aligneRead.bam -o RF1_uniqMap.bam

java -Xmx32g jar picard.jar MarkDuplicates MAX_RECORDS_IN_RAM=20000000
INPUT= RF1_uniqMap.bam OUTPUT= RF1_markDup.bam METRICS_FILE=
    RF1_metrics.txt
```

Calling nucleotides at genomic sites
Samtools, mpileup and bcftools were used to identify nucleotides at genomic sites. Bcftools filter was used to extract any nucleotide having a read coverage of 4 or higher. Bcftool call was used to call nucleotides with sufficient read coverage. Vcftools was used to remove InDels.

```
samtools mpileup -u -t DP,DV,DPR -v -
    f genome.fasta RF1_sorted_MarkDupl.bam | bcftools filter -
    i 'DP>4' | bcftools call -m -o RF1_bases_DP4.vcf

vcftools --vcf RF1_bases_DP4.vcf --remove-indels --recode --recode-
    INFO-all --out RF1_bases_DP4_noIndels
```

Identifying and filtering SNPs

The InDel-free vcf files were processed in a custom R script to remove heterozygous sites and nucleotide calls with QUAL<20. The homozygous sites had genotype values corresponding to 0/0 and 1/1. The files that resulted from this analysis were called 'sample_homozygote_qual20.txt'. Then, the 'sample_homozygote_qual20.txt' files were imported in another R script to identify SNPs between Chinese Spring vs Red Fife, Chinese Spring vs Stettler, and Red Fife vs Stettler. Nucleotide bases needed to be called in at least 3 of the 4 parental replicates. A table with a list of SNP markers was created from this step.

SNP effect protocol
The InDel free vcf files were annotated using snpEff as described in:
http://snpeff.sourceforge.net/SnpEff_manual.html

Protocol to estimate gene expression levels
Assembling transcripts and estimating gene expression levels with Stringtie

We estimated transcript abundances for each gene using the four replicate samples from Red Fife and Stettler. The processed .bam files from the "# Filtering read alignments" step above was used as input into Stringtie **(?version)** which assigns reads to transcripts and genes. The program prepDE.py, provided by Stringtie authors, was used to extract genes' raw aligned read counts from Stringtie output files.

We constructed a list of genes expressed in both parents. A gene was considered expressed in a parent if it had at least 10 counts in at least one of the four parent replicates. This filter resulted in 52,682 expressed genes from the 110,791 genes with at least one mapped read across the eight samples.

Normalization of reads across parental lines was done using the median of ratios method using the estimatesizefactors() function from deseq2 Anders and Huber (2010). Each individual sample's raw count value was divided by that sample's normalization factor to generate normalized count values. These count values were used to compare genes' expression levels, as described in the manuscript text.

Protocol to classify genes according to their presence/absence on subgenomes
SynMap (Haug-Baltzell *et al.*, 2017), downloaded from the CoGe web platform (https://genomevolution.org), classified genes based on their presence/absence in the three wheat subgenomes. We used default values for parameters.

Protocol to randomly assign nucleotides to transcripts
We wrote an R Script in Bioconductor to obtain the expected distribution of effects from SNPs of the same transition/ transversion class as observed SNPs randomly placed on the genome. This script is below.

Code
Protocol to randomly sample nucleotides:

```
#Code processes 2 chromosomes at a time. Processing more than 2 chromosomes at a time was
slow. First, subset the genome.
#obtain the .gff for desired chromosomes
#scp
eraherso@iqaluk.sharcnet.ca:/home/eraherso/installers/snpEff/data/iwgsc_refseq_v1/genes.gff .
#genes.gff is derived from iwgsc_refseqv1.0_HighConf_2017Mar13.gff3

##extract from main gff file only the lines you will use from target chromosomes.
awk '{if ($1 ~ /chr[67][ABD]_part[12]/) print $0}' genes.gff > chr67ABD.gff

#library('BSgenome')
##forge a BSgenome object as described in package instructions
forgeBSgenomeDataPkg("/Users/lewislukens/Research/Wheat_genome/BSgenome.Taestivum.6
7ABD-seed.txt")

#in shell:
R CMD build BSgenome.Taestivumfour.IWGSC.v1
R CMD check BSgenome.Taestivumtfour.IWGSC.v1
R CMD INSTALL BSgenome.Taestivumfour.IWGSC.v1

#now capture the cds sequences from all genes
library(BSgenome)
wseq<-getBSgenome("BSgenome.Taestivumfour.IWGSC.v1")

library("GenomicFeatures")
#A set of tools and methods for making and manipulating transcript centric annotations.

txdb <-
makeTxDbFromGFF("/Users/lewislukens/Research/Wheat_genome/Elies_genome/Gff/chr67AB
D.gff", format="gff3", organism="Triticum aestivum")
cds <- cdsBy(txdb, by="tx", use.names=TRUE)
#Generates a  GRangesList
cds_seqs <- extractTranscriptSeqs(wseq, cds)
# Generates a DNAStringSet

#now, select genes that were expressed.
/Users/lewislukens/Research/Wheat_genome/Elies_genes
grep TraesCS[67][ABD]0 both_expressed.csv | cut -f1 -d ','| sed s/\"//g > 67ABD_genes

#in R
expressed<-
read.table("/Users/lewislukens/Research/Wheat_genome/Elies_genes/67ABD_genes")
```

```
#install.packages("stringr")
library("stringr")
t_expressed<-str_subset(names(cds_seqs),paste(expressed[,1],collapse="|"))
e_cds_seqs<-cds_seqs[t_expressed]
ts_len<-data.frame(names(e_cds_seqs),width(e_cds_seqs))
write.table(ts_len,file="ts_len",sep=",",quote=F,col.names=FALSE)

###used python to choose longest cds
#/Users/lewislukens/Research/Wheat_genome
python3 t_choose.py > longest_cds #file ts_len is input

###With R
longest<-read.table("longest_cds")
e_cds_long<- cds[longest [,1]]
#GRangesList object that is a subset of cds GRangesList object
e_range <-ranges(e_cds_long)
e_range_str<- sapply(e_range,toString)
cds_chrs<-sapply(runValue(seqnames(e_cds_long)),toString) #extract chromosomes to which
genes belong

e_cds_seqs_data<-getSeq(wseq,e_cds_long)
cds_string2<-sapply(as.list(e_cds_seqs_data),toString)
aa<-
data.frame(names(cds_string2),as.character(cds_string2),as.character(e_range_str),as.character(c
ds_chrs))

write.table(aa,file="67ABD_seqs_and_ranges",col.names=F)

##Now extract strand
tx <- transcriptsBy(txdb, by="gene")
gene_strand<-sapply(runValue(strand(tx)),toString)
write.table(data.frame(names(gene_strand),as.character(gene_strand)),file="67ABD_strand",quo
te=F,col.names=F)

###In Python:
python3 get_rev_comp.py > 67ABD_seqs_and_ranges_revcomp
#outputs seqs_and_ranges but for - strand takes reverse complement of nucleotides
python3 extract_seqs2.py > 67ABD_snp.vcf
#this code randomly selects SNPs at defined frequencies
#Output can be analyzed with SNPeff


##Code for extract_seqs2.py
#seqs_and_ranges
#The input data has on one line:
# a number
# longest CDs in quotes
# The sequence in quotes "ATGCCGGCCGCGCAGCAC..." ; separated by commas if more than
   one
# The range(s) for each of the sequences, comma separated e.g. "689807957-689808739,
   689807555-689807857"
# Deleted column header from original output from xxx

# The objective is to randomly generate set of mutations within these genes that match observed
   nucleotide changes.

#extracts the information and puts it into a data structure
#file=open("1A1B1D_seqs_and_ranges_revcomp")
#file=open("45ABD_seqs_and_ranges_revcomp")
file=open("67ABD_seqs_and_ranges_revcomp")
#file=open("temp")  #head -100 1A1B1D_seqs_and_ranges_revcomp >temp

seq_pos=[]  #the list of all the sequences' positions
seq_nuc=[]  #the list of all the nucleotides from all the cds transcripts
seq_chr=[]  #the chromosome

#import re
import itertools
import random

for line in file:

  data= line.rstrip().split("\" \"") #splits on " " which separates the columns
  data[4]=data[4].rstrip("\"")

  #concatenate cds exons into a sequence list
  seqs= data[2].split(", ")  #removes the commas separating exon sequences; returns a list of exon
    sequences
  seqs[len(seqs)-1]= seqs[len(seqs)-1].rstrip("\"") #removes the trailing " from the last exon
    sequence#print (data[1]) #the transcript name
  #print(seqs[len(seqs)-1])  #the last exon of each transcript

  for i in seqs:  #i loops through exons; starts at first exon and ends at final exon. Appends nucs to
    the master list
    #seq_list=list(seqs[len(seqs)-1])
    seq_list=list(i)  #makes a list from exon in question
    seq_nuc.extend(seq_list)

  #now append each nucleotide position to the list for the nucleotides positions. data[3] has range
  ranges= data[3].rstrip("\"").split(", ")  #a list of one or more ranges for the transcript ; "x-y"
  length = 0  # initialize the variable for the ranges
  for i in ranges:  #loop through two-number ranges associated with each transcript exon
    rng=i.split("-") #for each range, make a list with range start and stop
    #TraesCS1A01G053600.1 has an exon with a single CDS, so no end point. "35528837-
     35528996, 35529295-35530453, 35532835"
    if (len(rng) < 2 ):
      rng.append(rng[0])

    seq_pos.extend(list(range(int(rng[0]), int(rng[1])+1))) #add to master list

    #here also add list of chromosome name and append to master list
    length=len(range(int(rng[0]), int(rng[1])+1))  #length of sequence
    seq_chr.extend(list(itertools.repeat(data[4],length)))

#now we summarize the three lists, having looped through lines and exons. These match as
     expected.
#print ("positions " + str(len(seq_pos)))
#print ("nucleotides " + str(len(seq_nuc)))
#print ("chromosome " + str(len(seq_chr)))
#print (seq_pos)
#print (seq_nuc)

#print a vcf
#print #CHROM     POS    ID     REF    ALT    QUAL FILTER      INFO
#22     17071756      .      T      C      .      .      .
#22     17072035      .      C      T      .      .      .
#22     17072258      .      C      A      .      .      .

#Now randomly selet nucleotides
#sample n
n=26609

#define the numbers of observed nucleotide changes
#A_C 8; A_G 24; A_T 2  34 As  0.2394366
#C_A 4; C_G 7; C_T 18  29 Cs  0.2042254
#G_A 33; G_C 9; G_T 6  48 Gs  0.3380282
#T_A 4; T_C 18; T_G 9  31 Ts  0.2183099
#142 total

hv={} #number of nucleotides sampled per nucleotide
wt={} #number of nucleotides wanted per nucleotide

hv.update({'A': 0, 'C': 0, 'G': 0, 'T': 0})
#hv['A']=0 #counts in each class A,G,C,T
#wt['A']=int(0.2394366*n)  #counts of A that you want- wt
wt.update({'A' : int(0.2394366*n), 'C' : int(0.2042254*n), 'G' : int(0.3380282*n), 'T' :
     int(0.2183099*n)})  #counts of A that you want- wt
```

```
done=0
while (done==0):
    a= random.randint(0, len(seq_nuc)-1) #get random position

    if (seq_nuc[a]=='A' and (hv['A']<wt['A'])):
        rnd=random.random()  #randomly choose nt change
        if(rnd<=(2/34)):
            print(seq_chr[a] + "\t" + str(seq_pos[a]) + "\t.\t" + seq_nuc[a] + "\tT\t.\t.\t.")
        elif ((rnd<=10/34) and rnd>=(2/34)):
            print(seq_chr[a] + "\t" + str(seq_pos[a]) + "\t.\t" + seq_nuc[a] + "\tC\t.\t.\t.")
        else:
            print(seq_chr[a] + "\t" + str(seq_pos[a]) + "\t.\t" + seq_nuc[a] + "\tG\t.\t.\t.")
        hv['A']=hv['A']+1
        #print ("Have A is " + str(hv['A']))

    if (seq_nuc[a]=='C' and (hv['C']<wt['C'])):
        rnd=random.random()  #randomly choose nt change
        if(rnd<=(4/29)):
            print(seq_chr[a] + "\t" + str(seq_pos[a]) + "\t.\t" + seq_nuc[a] + "\tA\t.\t.\t.")
        elif ((rnd<=11/29) and rnd>=(4/29)):
            print(seq_chr[a] + "\t" + str(seq_pos[a]) + "\t.\t" + seq_nuc[a] + "\tG\t.\t.\t.")
        else:
            print(seq_chr[a] + "\t" + str(seq_pos[a]) + "\t.\t" + seq_nuc[a] + "\tT\t.\t.\t.")
        hv['C']=hv['C']+1

    if (seq_nuc[a]=='G' and (hv['G']<wt['G'])):
        rnd=random.random()  #randomly choose nt change
        if(rnd<=(33/48)):
            print(seq_chr[a] + "\t" + str(seq_pos[a]) + "\t.\t" + seq_nuc[a] + "\tA\t.\t.\t.")
        elif ((rnd<=42/48) and rnd>=(33/48)):
            print(seq_chr[a] + "\t" + str(seq_pos[a]) + "\t.\t" + seq_nuc[a] + "\tC\t.\t.\t.")
        else:
            print(seq_chr[a] + "\t" + str(seq_pos[a]) + "\t.\t" + seq_nuc[a] + "\tT\t.\t.\t.")
        hv['G']=hv['G']+1

    if (seq_nuc[a]=='T' and (hv['T']<wt['T'])):
        rnd=random.random()  #randomly choose nt change
        if(rnd<=(4/31)):
            print(seq_chr[a] + "\t" + str(seq_pos[a]) + "\t.\t" + seq_nuc[a] + "\tA\t.\t.\t.")
        elif ((rnd<=22/31) and rnd>=(4/31)):
            print(seq_chr[a] + "\t" + str(seq_pos[a]) + "\t.\t" + seq_nuc[a] + "\tC\t.\t.\t.")
        else:
            print(seq_chr[a] + "\t" + str(seq_pos[a]) + "\t.\t" + seq_nuc[a] + "\tG\t.\t.\t.")
        hv['T']=hv['T']+1

    if ((wt['A']==hv['A']) and (wt['C']==hv['C']) and (wt['G']==hv['G']) and (wt['T']==hv['T'])):
```

done=1

Sliding window code

```
# Identical By Descent (IBD) regions
# Identify IDB chromosome regions using different criteria (windows of i
genes every j steps)
# i is equal to 100, 200, 300, ..., 1000; and j or step = 5
# Theorically, an IBD region is a portion of chr that Stettler is
inherited from Red Fife (its ancestor).
# Practically, an IBD region corresponds to a region (a window) of chr
that has two snp markers or less.
# snp marker number = 2 is defined as a threshold (see the graph section,
yintercept = log2(2) = 1)
# R script by Elie Raherison - May 24, 2020
######################################################

# 1- Load libraries
#######################################################
#install.packages('zoo')
library(zoo)
library(dplyr)
library(ggplot2)
library(tidyr)

# 2- Define the main path
#######################################################
mainPath='C:/Users/m_rah/Desktop/Article/G3-paper_Review/G3-Comments_05-
23-2020'
setwd(mainPath)

# 3- (TAB #1) Load files that have the number of SNPs per gene
#######################################################
# rs: SNP markers (Red Fife vs Stettler)
# ------------------------------------
pos_rs <-
read.table(file="15_1_count_SNPsperGene_346SNPtypes_expGenes_Markers.txt",
sep="\t",header=T);
#> head(pos_rs,3)
#            wheat_id   chr  start    end geneLength count nSNPper100b snp
#1 TraesCS1A01G000400 chr1A 121263 121559        297     0           0   0
#2 TraesCS1A01G000600 chr1A 157734 163815       6082     0           0   0
#3 TraesCS1A01G001500 chr1A 326099 326518        420     0           0   0
#> dim(pos_rs)
#[1] 50236     8

# mp: marker SNPs that are mapped to the genetic map
# ------------------------------------
pos_mp <-
read.table(file="15_1_count_SNPsperGene_346SNPtypes_mappedSNP.txt",sep="\t
",header=T);
#> head(pos_mp,3)
#            wheat_id   chr count    start      end geneLength nSNPper100b
snp
#1 TraesCS1A01G020500 chr1A     2 10067063 10069144       2082  0.09606148
mappedMarker
```

```
#2 TraesCS1A01G020600 chr1A     5 10069852 10072312       2461  0.20316944
mappedMarker
#3 TraesCS1A01G020900 chr1A     6 10132632 10133075        444  1.35135135
mappedMarker
#> dim(pos_mp)
#[1] 5714    8

# 4- (TAB #2) Load the file with the list of genes that have SNPs
#######################################################
# List of genes which were sorted in ascending order of their start
position
all_marks <- read.table("unique_markers",sep="\t")
names(all_marks) <- c("wheat_id","chr","start")
#> head(all_marks,3)
#            wheat_id   chr  start
#1 TraesCS1A01G000100 chr1A  58508
#2 TraesCS1A01G000300 chr1A 104607
#3 TraesCS1A01G000400 chr1A 121263
#> dim(all_marks)
#[1] 54392     3

all_marks_1 <- all_marks[!all_marks$wheat_id=="wheat_id",] # remove
headers (eg #54392 wheat_id chr    NA)
all_marks_1$start <- as.numeric(as.character(all_marks_1$start)) #convert
factor to numeric
all_marks_1$wheat_id <- as.character(all_marks_1$wheat_id)
#> head(all_marks_1,3)
#            wheat_id   chr  start
#1 TraesCS1A01G000100 chr1A  58508
#2 TraesCS1A01G000300 chr1A 104607
#3 TraesCS1A01G000400 chr1A 121263
#> dim(all_marks_1)
#[1] 54391     3

# 5- Merge TAB #1 and #2 to assign the number of SNPs per genes
#######################################################
# rs: SNP markers (Red Fife vs Stettler)
# ---------------------------------------
pos_rs$wheat_id <- as.character(pos_rs$wheat_id)
pos_rs_all <- left_join(all_marks_1, pos_rs, by = c('wheat_id','start'))
names(pos_rs_all)[2] = "chr"  #converts chr.x to chr
pos_rs_all$count[is.na(pos_rs_all$count)] = 0 #assigns "NA" snps to zero
#> dim(pos_rs_all)
#[1] 54391     9

# mp: marker SNPs that are mapped to the genetic map
# ---------------------------------------
pos_mp$wheat_id <- as.character(pos_mp$wheat_id)
pos_mp$snp <- as.numeric(pos_mp$snp)
pos_mp_all <- left_join(all_marks_1, pos_mp, by = c('wheat_id','start'))
names(pos_mp_all)[2] = "chr"  #converts chr.x to chr
pos_mp_all$count[is.na(pos_mp_all$count)] = 0 #assigns "NA" snps to zero
#str(pos_mp_all)
#> head(pos_mp_all)
```

```
#            wheat_id   chr  start chr.y count end geneLength nSNPper100b
snp
#1 TraesCS1A01G000100 chr1A  58508  <NA>    NA  NA         NA          NA
0
#2 TraesCS1A01G000300 chr1A 104607  <NA>    NA  NA         NA          NA
0
#3 TraesCS1A01G000400 chr1A 121263  <NA>    NA  NA         NA          NA
0
#> dim(pos_mp_all)
#[1] 54391     9

# 6- Remove all SNPs in the unknown chromosome
#######################################################
# rs: SNP markers (Red Fife vs Stettler)
# ------------------------------------
pos_rs_all <- pos_rs_all %>%
      filter(chr != "chrUn")
#> head(pos_rs_all,3)
#            wheat_id   chr  start chr.y    end geneLength count
nSNPper100b snp
#1 TraesCS1A01G000100 chr1A  58508  <NA>     NA         NA    NA
NA   0
#2 TraesCS1A01G000300 chr1A 104607  <NA>     NA         NA    NA
NA   0
#3 TraesCS1A01G000400 chr1A 121263 chr1A 121559        297     0
0   0
#> dim(pos_rs_all)
#[1] 53772    10
pos_rs_all$count = as.numeric(pos_rs_all$count)

# mp: marker SNPs that are mapped to the genetic map
# ------------------------------------
pos_mp_all <- pos_mp_all %>%
      filter(chr != "chrUn")
pos_mp_all$count = as.numeric(pos_mp_all$count)
#> dim(pos_mp_all)
#[1] 53772    10

# 7- A loop to count the SNPs in a window of n genes
#######################################################
windows = seq (100, 1000, by=100) # number of genes in each window
#> windows
# [1]  100  200  300  400  500  600  700  800  900 1000

for (i in windows){
      #i=200

#######################################################
      window <- i  # for i = 100 means 'take 100 genes at a time and count
the total number of SNPs in these genes'

#######################################################
      step <- 5 # Decrease this number if you want get smoother curves
      # step = 5;
      # 1 --> 100 genes
      # 5 --> 105 genes
      # 10 --> 110 genes
      # ...

      # 7-2 Calculate the number of SNPs per window and every step using
the rollapply function
      # ------------------------------------------------------------------
----------------------
      # rs: SNP markers (Red Fife vs Stettler)
      # --------------------------------------
      snp_win_rs <- pos_rs_all %>%
            arrange(desc(start)) %>%
                  group_by(chr) %>%
                        do(data.frame(
                              window.start = rollapply(.$start, width =
window, by = step, FUN = min, align = "left", fill = NA, partial = TRUE),
                              window.end = rollapply(.$start, width =
window, by = step, FUN = max, align = "left", fill = NA, partial = TRUE),
                              snp_num = rollapply(.$count, width =  window,
by = step, FUN = sum, align = "left", fill = NA, partial = TRUE)))
                              #genes with snps

      # mp: marker SNPs that are mapped to the genetic map
      # --------------------------------------
      snp_win_mp <- pos_mp_all %>%
            arrange(desc(start)) %>%
                  group_by(chr) %>%
                        do(data.frame(
                              window.start = rollapply(.$start, width =
window, by = step, FUN = min, align = "left", fill = NA, partial = TRUE),
                              window.end = rollapply(.$start, width =
window, by = step, FUN = max, align = "left", fill = NA, partial = TRUE),
                              snp_num = rollapply(.$count, width =  window,
by = step, FUN = sum, align = "left", fill = NA, partial = TRUE)))

      #########################################################

      # 7-3 Remove all rows with NA value and merge all tables in one
      # ------------------------------------------------------------------
----------------------
      snp_win_rs2 = snp_win_rs %>%
            drop_na(window.start)

      snp_win_mp2 = snp_win_mp %>%
            drop_na(window.start)

      colnames(snp_win_rs2)[4] = "rs"
      colnames(snp_win_mp2)[4] = "mp"

      # Merge tables
      all_tab=Reduce(function(...) merge(..., all = TRUE,
by=c("chr","window.start","window.end")),
               list(snp_win_rs2, snp_win_mp2))
```

```
	# Rename columns
	all_tab1 = all_tab
	names(all_tab1)[4:5] = c("marker", "mappedMarker")

	# Check duplicates
	# dup <-
all_tab[which(duplicated(all_tab1[,c('chr','window.start')])==T),]
	#> dim(dup)
	#[1] 3759    7

	# 7-4 Remove duplicates (chr and window.start). Keep only the row
with the max value in window.end
	# -----------------------------------------------------------------
----------------------
	all_tab10 = all_tab1 %>%
		group_by(chr, window.start) %>%
			top_n(1, window.end)

	# Create a column with log2 values
	all_tab10$marker_log2 = log(all_tab10$marker,2)  # Red Fife vs
Stettler or markers
	all_tab10$mappedMarker_log2 = log(all_tab10$mappedMarker,2)  #
mapped markers

	# Assign 0 to any undefined value [log2(0) == undefined]
	all_tab10$marker_log2 = ifelse(all_tab10$marker ==
0,0,all_tab10$marker_log2)
	all_tab10$mappedMarker_log2 = ifelse(all_tab10$mappedMarker ==
0,0,all_tab10$mappedMarker_log2)

	# Export tables
	#################################################################
#######
	write.table(all_tab10,file=paste("15_8_IBD_per_",window,"_genes_per"
,step,"_steps.txt",sep=""),row.names=FALSE,col.names=TRUE,sep="\t")
	#################################################################
#######

	# 7-5 Print the result (criteria and number of the IBD regions) on
the screen and in a pdf file
	# -----------------------------------------------------------------
----------------------
	interval <- function(x,y){
		lx <- length(x)
		ly <- length(y)
		n <- max(lx,ly)
	  as.vector(rbind(rep(x, length.out=n), rep(y, length.out=n)))}
	# NB log2 allows to separate out the curves from different types of
SNPs.
	# The problem is that log(0) is undefined # NEED TO BE CHECKED
################################################

	# Graphs with log2 values
```

```
      siz=1
      n=2
      my_breaks = seq(0,800,50)
      my_labels <- interval(seq(0,800,by=100), "") [-18]
      graph=ggplot(all_tab10, aes(x = window.start/1000000)) +
            # mp: marker SNPs that are mapped to the genetic map
            geom_line(aes(y = all_tab10$mappedMarker_log2), size=siz,
color="green")+
            # marker: SNP markers (Red Fife vs Stettler)
            geom_line(aes(y = all_tab10$marker_log2), size=0.3) +
            facet_wrap(~ chr, scales = "free_x", nrow = 7) +
            guides(linetype = FALSE) +
            theme_classic(base_size = 10) +
            #theme(axis.line = element_line (colour = "black", size = 1,
linetype = "solide"))+
            theme(axis.text=element_text(size=8),
axis.title=element_text(size=9,face="bold")) +
            #theme(axis.text.x = element_blank()) +
            #theme(axis.ticks.x = element_blank()) +
            geom_hline(yintercept = 1, linetype="dotted", colour="grey") +
# snp number = 2 (threshold)
            scale_x_continuous(breaks = my_breaks, labels = my_labels) +
            #scale_x_continuous(breaks = equal_breaks(s=50)) +
            #theme(axis.title.x=element_text("hello there"))
            labs( x = "Position (Mbp)", y = paste("log2 (Number of SNPs
per ",window," genes)", sep=""))

      # Export the graph
      ######################################################################
#######
      pdf(paste("15_8_plot_1MB_allChrom_Figure_log", n, "_", step,
"_steps_nbSNPper_",window,"_Genes.pdf", sep=""))
      print(graph)
      dev.off()
      ######################################################################
#######
}
################################## END
##############################################
################################## END
##############################################
################################## END
##############################################
```